České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

BAKALÁŘSKÁ PRÁCE

Využití robota LEGO MINDSTORMS při výuce předmětu A3B99RO Roboti

Praha, 2010 **Autor: Dan Martinec**

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne _____ _____
podpis

Poděkování

Tímto bych chtěl poděkovat vedoucímu bakalářské práce Ing. Martinu Hlinovskému Ph.D. za poskytnutí přípravků k úlohám a zapůjčení LEGO setů. Dále děkuji svým rodičům za jejich lásku, trpělivost a porozumění a také za to, že jsou těmi nejskvělejšími rodiči, které si dovedu představit. Rovněž děkuji svým bratrům, kteří jsou a zůstanou nejen mými vzory, ale i těmi nejlepšími přáteli.

Nevděčný je ten, kdo vrací dobrodiní bez úroku.

Seneca

Abstrakt

Tato práce pojednává o využití LEGO MINDSTORMS při výuce předmětu A3B99RO Roboti. Seznamuje s možnostmi využití stavebnice a podrobně rozebírá programování robota v prostředí LeJOS-NXJ. Dává instrukce jak provést instalaci a na několika příkladech ukazuje základní funkce, které toto prostředí nabízí. Součástí práce je i návrh a řešení dvou úloh, které mohou být použity jako semestrální úlohy pro zmíněný předmět. Závěrečná část práce se věnuje webovým stránkám, které budou přidány k existujícím stránkám k tomuto předmětu.

Abstract

This thesis discusses the usage of LEGO MINDSTORMS as a teaching tool in the lecture course A3B99RO Robots. It introduces the basic features of this robotics kit and explains in details programming of the robot in the environment LeJOS-NXJ. It gives the instructions of how to install the programming environment for LEGO MINDSTORMS on a PC and shows the basic functions offered by this environment. A part of this thesis is devoted to the designs of two projects and their solutions. The projects can be assigned as a term assignements in the course. The final part deals with constructing web pages on the thesis subject that will be included to the current web pages on this course.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Dan Martinec**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Využití robota LEGO MINDSTORMS při výuce 2 předmětu A3B99RO Roboti**

Pokyny pro vypracování:

1. Seznamte se s možnostmi robota LEGO Mindstorms (současný stav, HW a SW vybavení).
2. Nastudujte možnosti programování robota LEGO Mindstorms a připravte prezentaci v powerpointu a návod v pdf formátu a ve formě webových stránek pro LeJOS-NXJ.
3. Proveďte návrh jedné až dvou nových soutěžních úloh s řízením ve třech různých programovacích prostředích (NXT-G, NXC a LeJOS-NXJ) včetně dokumentace s konstrukcí robota a způsobu jeho programování.
4. Vytvořte webové stránky k navrženým soutěžním úlohám.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Martin Hlinovský, Ph.D.

Platnost zadání: do konce zimního semestru 2010/2011

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 30. 10. 2009

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Hardwarové vybavení

Hardwarové vybavení stavebnice *LEGO Mindstorms* je poměrně rozsáhlé. V základním balení *LEGO Mindstorms NXT* najdeme následující součástky:

- Inteligentní NXT kostku
- Dva dotykové senzory
- Jeden ultrazvukový senzor
- Jeden světelný senzor
- Tři interaktivní servomotory s rotačními senozory pro přesnější řízení
- Sedm šesti žílových spojovacích kabelů typu RJ-12

Nyní tento hardware popíšeme podrobněji.

1.1 Inteligentní NXT kostka

Obrázek 1.1: NXT inteligentní kostka

NXT kostka je základní řídící jednotkou robota. Obsahuje 32-bitový procesor, běžící na frekvenci 48MHz s 256KB flash paměti a 64KB RAM paměti. Výhoda této flash paměti je v tom, že do ní můžeme nahrát několik souborů (programů), které v ní zůstanou uloženy i po vypnutí NXT kostky.

Pro nahrání programu do NXT kostky můžeme využít konektor USB 2.0 nebo bezdrátový přenos pomocí technologie Bluetooth.

Pro připojení k jednotlivým motorům jsou k dispozici 3 výstupní porty označené A, B a C. Výstupní signál do motoru je signál PWM, kterým řídíme napětí dodávané do motoru.

NXT kostka je vybavena čtyřmi vstupními porty, do kterých se nechají připojit jednotlivé senzory.

Pro zobrazování zpráv uživateli je použit maticový LCD displej s rozměry 100x64 pixelů.

1.2 Senzory

Senzor je jednou z nejdůležitějších komponent každého robota. Pomocí senzorů může robot v reálném čase příjimat informace ze svého okolí a následně na ně reagovat. Robot

se tak může stát částečně nebo plně autonomní. Autonomní robot je takový robot, který se podle přijatého signálu sám rozhodne, jak se má zachovat.
V základním balení *LEGO Mindstorms NXT* nalezneme následující typy senzorů.

1.2.1 Dotykový senzor

Jedná se o nejjednodušší senzor stavebnice. Výstup je dvoustavový. Pokud je oranžová pohyblivá část senzoru stlačena, tak výstup odpovídá stavu stlačeno. V opačném případě je výstup ve stavu nestlačeno. Tento senzor je výhodné použít jako nárazník, který bude detekovat překážky, na které senzor narazí.

Obrázek 1.2: Dotykový senzor

1.2.2 Světelný senzor

Světelný senzor měří světelnou intenzitu. Obsahuje červenou LED diodu, kterou je možné využít jako zdroj světla, a dále fototranzistor pro snímání intenzity světla. Jedním z podstatných rozdílů oproti dotykovému senzoru je skutečnost, že výstup není pouze dvoustavový, ale hodnota v rozsahu od 0 do 1023. Zdůrazněme, že senzor nevnímá barvy jako lidské oko, ale pouze měří světelnou intenzitu. Barvu povrchu senzor zjistí díky tomu, že každá barva odráží jiné množství světelné intenzity.
Další zajímavou věcí je, že fototranzistor umístěný v senzoru, je mnohem citlivější na infračervené záření než lidské oko. Jeho viditelné spektrum je od 400 nm do 1150 nm.
Světelný senzor je zvláště užitečný v úloze "sledování čáry".

Obrázek 1.3: Světelný senzor

1.2.3 Ultrazvukový senzor

Ultrazvukový senzor je jeden z nejužitečnějších senzorů, které ve stavebnici najdeme. Umožňujeme měřit vzdálenost k objektu, na který je nasměrován. Princip je následující. Senzor vyšle ultrazvukový signál na frekvenci 40 kHz, který lidské ucho nezaznamená. Ten se odrazí od překážky zpět k senzoru, který signál zachytí a na základě jeho doby šíření vypočítá vzdálenost k objektu. Poznamenejme, že pomocí stejného principu se orientují mnohá zvířata v přírodě, např. netopýři či delfíni.
Senzor je schopen změřit vzdálenost od 0 cm do 250 cm s přesností +/- 3 cm.
Tento senzor se především uplatní ve chvíli, kdy je potřeba vyhnout se překážkám na trati.

Obrázek 1.4: Ultrazvukový senzor

1.2.4 Zvukový senzor

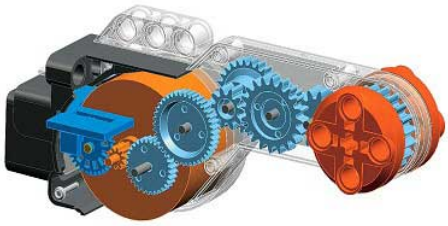
Pomocí zvukového senzoru můžeme měřit různé úrovně hlasitosti zvuku v decibelech (dB). Rozsah senzoru je $< 0,90\ dB >$.

Obrázek 1.5: Zvukový senzor

1.3 NXT Motor

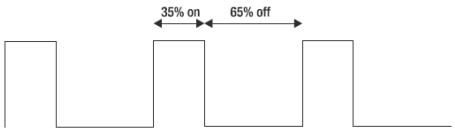
NXT motor je v mnohém lepší oproti starším LEGO motorům. Stejnosměrné motory využívají napětí 9V a mohou pohánět otočnou část motoru rychlostí až 1500 otáček za minutu. Aby NXT motor dokázal udržet i pomalejší rychlost, tak využívá systém několika ozubených kol, které fungují jako převody.

Obrázek 1.6: NXT motor



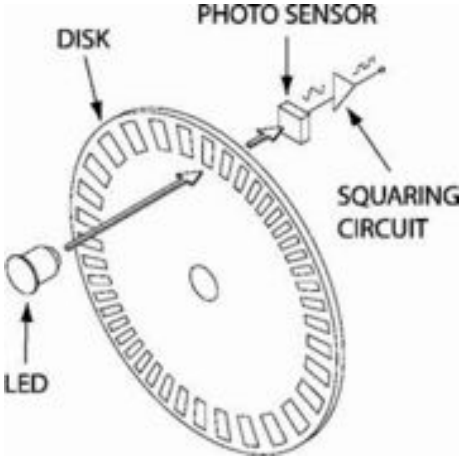
Obrázek 1.7: Schéma NXT motoru

Rychlost motoru ovládáme pulzní šírkovou modulací (PWM). Pomocí tohoto signálu se řídí průměrné napětí, kterým se napájí motor. Na obrázku 1.8 vidíme průběh PWM signálu pro 35% úroveň dodávané energie.



Obrázek 1.8: PWM signál pro 35% úroveň

Další důležitá součást motoru je rotační senzor, který umožňuje velmi přesné otočení a tím i přesný pohyb robota. Princip rotačního senzoru je následující. LED dioda emituje infračervené paprsky světla, které procházejí skrz kotouč spojený s motorem. Na kotouči jsou pravidelně rozmístěny štěrbiny, které propouštějí světlo. Za štěrbinami jsou umístěny fototranzistory, které detekují světlo a podle impulzů, které takto vznikají, lze určit o kolik se motor otočil.



Obrázek 1.9: Princip rotačního senzoru

Kapitola 2

Programování robota

NXT kostku můžeme programovat v mnoha různých jazycích. Jejich hlavní odlišnosti popisuje tabulka 2.1

	NXT-G	**NXC**	**RobotC**	**LeJOS NXJ**	**Matlab**
Jazyk	Grafický	Not-exactly C	C	JAVA	RWTH – Mind-storms NXT Toolbox
Platforma	Windows, MAC OS	Windows, Linux, MAC OS	Windows	Windows, Linux, MAC OS	Windows
IDE	Ano	Ano	Ano	plugin do Eclipse a NetBeans	Ano
Více-vláknové aplikace	Ano	Ano	Ano	Ano	Ano
Obsluha událostí	Ne	Ne	Ano	Ano (JAVA events)	Ano
Podpora datového typu float	Ne	Ne	Ano	Ano	Ne
Podpora Bluetooth	Ano	Ano	Ano	Ano	Ano

Tabulka 2.1: Přehled programovacích prostředků

Dále se budeme věnovat prostředí LeJOS-NXJ.

2.0.1 Výhody

- Mnoho předpřipravených metod, které usnadňují a urychlují programování.
- Programovací prostředí NetBeans, které lze volně stáhnout na internetu. Pokud je nastaveno správně, tak jeho nápověda umožňuje pohodlnější psaní programu.
- Možnost objektového programování ve standartním jazyce JAVA.
- Podpora více platforem.
- Podpora datového typu float.
- Podpora obsluhy událostí.
- ...a mnoho dalšího na internetové stránce [1].

2.0.2 Nevýhody

- Komplikovaná instalace.
- Nutnost základní znalosti programování.

2.1 Návod na instalaci

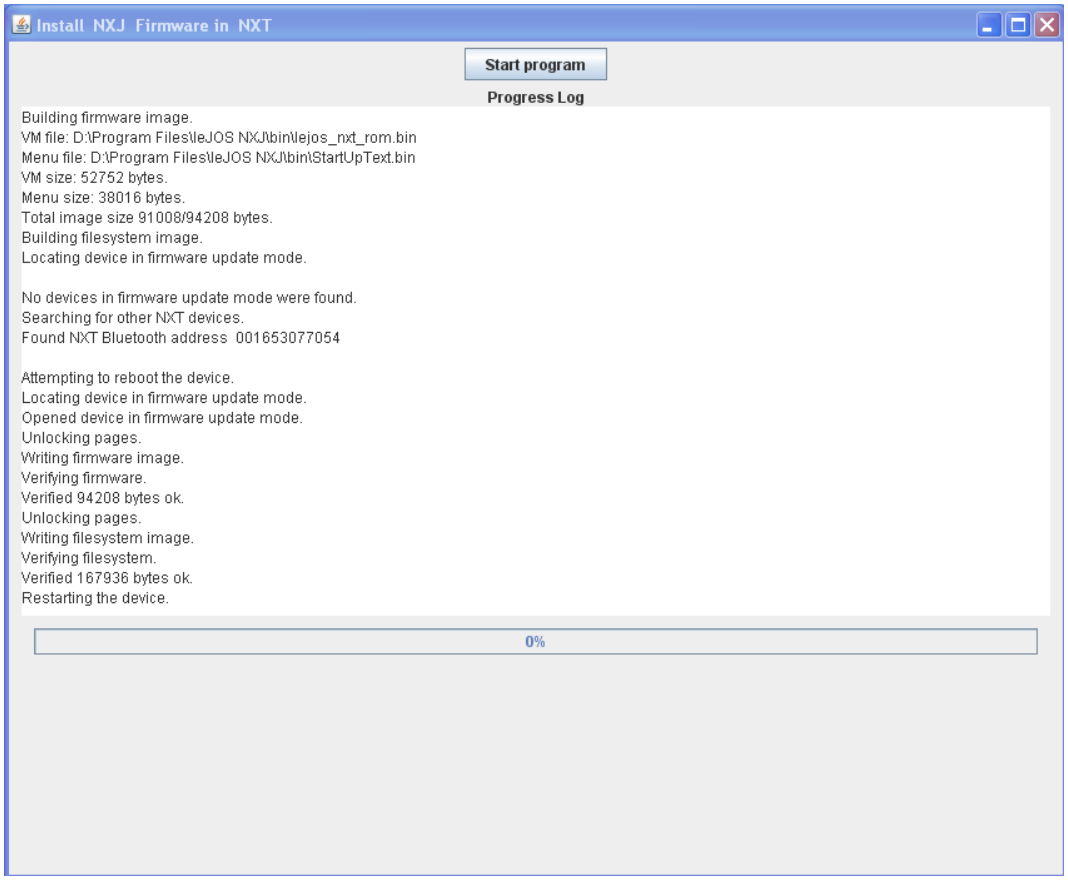
1. Nainstalujte ovladač na připojení NXJ přes USB rozhraní. Pokud jste nainstalovali software od LEGO Mindstorms, který se k legu přikládá, tak se již tento ovladač nainstaloval a tento krok se přeskočí. Pokud nechcete instalovat software od LEGO Minstorms, tak lze ovladač stáhnout z webových stránek [2].
2. Dále bude potřeba mít nainstalovaný Java Development Kit (JDK). Nejnovější JDK lze stáhnout z webových stránek [3]. Doporučuje se mít nainstalovanou alespoň verzi JDK 1.6.
   Po instalaci se ujistěte, že se správně nastavily proměnné prostředí. Ty zajistí, že další programy "uvidí", že je již nainstalovaný JDK. Proměnné prostředí můžete editovat v sekci *Ovládací panely* → *Systém* → *Upřesnit* → *Proměnné prostředí*. Měly by být nastaveny takto:

Proměnná	**Hodnota**	**Příklad**
JAVA_HOME	Cesta k nainstalovanému JDK	C:\Program Files\Java\jdk1.6.0_16
PATH	Cesta k *bin* složce nainstalovaného JDK	C:\Program Files\Java\jdk1.6.0_16\bin

Tabulka 2.2: Proměnné prostředí

3. Nyní je potřeba stáhnout LeJOS NXJ software. Ten lze stáhnout z webové stránky [4]. Po stažení spusťte instalační soubor a řiďte se instrukcemi.

4. Po dokončení instalace se objeví okno, pomocí kterého můžete do NXJ nahrát nový firmware. (Stejné okno můžete vyvolat příkazem *nxjflashg*.)



Obrázek 2.1: Flash firmware

Ujistěte se, že NXT je připojen přes USB k počítači a stiskněte tlačítko *Start program*

5. Po úspěšném flashnutí firmwaru se zobrazí nové okno (obrázek 2.2) a na displeji NXT se bude při spouštění zobrazovat text *LEJOS*.

Obrázek 2.2: Nové okno na konci flashování

2.2 Kompilace a nahrání programu do NXT

Programy pro NXT se píšou v jazyce JAVA. Takové programy lze psát v jednoduchém textovém prohlížeči a poté je pomocí příkazové řádky přeložit do JAVA formátu. Alternativou je použít vývojové prostředí. Tím se získá soubor typu .java. Aby takovýto soubor mohl být nahrán do NXT, je nutné ho zkompilovat (převést .java do .nxt). Tato operace lze provést více způsoby, asi nejjednodušší je použití příkazové řádky.

2.3 Příkazová řádka

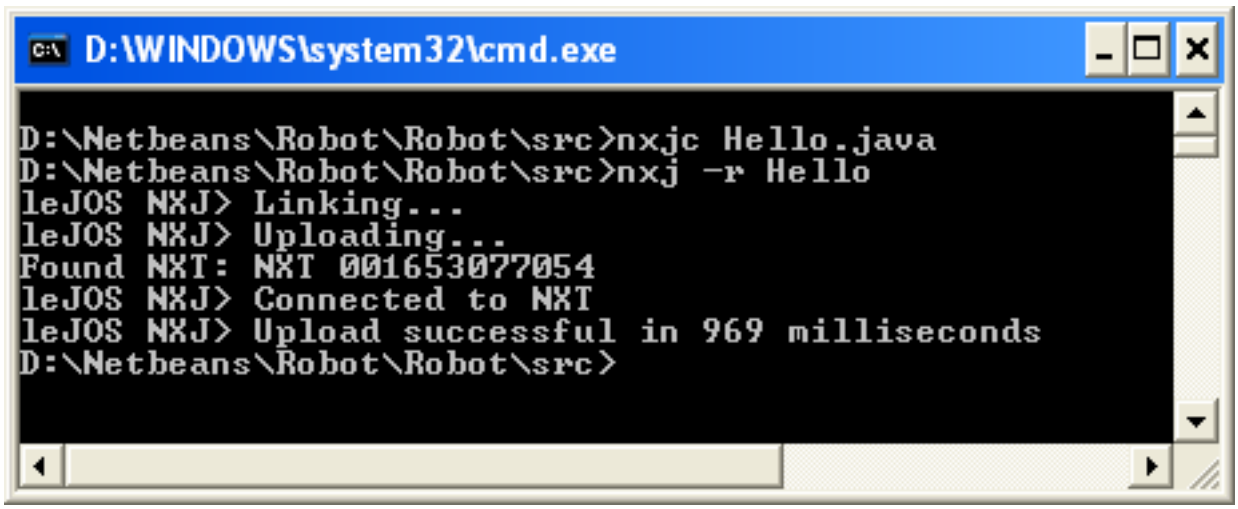
Existují 4 příkazy: *nxjc*, *nxjlink*, *nxjupload*, *nxj*

nxjc *soubor.java* Zkompiluje se soubor.java. Vznikne *soubor.class*

nxjlink *soubor_class* -o *soubor.nxj* Zavolá se leJOS NXJ linker. Ze souboru *soubor_class.class* se vytvoří *soubor.nxj*. Tento soubor se již může nahrát do NXT.

nxjupload *soubor.nxj* Nahraje *soubor.nxj* do NXT.

nxj -r *soubor_class* Tento příkaz je kombinací předchozích dvou. Ze souboru *soubor_class.class* vytvoří *soubor.nxj*, který následně nahraje do NXT. Pokud je zapnutý přepínač *-r*, tak se tento nahraný program ihned po nahraní do NXT spustí.



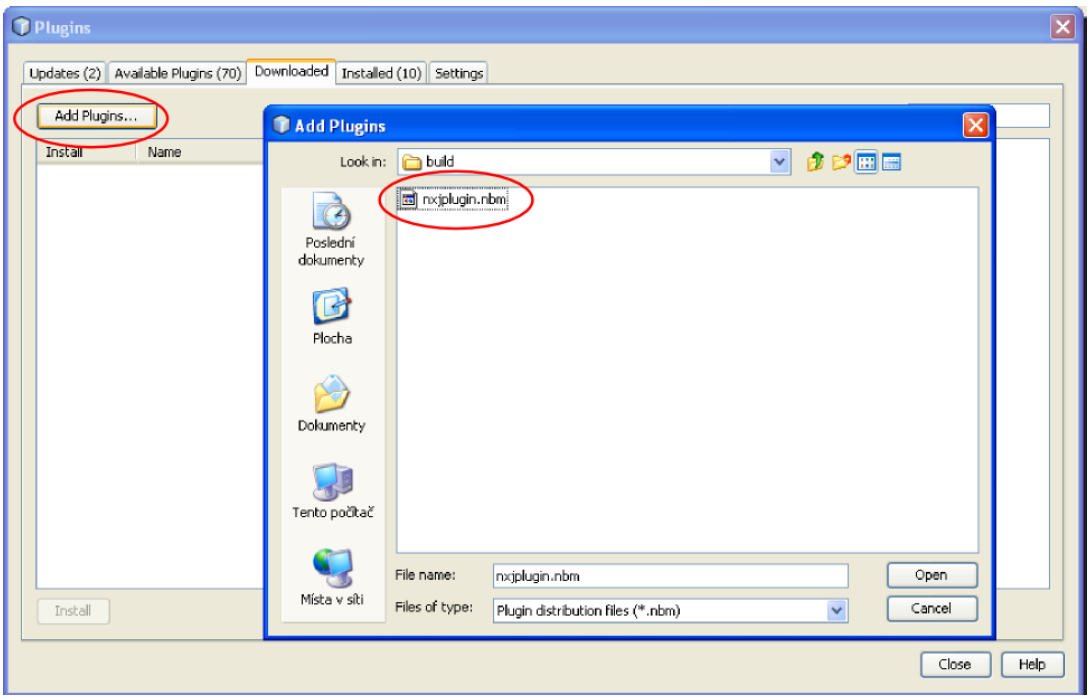
Obrázek 2.3: Ukázka použití příkazů *nxjc* a *nxj*

2.4 Vývojové prostředí NetBeans

Tento způsob je na nastavení o něco složitější, ale výhoda je, že pokud vše správně nastavíme, bude stačit kliknout na jediné tlačítko a kompilace, linkování a nahrání do NXT kostky se provede najednou.
Vývojové prostředí NetBeans, lze stáhnout z internetových stránek [5].

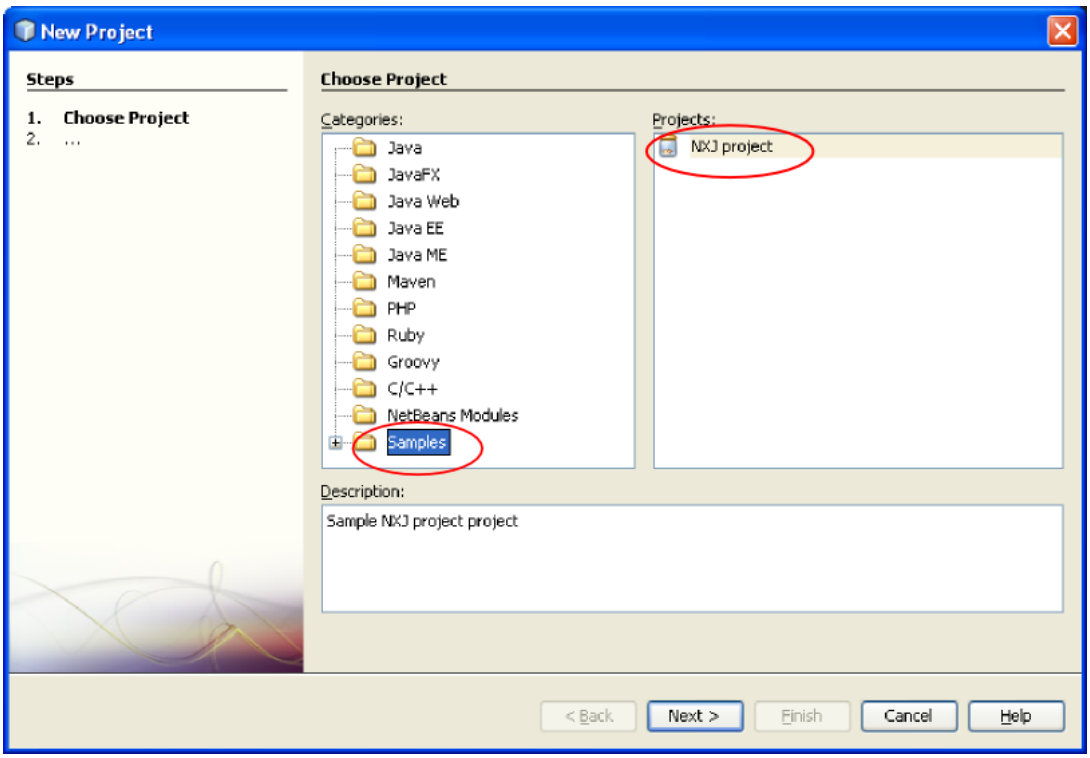
Nejdříve je potřeba do NetBeans nahrát NXJ plugin. Ten najdete v místě, kam jste nainstalovali *leJOSNXJProjects*. Obvykle bývá ve složce *Documents and settings*. Plugin poté najdete v *leJOSNXJProjects\NXJPlugin\build\nxjplugin.nbm*.

Do NetBeans tento plugin nainstalujete následujícím způsobem. Klikněte na *Tools* → *Plugins* → *Downloaded* → *Add Plugins* a vyberte *nxjplugin.nbm*



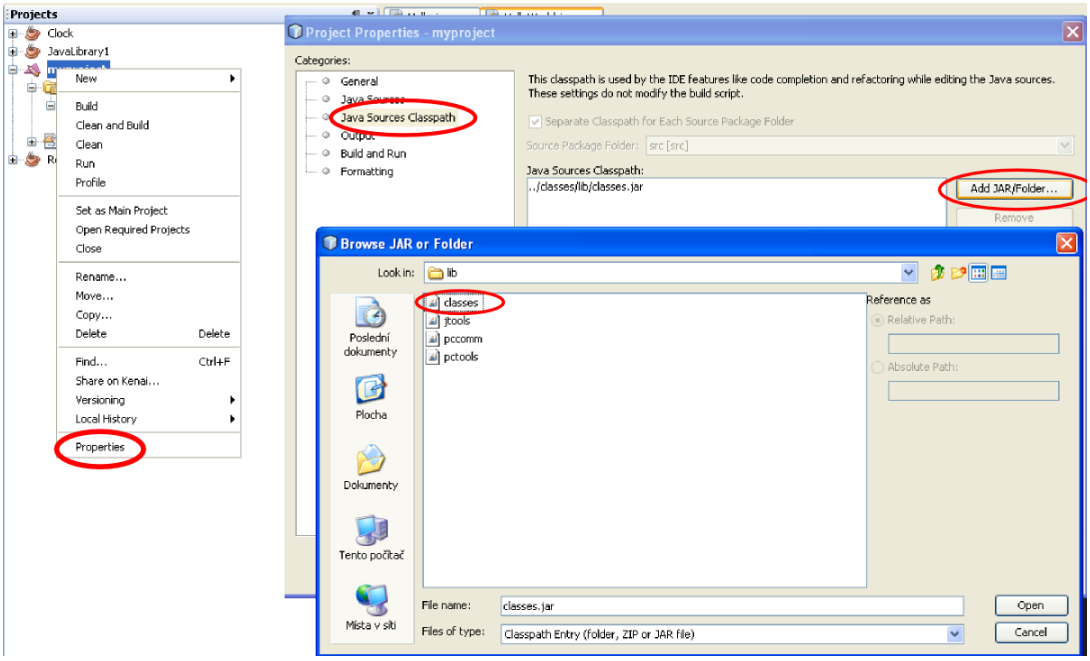
Obrázek 2.4: Ukázka přidání pluginu NXJ do NetBeans

Po přidání pluginu do NetBeans, je možné vytvářet NXJ projekt následujícím způsobem. Klikněte na *File* → *New Project* → *Samples* → *NXJ project* → *Next* a určete název a místo uložení projektu.



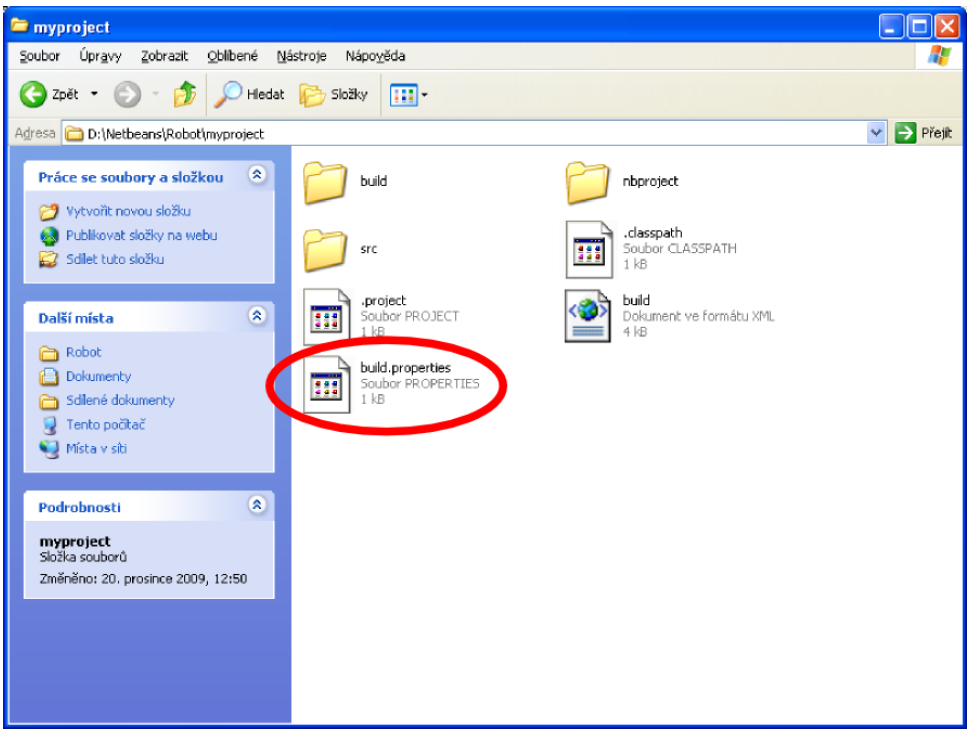
Obrázek 2.5: Ukázka vytvoření NXJ projektu v NetBeans

Nyní je potřeba do tohoto projektu zadat cestu k LeJOS balíčkům, abychom mohli pracovat s LeJOS třídami a metodami. Nejdříve přidáme balíček LeJOS do právě vytvořeného projektu v NetBeans. Klikněte pravým tlačítkem na název projektu *Properties* → *Java Sources Classpath* → *Add JAR/Folder* → *classes.jar*. Soubor *classes.jar* najdete v adresáři kam jste nainstalovali *LeJOS NXJ*, tedy obvykle v *C:\Program Files\leJOS NXJ\lib\classes.jar*



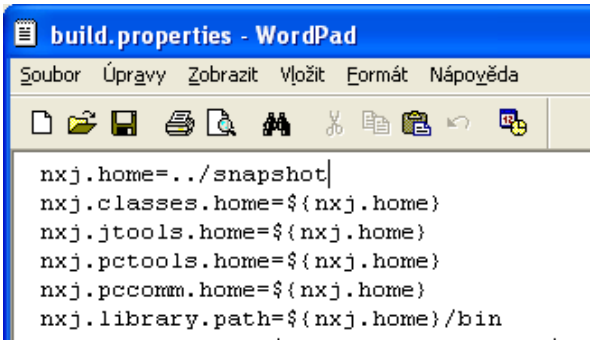
Obrázek 2.6: Přidání balíku *classes.jar* do projektu v NetBeans

Tím jsme nastavili cestu pro projekt v NetBeans. Další věcí, kterou je potřeba nastavit je cesta k LeJOS balíčkům pro *build.xml*, pomocí kterého nahráváme program do NXT kostky. Abychom ji nastavili, tak musíme otevřít adresář, kam jsme uložili náš NetBeans projekt. Měl by se zobrazit následující adresář.

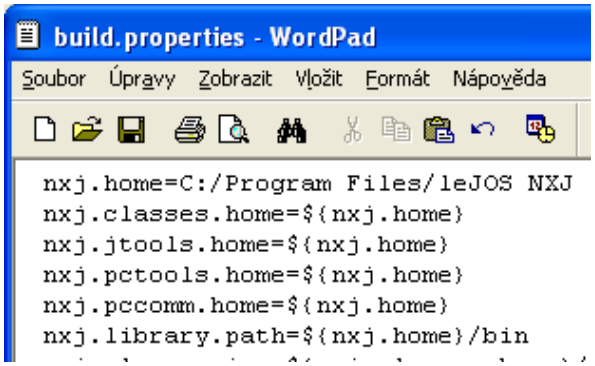


Obrázek 2.7: Adresář NetBeans projektu

Otevřete si soubor *build.properties* v textovém editoru (stačí např. program *WordPad*). Na prvním řádku je napsáno *nxj.home=../snapshot*. Text za rovnítkem změňte na cestu k adresáři LeJOS NXJ. První řádek bude vypadat např. takto *nxj.home=C:/Program Files/leJOS NXJ* (ujistěte se, že jste správně zadali lomítka a řádek nekončí mezerou).

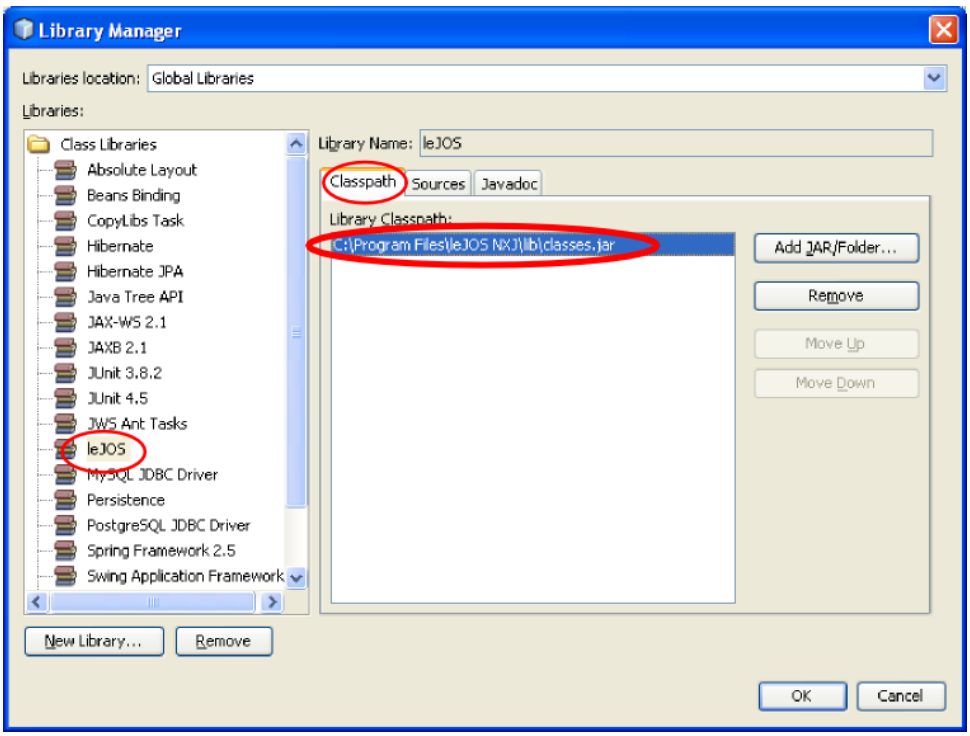


Obrázek 2.8: Build.properties před změnou



Obrázek 2.9: Build.properties po změně

Jako poslední věc je potřeba do NetBeans přidat Javadoc (dokumentaci), aby se kód psal lépe. To se udělá následujícím způsobem. Klikněte na *Tools* → *Libraries* → *New Library* a pojmenujte ji *LeJOS*. Do sekce *classpath* přidejte stejnou třídu jako jste přidali do projektu na obrázku 2.6. Do sekce *Javadoc* přidejte adresář *...\lejosNXJProjects\classes\doc*



Obrázek 2.10: Přidaná knihovna

Nyní je vše nastaveno. Jednotlivé akce (kompilace, linkování, upload) provádíme pomocí skriptu *build.xml*. Na příslušnou záložku klikneme pravím tlačítkem a vybereme *Run Target*. Nebo můžeme spustit projekt v NetBeans (F6), tím se nahraje program do NXT a poté se spustí.

2.5 Základní příkazy

2.5.1 Několik základních příkazů

Třída	Jméno metody	Poznámka
Button	void waitForPress()	Čeká se dokud se nezmáčkne tlačítko.
LCD	void drawString(String str, int x, int y)	Vykreslí řetězec na displej na určenou pozici.
Motor	void backward()	Spustí motor, který se bude točit dozadu.
	void changeDirection()	Obratí směr chodu motoru.
	void forward()	Spustí motor, který se bude točit dopředu.
	void setSpeed(int speed)	Nastaví rychlost pohybu (úhlových stupňů za vteřinu) motoru. Maximum je přibližně 900 při plné baterii.
	void stop()	Vypne motor.
	void rotate(float angle)	Otočí motor o úhel (ve stupních).

Tabulka 2.3: Základní příkazy

Ukázka použití příkazů

```
Motor.A.setSpeed(200);
```

Tímto příkazem vybereme třídu *Motor*. Specifikujeme, že chceme používat motor na pozici *A* a vybereme metodu *setSpeed* se vstupním parametrem *200*.
Výsledkem bude, že motor A se začne točit rychlostí 200 úhlových stupňů za vteřinu.

2.5.2 Příklad

Robot pojede dopředu, dokud se nestiskne tlačítko. Poté zrychlí a po druhém stisknutí tlačítka začne jezdit dokola. Po třetím stisknutí tlačítka se vypne.

```
package org.lejos.example;
import lejos.nxt.*;
public class Example {
```

```
        public static void main(String[] args) {
                // Spusti se motory A a C, vypise se text na LCD a ceka se
                // dokud se nezmackne tlacitko
                Motor.A.setSpeed(200);
                Motor.C.setSpeed(200);
                Motor.A.forward();
                Motor.C.forward();
                LCD.drawString("DOPREDU", 0, 0);
                Button.waitForPress();

                //Zrychli se
                Motor.A.setSpeed(500);
                Motor.C.setSpeed(500);
                LCD.drawString("RYCHLEJI", 0, 1);
                Button.waitForPress();

                //Zacne jezdit v kruhu
                Motor.A.setSpeed(500);
                Motor.C.setSpeed(200);
                LCD.drawString("DOKOLA", 0, 2);
                Button.waitForPress();

                //Zastavi se motory
                Motor.A.stop();
                Motor.C.stop();

        }
}
```

2.6 Pokročilejší příkazy

2.6.1 Několik pokročilejších příkazů

Jedna z výhod používání programovacího jazyku JAVA je možnost použít předdefinovaných tříd a metod. Jedním z velmi užitečných prostředků je interface *Pilot*.

Interface *Pilot* se používá pro robota typu vozidlo. Tedy takového robota, který má dva nezávislé motory, které pohánějí kola. Ukážeme si použití třídy *TachoPilot*, která interface *Pilot* implementuje. Do konstruktoru této třídy zadáme průměr a rozkol poháněných kol a pozice, na kterých jsou motory připojeny. Poté již můžeme využívat pokročilejších funkcí.

Konstruktory třídy *Pilot*:

TachoPilot(float wheelDiameter, float trackWidth,Motor leftMotor, Motor rightMotor)

TachoPilot(float wheelDiameter, float trackWidth,Motor leftMotor, Motor rightMotor, boolean reverse)

Hodnoty parametrů můžeme zadávat v libovolných jednotkách. Důležité je, abychom ve všech metodách používali stejné jednotky, jaké jsme zadali do konstruktoru.

Rovněž velmi užitečná je třída *SimpleNavigator*. Tato třída poskytuje hrubý odhad polohy. Funguje tak, že na počátku programu umístí robota do polohy {0,0}. Postupně jak se vykonávají příkazy, tak se přepočítává a zaznamenává aktuální poloha. Takto udržíme přehled o tom, kde se robot pohybuje.

Konstruktor třídy *SimpleNavigator*:
SimpleNavigator(Pilot pilot)
SimpleNavigator(float wheelDiameter, float trackWidth,Motor leftMotor, Motor rightMotor)
SimpleNavigator(float wheelDiameter, float trackWidth,Motor leftMotor, Motor rightMotor, boolean reverse)

Příklad použití: Můj robot-vozítko má průměr kola (napsáno na pneumatice) 81.6mm, rozkol kol je přibližně 110mm a motory mám zapojeny na výstupy *A* a *C*. Konstruktory pro tuto situaci bude vypadat takto:
Pilot pilot = new TachoPilot(81.6f, 110f, Motor.A, Motor.C);
SimpleNavigator navigator = new SimpleNavigator(pilot);

Metody třídy ***Pilot***:

Jméno metody	**Poznámka**
void arc(float radius); void arc(float radius, float angle)	Robot pojede po oblouku. Hodnota radius určuje poloměr zatáčky. Pokud je $radius > 0$, tak se robot začne otáčet doleva, pokud je $radius < 0$, tak doprava. angle je hodnota úhlu ve stupních o kterou se robot otočí
void backward()	Robot se začne pohybovat zpět.
void forward()	Robot se začne pohybovat dopředu.
float getAngle()	Metoda vrátí hodnotu úhlu ve stupních, kterou robot urazil od začátku běhu.
void setMoveSpeed(float speed)	Nastaví rychlost pohybu (hodnota proměnné speed za vteřinu).
void steer(float turnRate); void steer(float turnRate, float angle)	Robot pojede po zakřivené dráze. Hodnota turnRate může být -200 až 200. turnRate = 100 ... levé kolo stojí, pravé se točí dopředu→ ostrá levé zatáčka turnRate = 200 ... levé kolo se točí dozadu, pravé se točí dopředu → otočení doleva na místě Pokud před hodnotu dopíšeme minus, tak se robot bude otáčet doprava. angle je hodnota úhlu ve stupních o kterou se robot otočí
void travel(float distance)	Robot pojede dopředu (pokud je hodnota distance záporná, tak dozadu), dokud neurazí hodnotu proměnné distance.
void travelArc(float radius, float distance)	Robot pojede po oblouku. Hodnota radius určuje poloměr zatáčky. Pokud je $radius > 0$, tak se robot začne otáčet doleva, pokud je $radius < 0$, tak doprava. hodnota distance je dráha, kterou robot urazí.
float getTravelDistance()	Metoda vrátí vzdálenost, kterou robot urazil od začátku běhu.

Tabulka 2.4: Přehled nejdůležitějších metod třídy *Pilot*

Metody třídy ***SimpleNavigator***:

Jméno metody	**Poznámka**
float angleTo(float x, float y)	Metoda vrátí hodnotu úhlu ve stupních mezi aktuální pozicí a zadaným bodem $\{x,y\}$
float distanceTo(float x, float y)	Metoda vrátí vzdálenost k zadanému bodu $\{x,y\}$
float getAngle(float x, float y)	Metoda vrátí hodnotu úhlu ve stupních mezi směrem, kam robot směřuje, a směrem, kam robot směřoval na začátku.
float getX()	Metoda vrátí souřadnici x aktuální pozice.
float getY()	Metoda vrátí souřadnici y aktuální pozice.
void goTo(float x, float y)	Metoda zařídí, aby robot dojel na určenou pozici $\{x,y\}$
void setPose(float x, float y, float heading)	Nastaví nové hodnoty jako výchozí bod.

Tabulka 2.5: Přehled nejdůležitějších příkazů třídy *SimpleNavigator*

2.6.2 Příklady

Robot pojede po přímé dráze a udělá zatáčku o 180°. Takto bude jezdit, dokud se nestiskne tlačítko *Escape*. Robot tedy bude jezdit po dráze, která má tvar stadionu.

```
package org.lejos.example;

import lejos.nxt.*;
import lejos.robotics.navigation.Pilot;
import lejos.robotics.navigation.TachoPilot;

public class Example2 {

    Pilot pilot = new TachoPilot(81.6f, 120f, Motor.A, Motor.C);
    boolean stop = false;

    public void go() {

        /* Vytvorim listenera na tlacitko CANCEL -> pote
         * co ho stisknu, tak se hodnota stop zmeni na true
         */
        ButtonListener listen = new ButtonListener() {

            public void buttonPressed(Button b) {
                stop = true;
            }

            /* ButtonListener je interface, proto musim napsat
             * vsechny jeho tridy, i ty ktere nepouziji
             */
            public void buttonReleased(Button b) {
            }
```

```
        };

        /* listenera inicializuji
         */
        Button.ESCAPE.addButtonListener(listen);

        /*Jedu, dokud se nezmeni hodnota stop,
         * tu prubezne kontroluji.
         */
        while (stop == false) {
            if (stop == false) {
                pilot.travel(250);
            }

            if (stop == false) {
                pilot.steer(100, 180);
            }
        }
    }

    public static void main(String[] args) {

        //Spusti se metoda go()
        Example2 exam = new Example2();
        exam.go();

    }
}
```

2.7 Příkazy ovládající senzory

Nyní uvedeme příkazy, kterými budeme obsluhovat jednotlivé senzory.

2.7.1 Dotykový senzor

Konstruktor

TouchSensor(ADSensorPort port)

Metoda

Pro dotykový senzor máme k dispozici metodu
boolean isPressed()
Metoda zkontroluje, jestli senzor sepnul (nastal dotyk). Pokud ano, tak navrací *true*, jinak *false*.

Příklad

Dokud nesepne dotykový senzor, tak motory na pozicích A a C budou v chodu.

```
package org.lejos.example;

import lejos.nxt.*;

public class Example {

    public static void main(String[] args) {

        //Senzor je na vstupu cislo 1
        TouchSensor sensor = new TouchSensor(SensorPort.S1);

        while(sensor.isPressed() == false){
            Motor.A.forward();
            Motor.C.forward();
        }
    }
}
```

2.7.2 Světelný senzor

Konstruktory

LightSensor(ADSensorPort port)

LightSensor(ADSensorPort port, boolean floodlight)

Hodnotou proměnné *floodlight* říkáme, zda chceme použít červenou LED diodu pro osvícení povrchu. Pokud použijeme konstruktor bez této proměnné (ten první), tak se dioda rozsvítí automaticky. Tedy

LightSensor(SensorPort.S1) je ekvivalentní *LightSensor(SensorPort.S1,true)*

Konstruktorem *LightSensor(SensorPort.S1,false)* zajistíme, že se dioda nerozsvítí.

Metody

int readNormalizedValue()

Metoda vrátí normalizovanou hodnotu jasu v rozsahu od 0 (úplná tma) do 1023 (intenzivní světlo).

int readValue()

Metoda vrátí hodnotu jasu v procentech podle nastavené kalibrace.

void setFloodlight(boolean floodlight)

Touto metodou můžeme rozsvítit nebo zhasnout červenou LED diodu.

void setHigh(int high)
Tato metoda provede kalibraci horní hranice senzoru. Metoda *readValue* poté bude pro hodnotu *high* vracet hodnotu 100%.

void setLow(int low)
Tato metoda provede kalibraci spodní hranice senzoru. Metoda *readValue* poté bude pro hodnotu *low* vracet hodnotu 0%.

Příklad

Robot reagující na světlo. Pokud robota osvítíme světlem, tak pojede do té doby, než světlo zhasne. Jeho rychlost bude úměrná intenzitě světla.

```
package org.lejos.example;

import lejos.nxt.*;

public class Example {

    public static void main(String[] args) {

        LightSensor sensor = new LightSensor(SensorPort.S1,false);

        // Program se ukonci pote co se stiskne tlacitkou ESCAPE.
        while(Button.ESCAPE.isPressed() == false){

            int light = sensor.readNormalizedValue();

            /* Pokud je uroven jasu vice jak 180, tak se motory spusti a
             * jejich rychlost se nastavi na zmerenou hodnotu jasu.
             * Hodnota 180 byla experimentalne nastavena.
             */

            if(light >= 180)
            {
                Motor.A.setSpeed(light);
                Motor.C.setSpeed(light);
                Motor.A.forward();
                Motor.C.forward();
            }

            // Pokud ne, tak se motory zastavi
            else{
             Motor.A.stop();
             Motor.C.stop();
            }
        }
    }
}
```

2.7.3 Ultrazvukový senzor

Konstruktor

UltrasonicSensor(I2CPort port)

Metody

U ultrazvukového senzoru je nejdůležitější metoda *float getRange()*
Metoda vrátí vzdálenost v centimetrech k nejbližšímu objektu. Funguje spolehlivě v rozsahu přibližně 5cm až 190cm.

Příklad

Dokud nestisknu tlačítko *Escape*, bude se na displej vypisovat vzdálenost k nejbližšímu objektu.

```
package org.lejos.example;
import lejos.nxt.*;
public class Example {
    public static void main(String[] args) {
        //Senzor je na vstupu cislo 1
        UltrasonicSensor sensor = new UltrasonicSensor(SensorPort.S1);
        // Program se ukonci pote co se stiskne tlacitkou ESCAPE.
        while(Button.ESCAPE.isPressed() == false){
        LCD.drawString(""+sensor.getRange(), 0, 0);
        }
    }
}
```

2.7.4 Zvukový senzor

Konstruktory

SoundSensor(ADSensorPort port)
SoundSensor(ADSensorPort port, boolean dba)
Proměnnou *dba* nastavujeme režim senzoru. Pokud je *dba true*, tak bude senzor nastaven do režimu dBA, jinak bude v režimu dB.

Metody

int readValue()

Metoda vrátí hodnotu naměřenou senzorem v dB.

void setDBA(boolean dba)

Nastavuje režim senzoru stejným způsobem, jaký je vysvětlen u konstruktoru.

Příklad

Dokud nestisknu tlačítko *Escape*, bude se na displej vypisovat velikost hluku.

```
package org.lejos.example;

import lejos.nxt.*;

public class Example {

    public static void main(String[] args) {

        //Senzor je na vstupu cislo 1, rezim dBA
        SoundSensor sensor = new SoundSensor(SensorPort.S1, true);

        // Program se ukonci pote co se stiskne tlacitkou ESCAPE.
        while(Button.ESCAPE.isPressed() == false){
        LCD.drawString(""+sensor.readValue(), 0, 0);
        }
    }
}
```

Kapitola 3

Soutěžní úlohy

Jedním z úkolů bakalářské práce bylo provést návrh jedné nebo dvou nových soutěžních úloh s řízením ve třech různých programovacích prostředích (NXT-G, NXC a leJOS-NXJ). Obě dvě navržené úlohy jsou však natolik komplikované, že je v grafickém prostředí NXT-G nelze naprogramovat, ale je potřeba využít vyššího programovacího jazyka, jako je NXC nebo LeJOS-NXJ, ke zdárné implementaci. Proto řešení úloh v programovacím prostředí NXT-G není v práci uvedeno.

3.1 Úloha: Sledování čáry s křížením

Návrh této úlohy byl připraven jako nová úloha pro předmět ROBOTI. V této úloze se využije především práce se světelným senzorem a pohybovými součástmi robota.

3.1.1 Pravidla

Cíl úlohy

Z poskytnutých LEGO dílů postavit a naprogramovat robota tak, aby samostatně a co nejrychleji ujel dvě kola vyznačenou dráhou, aniž by se střetl s druhým závodícím robotem.

Technické prostředky

Týmu je na začátku soutěže zapůjčena základní souprava LEGO® MINDSTORMS® Education (9797). Dále souprava doplňkových technických dílů (9648) a síťový adaptér (9833), které se po skončení soutěže vrátí v kompletním stavu organizátorovi soutěže na

Katedře řídící techniky. Robota lze sestrojit libovolným způsobem, ale pouze ze součástek zapůjčených dílů. Vypůjčování si součástek od jiných týmů je zakázáno. Při vlastní soutěži je možné použít pouze akumulátor nebo baterie, nikoliv síťový adaptér.

NXT kostku lze naprogramovat libovolným způsobem, ale při navrácení musí kostka obsahovat standardní firmware LEGO MINDSTORMS.

3.1.2 Plán soutěže

Podrobnosti k pravidlům průjezdu dráhy

Robota lze do startovní pozice položit ručně. Dále již musí pokračovat sám, bez jakékoliv vnější pomoci.

Robot musí ujet dvě kola na dráze, a to jedno kolo vnitřní dráhou a jedno kolo vnější dráhou. Cílová pozice robota je tedy stejná, jako jeho počáteční pozice.

Robot při průjezdu nesmí žádnou svojí částí přesáhnout hranici své dráhy, která je vyznačena žlutou čárou.

Prostor křížení je vyznačen žlutou čarou. Robot, který jako první překročí tuto čáru má na křižovatce přednost. V případě kolize bude posouzena míra zavinění a v dané rozjížďce diskvalifikován ten robot, který porušil pravidla. Pokud je viník kolize nejasný, může se zopakovat rozjížďka s prohozenými startovními pozicemi. Pokud by mělo dojít k opakování rozjížďky u stejných soupeřů potřetí, pak bude závod ukončen jako nerozhodný.

Porušení jakéhokoliv pravidla vede k diskvalifikaci robota v příslušné rozjížďce. Druhý robot však musí jízdu dokončit. Diskvalifikace obou robotů je chápána jako nerozhodný výsledek.

Soutěžní dráha

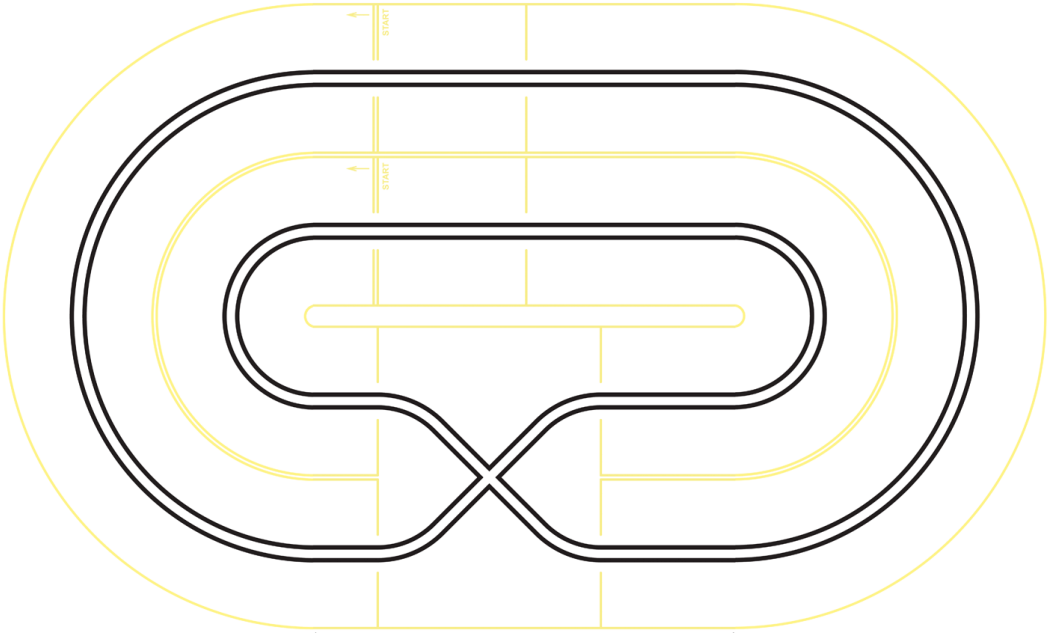
Vodící čáru jedné dráhy představují dvě souběžně vedené černé čáry, jejichž tloušťka je přibližně 10 mm. Vzdálenost mezi těmito čarami je přibližně 20 mm.

Rovný úsek dráhy je dlouhý 1000 mm. Poloměr vnějšího oblouku je 560 mm (délka 1759 mm) a poloměr vnitřního oblouku je 200 mm (délka 628 mm).

Na rovném úseku je žlutou čárou vyznačen startovní prostor. Startovní prostor je čtverec o straně 350 mm, jehož středem prochází vodící čára. Robot musí být při startu celý v tomto startovním prostoru (hranice startovního prostoru se promítá svisle vzhůru). Za start je považována osa vnitřní startovní čáry (zdvojená čára). Start je vyznačen 155 mm od první zatáčky.

Prostor křížení začíná 155 mm za první zatáčkou. Poloměr zatáčky realizované v křížení je 200 mm (délka dráhy 157 mm). Délka rovného úseku je 343 mm. Křižovatka tak zabírá 526 mm rovného úseku a při jejím projetí se urazí vzdálenost 657 mm. Zbývající rovné úseky v součtu tvoří dráhu dlouhou 474 mm.

Hranice dráhy, ve které se robot smí pohybovat je široká 350 mm. Celková dráha, kterou robot musí ujet, tj. obě dvě kola, má délku 9036 mm.



Obrázek 3.1: Soutěží dráha pro úlohu Sledování čáry s křížením

3.1.3 Hardwarové řešení úlohy

V této úloze bude nejdůležitější součástkou světelný senzor, ten je výhodné umístit co nejníže nad povrch dráhy, aby fototranzistor senzoru byl zaměřen na malou plochu, čímž se měření stane přesnější. Musíme ale dát pozor na nerovnosti trati, protože čím je senzor umístěn níže, tím více se stává náchylnější na rušení v podobě nerovností na trati.

Pro konstrukci robota využijeme základní návod, který je k dispozici na webové stránce [6], který je součástí základní soupravy LEGO® MINDSTORMS® Education (9797). Použijeme modifikaci podle návodu na webové stránce [7], ve kterém je světelný senzor umístěn na čele robota. Další senzor, který bude zapotřebí, je ultrazvukový senzor. Ten namontujeme nad světelný senzor tak, aby směřoval před robota.

Obrázek 3.2: Konstrukce robota pro Sledování čáry

3.1.4 Řešení v programovacím prostředí NXC

Jak ještě bude zmíněno, prostředí NXC má delší periodu vykonávání jednotlivých metod, proto není možné implementovat koncept "Zig-Zag"(bude vysvětleno dále). Z tohoto důvodu zde použijeme jiný koncept.

Koncept Sledování čáry

Koncept "Sledování čáry"je velmi populární soutěžní úlohou, na které můžeme ukázat základní koncept řízení. Robot pojede na rozhraní černé a bílé čáry a světelným senzorem bude snímat intenzitu světla, která se odráží od dráhy, na kterou svítí červenou LED diodou. Naměřená intenzita světla leží v jistém rozsahu hodnot. Je výhodné nejdříve změřit maximální a minimální možné velikosti intenzity, abychom podle těchto hodnot mohli kalibrovat senzor. Po kalibraci bude senzor vracet relativní hodnotu intenzity světla od 0% (černá čára) do 100% (bílá čára). Náš program se tím stane přehlednější.

Je třeba si uvědomit, že robot nedokáže jet pouze po černé nebo bílé čáře, ale pouze po jejich rozhraní. To vede k rozhodnutí, jakou čáru si zvolit a ze které strany ji sledovat.

Zvolíme čáru, která je napravo z pohledu robota a tuto čáru bude robot sledovat z levé strany. Později ukážeme, že tato volba je nejvýhodnější. Předpokládejme, že jsme úspěšně kalibrovali senzor. Budeme požadovat, aby senzor měl na výstupu hodnotu 70%. Pojede tedy blíže bílé čáře. K dosažení tohoto požadavku musíme sestavit regulátor.

Začněme s nejjednodušším možným regulátorem. Senzorem změříme relativní intenzitu světla a odchylku od námi požadované hodnoty vynásobíme konstantou. Tím získáme akční zásah nebo-li vstup do soustavy. Toto můžeme zapsat do rovnice

$$u(t) = k_p \cdot e(t) \tag{3.1}$$

Pro náš diskrétní případ bude rovnice vypadat

$$u = k_p \cdot (y - r) \tag{3.2}$$

kde *e(t)* je odchylka od požadované hodnoty, u je akční zásah, y je výstup senzoru, r je požadovaná hodnota a k_p je proporcionální konstanta, kterou v tomto případě nalezneme metodou cyklické optimalizace. Takovýto regulátor nazýváme *P-regulátor* a regulaci *řízení odchylkou.*

V našem případě je tento regulátor nedostačující. Perioda vykonávání metod je v prostředí *NXC* delší než v prostředí *LeJOS*. Díky tomu robot příliš překmitává při průjezdu zatáček. Zmíněný jev odstraní použití derivační složky. Tato složka předpovídá vývoj stavu v nejbližší budoucnosti podle údajů z minulosti. Akční zásah poté bude mít tvar

$$u(t) = k_p \cdot e(t) + k_d \cdot \dot{e}(t) \tag{3.3}$$

Pro náš diskrétní případ bude rovnice vypadat

$$u = k_p \cdot (y - r) + k_d \cdot ((y - r) - (y_{-1} - r)) \tag{3.4}$$

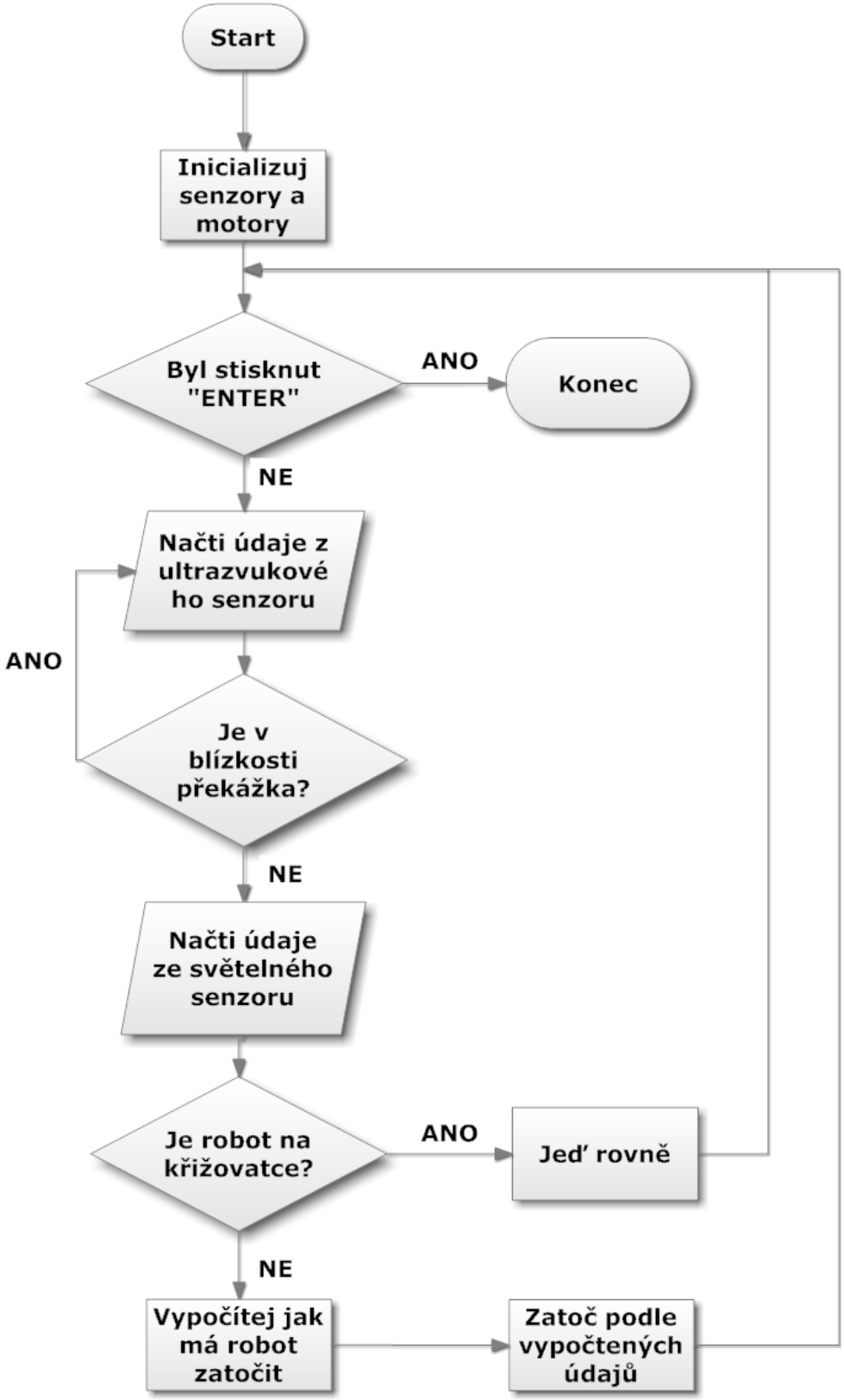
kde *e(t)* je odchylka od požadované hodnoty, $e(\dot{t})$ je derivace odchylky od požadované hodnoty, u je akční zásah, y je výstup senzoru, y_{-1} je výstup senzoru v předchozím kroku, r je požadovaná hodnota k_p je proporcionální konstanta, k_d je derivační konstanta, kterou opět nalezneme experimentálně. Tento regulátor se nazývá *PD-regulátor.*

S tímto regulátorem dokážeme sledovat čáru a zároveň eliminovat překmity. Zbývá vyřešit průjezd křižovatkou, kde robot na chvíli sjede z čáry. Toto vyřešíme zavedením podmínky vyžadující, že pokud robot sjede z černé čáry, tak pojede rovně dokud na ni

opět nenarazí. To je právě důvodem, aby robot jel nalevo od pravé čáry, jinak by mohla nastat situace, že by robot vyjel z trati.

Na křižovatce rovněž musíme zabránit srážce robotů. K tomu využijeme ultrazvukový senzor. Pokud senzor detekuje, že v bezprostřední blízkosti před ním se nachází nějaký objekt (v našem případě druhý robot), tak robot zastaví a vyčká, dokud druhý robot neprojede.

Vývojový diagram



Obrázek 3.3: Vývojový diagram pro koncept Sledování čáry

Důležité části zdrojového kódu

Uvedeme zde nejdůležitější části použitého kódu. Poznamenejme, že pokud se v uvedeném kódu vyskytnou ... (tři tečky), tak se jedná o část, která není důležitá pro pochopení kódu, proto byla vypuštěna z následujícího přehledu.

NXC nenabízí možnost kalibrovat senzor, proto je každá změřená hodnota přepočítána následující metodou

```
int ReadCalibratedSensor(){
    int readings = 0;
    sensorValue = Sensor(IN_1);
    readings = (sensorValue - LOW_VALUE) * 100;
    readings= readings/(HIGH_VALUE-LOW_VALUE);
    return readings;
}
```

Výše zmíněný PD regulátor realizuje funkce *CalculateSteerRate*

```
int CalculateSteerRate(){
    last_error = error;
    int calculation = 0;
    error = calibratedSensorValue - CALIBRATION;
    derivative = error - last_error;
    ...
    // PD regulátor
    calculation = (KP * error)/100 + (KD * derivative)/100;
    return calculation;
}
```

Průjezd křižovatkou

```
if(sensorValue<100){
...
   OnFwdSync(OUT_AC, SPEED,steerRate);
}
// Na křižovatce
else{
    OnFwdReg(OUT_AC, SPEED,1);
}
```

3.1.5 Řešení v programovacím prostředí LeJOS NXJ

Prostředí LeJOS NXJ nabízí mnoho výhod při programování robota. Jednou z nich je menší perioda vykonávání metod. Robot následně může jet rychleji a přesněji. Při programování konceptu "Sledování čáry"byl čas projetí dvou kol 2 minuty a 7 vteřin. Oproti prostředí NXC se průjezd zrychlil o 8 vteřin. Rovněž nebyla zapotřebí derivační složka. Dále se budeme věnovat pouze konceptu "Zig-Zag".

Koncept Zig-Zag

Koncept "Zig-Zag" nám dovoluje použít skutečnost, že dráhu tvoří dvojice černých čar. Jak už sám název konceptu napovídá, robot bude střídavě jezdit od jedné čáry ke druhé. Díky tomu budeme moci výrazně zvýšit jeho rychlost. Oproti výše zmíněné metodě je rychlost robota více jak dvojnásobná. Zásadní podmínkou, kterou musíme dodržet, aby měl tento koncept šanci na úspěch, je střídání černých čar. Pokud by nastala situace, že robot dvakrát za sebou najede na stejnou čáru, tak se ztratí a zpět na dráhu by ho vrátila pouze šťastná shoda okolností.

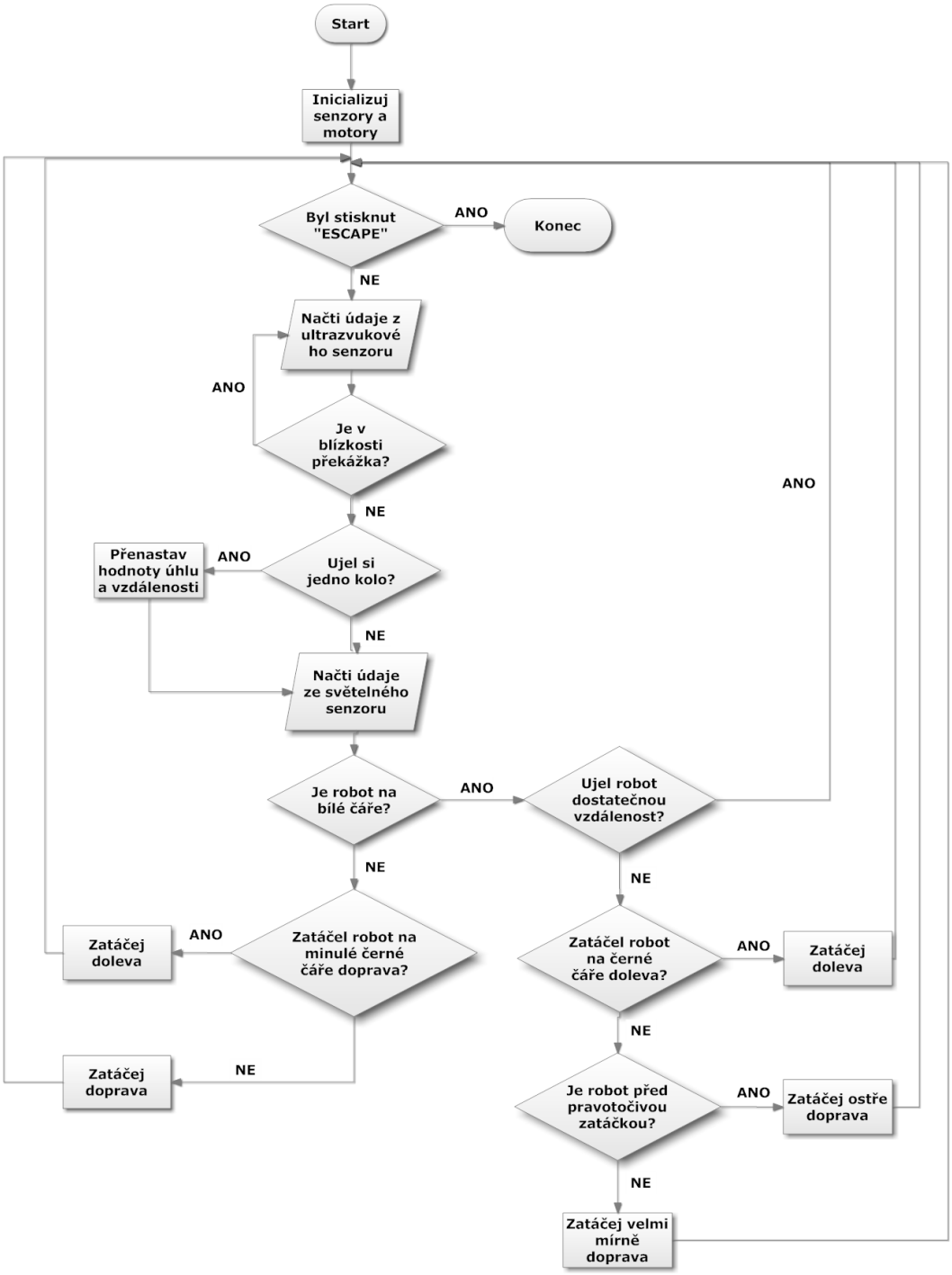
Robot tedy musí přejíždět od pravé čáry k levé čáře a zpět pod co největším úhlem, aby dokázal bezpečně projet i těmi nejostřejšími zatáčkami na trati. Pokud ovšem zvolíme tento úhel příliš velký, tak se robot bude posouvat dopředu velmi pomalu. Jedná se tedy o kompromis mezi bezpečným a rychlým projetím.

Tvar trati nám napoví, jak tento problém vyřešit. Všimneme si, že většina zatáček je levotočivých. Toho využijeme a robota naprogramujeme tak, aby od pravé čáry k levé čáře točil pod velkým úhlem a od levé čáry zpět k pravé se točil velmi pozvolna, až neznatelně. Tím se znatelně zrychlí rychlost robota a levotočivé zatáčky projede bezpečně. Bohužel tím vznikl problém s projetím pravotočivých zatáček, které jsou na trati právě dvě. Tuto situaci vyřešíme tak, že robot při přiblížení k pravotočivé zatáčce změní režim a začne přejíždět od levé čáry k pravé pod větším úhlem. Po projetí zatáčky se vrátí do původního režimu. Robot tedy musí vědět, kde se na trati přibližně nachází. S tímto úkolem nám pomohou *LeJOS* metody, které bychom v *NXC* nenašli. Jedná se o metody *getAngle()* pro zjištění celkového úhlu otočení a *getTravelDistance()* pro zjištění celkové uražené vzdálenosti. Pomocí těchto dvou metod můžeme polohu pravotočivých zatáček poměrně přesně stanovit. Nesmíme však zapomenout na skutečnost, že nevíme jestli bude robot začínat na vnitřní nebo vnější trati. Proto musíme počítat s periodou -360°, respektive 4400 mm.

Třetí zásadní metodou, kterou budeme používat je *steer(float turnRate)*. Tato metoda nám umožní hladce projíždět zatáčky tím, že se motory navzájem synchronizují v pohybu. Jak je uvedeno v tabulce 2.4, jako argument této metody můžeme zadat číslo od -200 do 200. Pokud zvolíme číslo, které je absolutně větší než 100, tak robot točí jedním motorem dopředu a druhým dozadu, čímž se otočí na menším poloměru. Bohužel zjistíme, že pokud jako argument zadáme takovéto číslo, tak se robot začne točit na místě a stane se neovladatelným. Jedná se patrně o chybu firmwaru (0.8.5beta) a v novějších verzích již může být tento problém vyřešen.

Konečně, musíme zabránit, aby se roboti střetli při průjezdu křižovatkou, což vyřešíme stejným způsobem jako v prostředí NXC. Ultrazvukovým senzorem budeme hlídat, aby před robotem nebyl žádný objekt.

Vývojový diagram



Obrázek 3.4: Vývojový diagram pro "Zig-Zag" koncept

Důležité části zdrojového kódu

Zda je robot na černé nebo bílé čáře se pozná podle podmínky:

```
sensor_value = sensor.readValue();
if (sensor_value > WHITE_THRESHOLD) {
   white_line();
}
else {
   black_line();
}
```

Metoda pro jízdu po bílé čáře

Pokud robot na černé čáře zatáčel doleva, tak v tom bude pokračovat. Pokud zatáčel doprava, tak se ověří, zda není před pravotočivou zatáčkou a podle toho se zachová.

```
private void white_line() {
...
   if (steerRate < 0) {
      pilot.steer(WHITE_LEFT_STEER);
   } else {
      // Jsem před pravotočivou zatáčkou?
      if((check_front_right_angle() && check_front_right_distance()) ||
      (check_back_right_angle() && check_back_right_distance())){
        pilot.setMoveSpeed(SPEED_SLOW);
    pilot.steer(WHITE_RIGHT_STEER);
      }
      else{
  pilot.setMoveSpeed(SPEED);
         pilot.forward();
      }
   }
}
```

Metoda pro jízdu po černé čáře

Nejdříve se podle údaje ze senzoru vypočte jak prudce zatočit, poté se zjistí, zda robot posledně zatáčel doprava a podle těchto údajů robot zatočí.

```
private void black_line() {
    ...
    steerRate = 100 - (KP * sensor_value);
    // Zatáčel jsem doprava?
    if (steeredRight) {
        steerRate = -steerRate;
    }
    ...
    pilot.steer(steerRate);
}
```

Kontrola, zda před robotem není překážka.

```
while(ultra_sensor.getRange() < SAFETY_DISTANCE){
     pilot.stop();
}
```

3.1.6 Výsledek

V obou prostředích se podařilo robota úspěšně naprogramovat tak, aby projel tratí. V prostředí *NXC* byl při konceptu "Sledování čáry"čas 2 minuty 15 vteřin. Stejným konceptem implementovaným v prostředí *LeJOS* dosáhneme času 2 minuty 7 vteřin. Jako velmi efektivní se ukázal koncept "Zig-Zag", pomocí kterého robot dokázal projet trať za čas 1 minuta 7 vteřin. Tedy o celou minutu rychleji, než při sledování čáry. Videa těchto průjezdů je možno shlédnout na internetových stránkách [8], [9].

Oba koncepty byly úspěšně předvedeny na výstavě AMPER dne 15.dubna 2010. Fotogalerii, která je věnována této výstavě lze najít na webové stránce [11]

3.2 Úloha: Skladiště

Návrh této úlohy byl připraven jako nová úloha pro předmět ROBOTI. V této úloze se využije téměř všechny součásti robota jak senzorické, tak i pohybové.

3.2.1 Pravidla

Cíl úlohy

Z poskytnutých LEGO dílů postavit a naprogramovat robota tak, aby samostatně co nejrychleji dokázal přemístit předměty na trati do vyznačených prostor.

Technické prostředky

Týmu je na začátku soutěže zapůjčena základní souprava LEGO® MINDSTORMS® Education (9797). Dále souprava doplňkových technických dílů (9648) a síťový adaptér (9833), které se po skončení soutěže vrátí v kompletním stavu organizátorovi soutěže na Katedře řídící techniky. Robota lze sestrojit libovolným způsobem, ale pouze ze součástek ze zapůjčených dílů. Vypůjčování si součástek od jiných týmů je zakázáno. Při vlastní soutěži je možné použít pouze akumulátor nebo baterie, nikoliv síťový adaptér.

NXT kostku lze naprogramovat libovolným způsobem, ale při odevzdání musí kostka obsahovat standardní firmware LEGO MINDSTORMS.

3.2.2 Plán soutěže

Podrobnosti k pravidlům

Robot může být umístěn kamkoliv do prostoru startovního pole, které je před startovní čárou. Po jeho spuštění se musí pohybovat zcela samostatně a nesmí být jakkoliv naváděn ze strany soutěžících.

Robot musí nejprve zavést první předmět do skladovacího prostoru (na obr. 3.5 vyznačen zelenou barvou), poté musí druhý předmět přemístit do cílového prostoru (na obr. 3.5 vyznačen fialovou barvou).

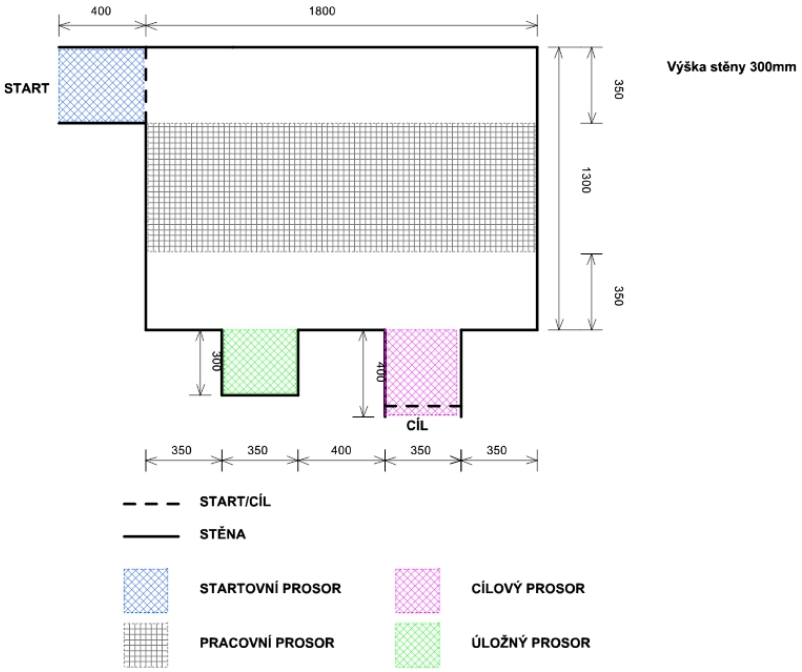
Jednotlivé předměty mohou být buď tenisový míček nebo plechovka od nápoje.

Měří se čas od doby, kdy robot poprvé protne startovní linii až do chvíle, kdy robot projede cílovou linií.

Soutěžní dráha

Do skladiště i do cílového prostoru vede černá čára, která nemá pravidelný tvar a začíná vždy u protější stěny, na kterou je kolmá. Tato čára je tvořena černou elektrikářskou lepicí páskou, širokou nejméně 1 cm.

Povrch soutěžního plánu je tvořen bílou laminátovou deskou a výška stěny je přibližně 35 cm.



Obrázek 3.5: Soutěží dráha pro úlohu Skladiště

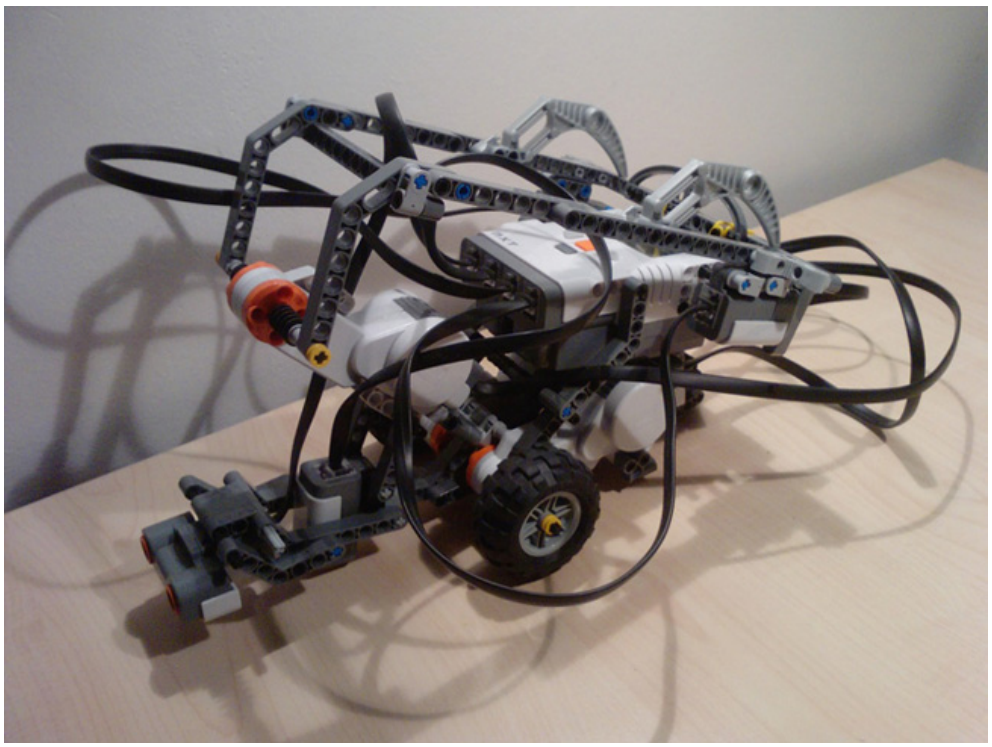
Obrázek 3.6: Fotografie soutěžní dráhy pro úlohu Skladiště

3.2.3 Hardwarové řešení úlohy

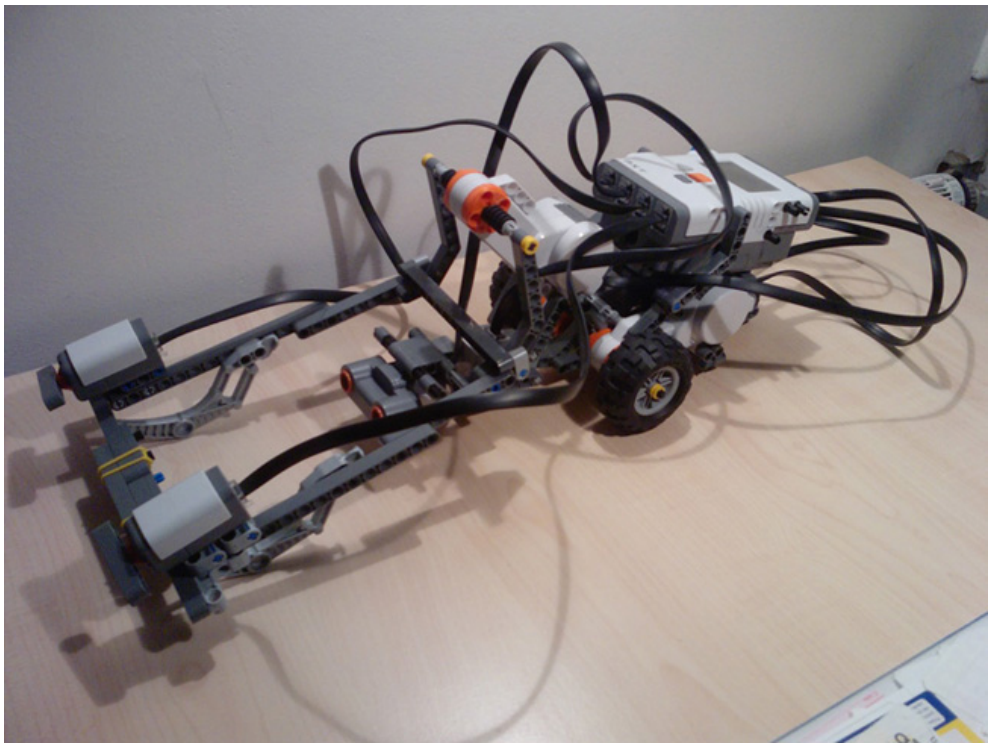
V této úloze bude konstrukce robota o něco složitější, než v předcházející úloze. Budeme využívat všechny 3 servomotory a všechny senzory s výjimkou zvukového. V rámci bakalářské práce byly použity dvě různá hardwarová řešení této úlohy.

První varianta

První řešení vychází z předchozí úlohy. Jeden z rozdílů je v umístění ultrazvukového senzoru, který byl přemístěn těsně nad povrch desky. Rovněž NXT kostka byla přemístěna více dozadu, aby se robot vyvážil a vzniklo místo pro třetí servomotor. Tento servomotor obsluhuje rameno, kterým se zajistí přepravovaný objekt tak, aby při jízdě nevypadl. Na přední část tohoto ramene byly namontovány dotykové senzory, které slouží k detekci stěny.



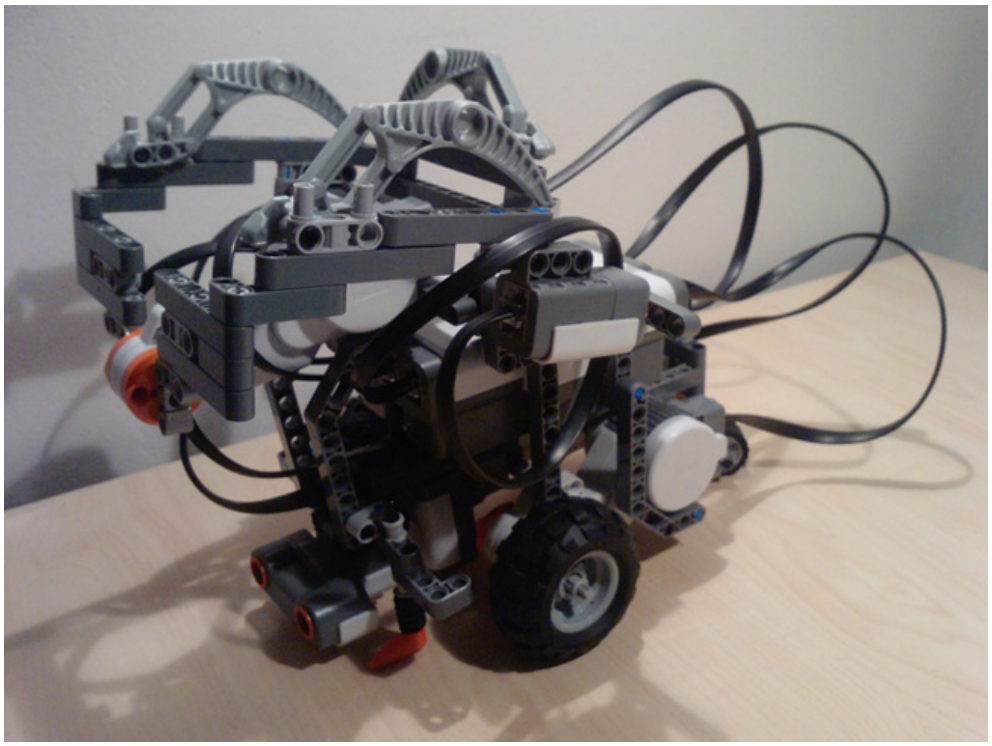
Obrázek 3.7: Konstrukce robota v úloze "Skladiště" var.1. Poloha = rameno nahoře



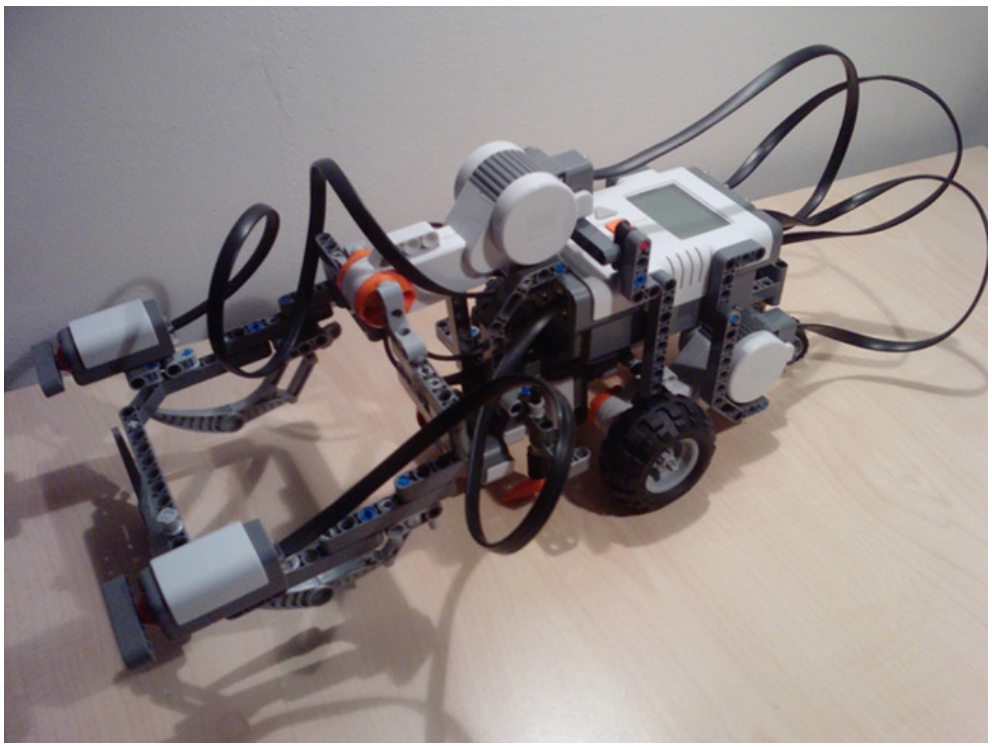
Obrázek 3.8: Konstrukce robota v úloze "Skladiště" var.1. Poloha = rameno dole

Druhé varianta

Druhé řešení používá jako základ konstrukci, která byla převzata z webové stránky [12]. Tuto konstrukci mírně upravíme použitím jiného zadního kolečka a přidáním senzorů. Světelný senzor umístíme pod NXT kostku mezi dva servomotory těsně nad povrch desky a ultrazvukový senzor umístíme před něj. Při umístění třetího servomotoru, který bude obsluhovat rameno se necháme inspirovat návodem z webové stránky [13]. Přední část ramene, stejně jako v předchozím případě, osadíme dotykovými senzory pro detekci stěny.



Obrázek 3.9: Konstrukce robota v úloze "Skladiště"var.2. Poloha = rameno nahoře



Obrázek 3.10: Konstrukce robota v úloze "Skladiště"var.2. Poloha = rameno dole

3.2.4 Programátorské řešení

Při řešení v obou programovacích prostředích budeme používat jednotný koncept, a to následující:

Řešení úlohy rozčleníme do několika menších částí:

a) Jízda k čáře

b) Sledování čáry

c) Odvoz objektu do skladovacích prostor

d) Jízda k protější stěně

Jízda k čáře

V této části úlohy musíme nalézt černou čáru, kterou posléze budeme sledovat. Budeme předpokládat, že čára leží před robotem a že mezi robotem a čárou není žádná překážka. Stačí, když robot pojede rovně a bude světelným senzorem snímat podklad, po kterém jede. Až robot přejede černou čáru (pojede tedy na levé straně černé čáry), tak se otočí o 90 stupňů doprava, protože objekt k odvezení bude umístěn v pravé části skladiště (z pohledu startu).

Sledování čáry

V této části sledujeme černou čáru až k objektu, který chceme přepravit. Můžeme použít řešení konceptu "Sledování čáry"z úlohy "Sledování čáry s křížením", které mírně upravíme. Vynecháme z něj část, která zajišťovala průjezd křižovatkou, a mírně pozměníme úsek kódu, který zabraňoval srážce s druhým robotem. Cizí objekt detekujeme pomocí ultrazvukového senzoru. Toto je jeden z méně spolehlivých článků řetězce řešení celé úlohy, protože tento senzor sice spolehlivě detekuje stěnu, ale menší překážku, jako například míč nebo plechovka, detekuje obtížně.

Odvoz objektu do skladovacích prostor

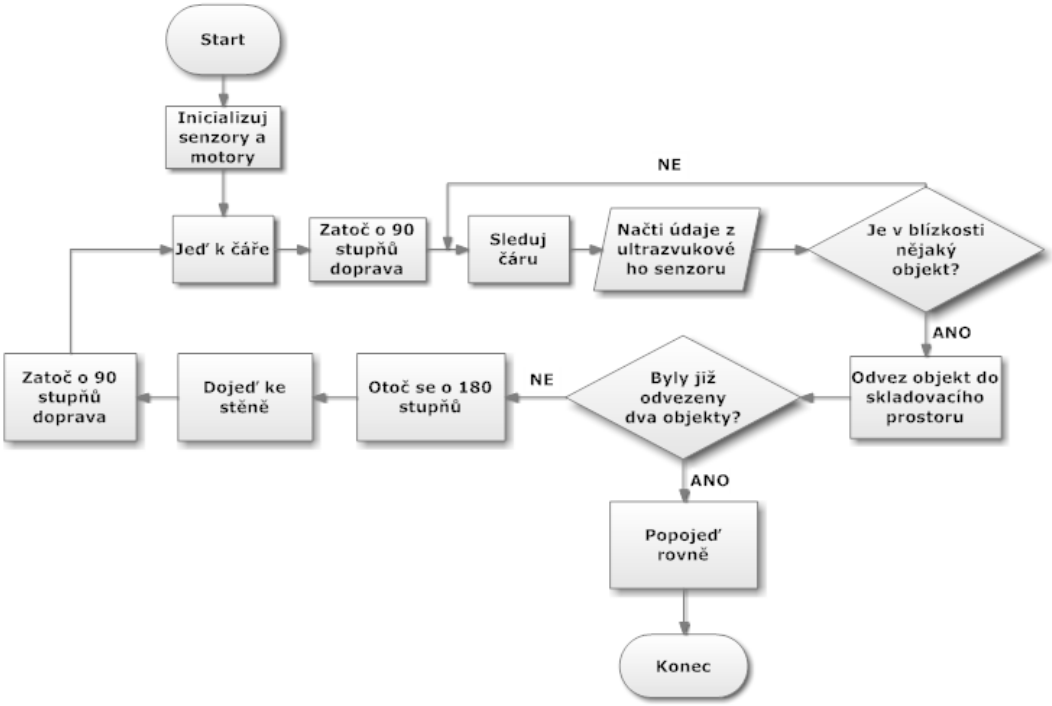
Nyní jsme se dostali do situace, že jsme se zastavili u nákladu, který je potřeba odvést. V této části objekt odvezeme do skladu, přičemž budeme předpokládat, že ke skladu vede černé čára namalovaná na podložce. Robot nejdříve skloní přepravní rameno, ve kterém zajistí náklad, a poté bude pokračovat ve sledování černé čáry. Sledování čáry se ukončí, až dotykové senzory umístěné na rameni detekují stěnu, čímž se zjistí, že robot dojel do skladu. Poté robot kousek poodjede dozadu, aby mohl bezpečně zvednout přepravní rameno. Následně postrčí objekt do skladu.

Po tomto kroku se robot rozhodne jestli má pokračovat a odvést další náklad. Pokud již odvezl oba náklady, tak pouze vyjede ze skladiště a zastaví. V opačném případě se otočí o 180 stupňů a pojede k protější stěně.

Jízda k protější stěně

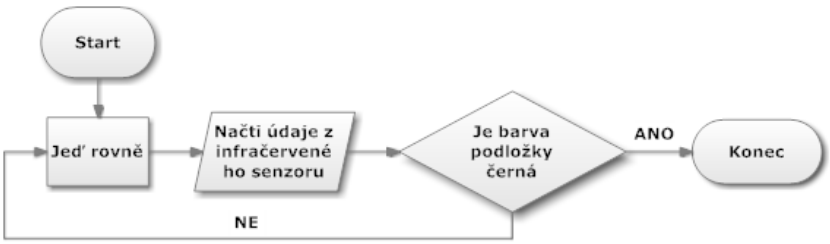
V této části robot pojede k protější stěně, u které se otočí. Jelikož v tomto okamžiku by robot měl mít zvednuté přepravní rameno, tak se může orientovat podle ultrazvukového senzoru. Pokud v blízkosti detekuje stěnu, tak se otočí o 90 stupňů doprava a začne znovu část "Jízda k čáře".

Vývojové diagramy



Obrázek 3.11: Vývojový diagram pro skladiště

Část pro jízdu k čáře

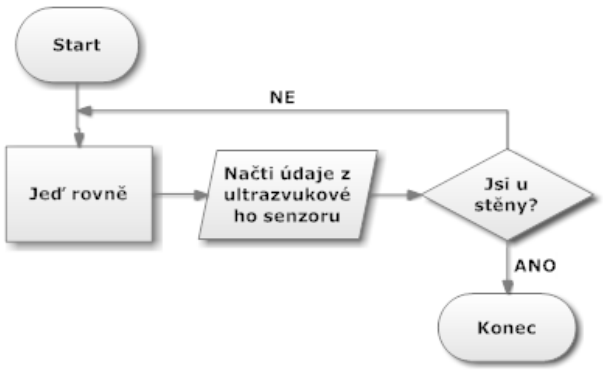


Obrázek 3.12: Vývojový diagram pro "Jeď k čáře"

Část pro odvoz objektu do skladovacího prostoru

Obrázek 3.13: Vývojový diagram pro "Odvez objekt do skladovacího prostoru"

Část pro jízdu ke stěně



Obrázek 3.14: Vývojový diagram pro "Jeď ke stěně"

3.2.5 Řešení v programovacím prostředí NXC

Metoda pro jízdu k černé čáře

```
void runToLine(){
        int i = 0;
        while(i<10){
            if(ReadCalibratedSensor() > BLACK_LINE_THRESHOLD){
                OnFwdReg(OUT_AC, SPEED,1);
            }
            else{
                i++;
            }
        }
    }
```

Metoda pro odvoz nákladu

```
void transportCargo(){
            ...
            armDown();
            while((checkBothTouchSensors() == false)){
                followLine();
            }
            RotateMotor(OUT_AC, SPEED, -100);
            armUp();
            RotateMotor(OUT_AC, SPEED, 80);
            RotateMotor(OUT_AC, SPEED, -220);
            cargo ++;
}
```

Metoda pro dojezd ke zdi

```
void runToWall(){
        while(SensorUS(IN_2)>DISTANCE_TO_WALL){
           OnFwdReg(OUT_AC, SPEED,1);
        }
}
```

Hlavní metoda

```
void go(){
       ...
       while(cargo<2){
           runToLine();
           // Otoc se o 90 stupnu doprava
           RotateMotorEx(OUT_AC, 45, 220, 100, true, true);
            ...
           // Sleduj caru dokud nenarazis na objekt
            while(close<2){
               followLine();
               if(checkCloseDistance()){
                   close++;
               }
           }
           close = 0;
            transportCargo();

            // Odvezl si uz dva objekty
            if(cargo>1){
                break;
            }
            ...
            // Otoc se o 180 stupnu
            RotateMotorEx(OUT_AC, 45, 447, 100, true, true);

            runToWall();

            // Otoc se o 90 stupnu doprava
            RotateMotorEx(OUT_AC, 45, 220, 100, true, true);
            ...
       }
    ...
}
```

3.2.6 Řešení v programovacím prostředí LeJOS NXJ

Metoda pro jízdu k černé čáře

Proměnná *i* užitá jako parametr funkce *while* slouží pro filtraci chyb senzoru.

```
private void runToLine(){
        int i = 0;
        while(i<12){
            if(lightSensor.readValue() > BLACK_LINE_THRESHOLD){
                pilot.steer(STRAIGHT_STEERING);
            }
            else{
                i++;
            }
        }
}
```

Metoda pro odvoz nákladu

```
private void transportCargo(){
            ...
            armDown();
            ...
            while((checkBothTouchSensors() == false)){
                followLine();
            }
            pilot.travel(-40);
            armUp();
            pilot.travel(30);
            pilot.travel(-100);
            cargo ++;
}
```

Metoda pro dojezd ke zdi

```
private void runToWall(){
        while(ultraSensor.getRange()>DISTANCE_TO_WALL){
            pilot.steer(STRAIGHT_STEERING);
        }
}
```

Hlavní metoda

Proměnná *close* užitá jako parametr funkce *while* slouží pro filtraci chyb senzoru.

```
public void go(){
...
       while(cargo<2){
           runToLine();
           pilot.rotate(90);
            ...
           // Sleduj caru dokud nenarazis na objekt
            while(checkCloseDistance()==false){
                followLine();
            }
            while(close<2){
               followLine();
               if(checkCloseDistance()){
```

```
                close++;
            }
        }
        close = 0;
         transportCargo();

         // Odvezl si uz dva objekty
         if(cargo>1){
             break;
         }
         pilot.rotate(-190);
         runToWall();
         pilot.rotate(90);
    }
    ...
}
```

3.2.7 Výsledek

Úspěšně se podařilo naprogramovat robota tak, aby přemístil oba předměty (v našem případě dva tenisové míčky) do vyznačených prostor. Záznam tohoto řešení lze shlédnout na internetové stránce [10].

Přestože byla úloha úspěšně vyřešena, je třeba zmínit komplikace, které vyvstaly při použití ultrazvukového senzoru pro detekci přepravovaných předmětů. Ultrazvukový senzor dokáže spolehlivě měřit vzdálenost k nejbližší stěně, ale je nespolehlivý pro detekci menších objektů. Detekce, která je implementována ve výše uvedeném programu je založena na skutečnosti, že pokud je předmět (v našem případě tenisový míček) příliš blízko senzoru, tak senzor vrací hodnotu 255. Při pohledu do dokumentace zjistíme, že ultrazvukový senzor vrací hodnotu 255 pouze v případě, že v zorném poli senzoru není žádná překážka. Naše detekce je tedy založena na paradoxu, že pokud senzor nevidí žádnou překážku, tak je v jeho těsné blízkosti (cca do 7 cm) přepravovaný předmět. Z toho důvodu není tato detekce příliš spolehlivá. Detekovat předmět pomocí ostatních senzorů je bohužel nemožné, protože dotykové senzory nejsou dostatečně citlivé a světelný senzor je potřeba použít pro sledování čáry. Přestože naše řešení není příliš spolehlivé , tak se senzory, které máme k dispozici, žádné jiné patrně neexistuje.

Přestože to tedy není ideální řešení, tak žádné jiné, se senzory které máme k dispozici, nejspíše neexistuje.

Kapitola 4

Webové stránky

V rámci bakalářské práci byly rovněž vytvořeny webové stránky o programování v jazyce JAVA LeJOS a rovněž o obou soutěžních úlohách.

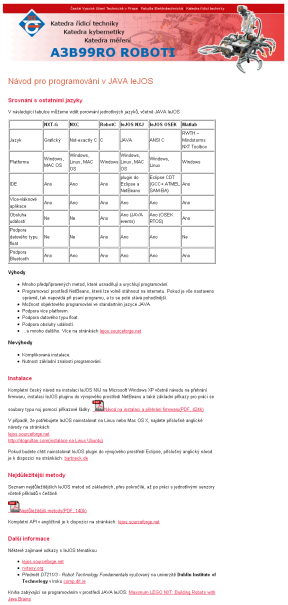
Při tvorbě webových stránek byly použity stejné kaskádové styly [14] a stejná hlavička [15], jaké se vyskytují na již fungujících webových stránkách o robotech [16]. Bylo tak učiněno z estetických důvodů, protože tyto nově vzniklé webové stránky budou k těm existujícím přidány.

Celkem vznikly 3 soubory typu html (*lejos_navod.html*, *krizeni.html*, *skladiste.html*), které obsahují návod pro programovací prostředí JAVA LeJOS a dvě výše uvedené soutěžní úlohy.

4.1 Webová stránka: Návod pro JAVA LeJOS

Na této webové stránce se uživatel dozví, jaké jsou výhody a nevýhody jazyka LeJOS a celkové srovnání jednotlivých programovacích jazyků, ve kterých je možné NXT programovat. Dále je uveden návod pro instalaci prostředí LeJOS a seznam jeho nejdůležitějších metod. V závěru stránky jsou uvedeny odkazy na webové stránky, které se také zabývají LeJOS tématikou.

Ukázku této webové stránky je na obrázku 4.1. Ukázky ostatních stránek nejsou z úsporných důvodů přidány.



Obrázek 4.1: Ukázka webové stránky o návodu pro JAVA LeJOS

4.2 Webová stránka: Sledování čáry s křížením

Na webové stránce o sledování čáry s křížením jsou uvedena pravidla a cíl této úlohy. Rovněž je popsán a vyobrazen soutěžní plán úlohy. Na závěr jsou pro inspiraci uvedeny odkazy na internetové stránky [8] a [9], kde je umístěno video s úspěšným vyřešením úlohy.

4.3 Webová stránka: Skladiště

Webová stránka o úloze skladiště je velmi podobná té předchozí. I zde jsou popsána pravidla a cíl úlohy, zobrazen soutěžní plán a uveden odkaz na internetovou stránku [10] s řešením úlohy.

Kapitola 5

Závěr

V této práci vznikl návod pro programovací prostředí JAVA LeJOS, jak v podobě PDF souboru, tak v podobě webových stránek a prezentace. Tento návod pokrývá základní, mírně pokročilé a senzorové příkazy, se kterými se v tomto prostředí pracuje a na příkladech ilustruje jejich použití. Programování v JAVA LeJOS je pro znalce jazyka JAVA velmi pohodlné, díky přívětivému prostředí NetBeans. Bohužel se ale vyskytly potíže s některými metodami, které LeJOS nabízí. Některé tu byly již zmíněné. Jedná se o chyby ve firmwaru, který se jistě časem doladí a problémy se odstraní.

Dále byly navrženy dvě soutěžní úlohy. Jedna s názvem Sledování čáry s křížením. Bylo ukázáno, že tuto úlohu lze řešit různými způsoby. Buď metodou sledování jedné čáry, ve které se uplatní klasický koncept řízení PD regulátorem, nebo metodou Zig-Zag, která využívá možností obou černých čar. Ta se opírá o správné naladění konstant a dobrý odhad v nastavení úhlu přejíždění. Druhá metoda umožnila nastavit vyšší rychlost robota díky využití druhé čáry, a tím se stala časově efektivnější.

Druhá soutěžní úloha, která byla navržena, byla nazvána Skladiště. Tato úloha simuluje situaci, která může nastat ve skladu při přemísťování jednotlivých kusů nákladu. Řešení se opírá o přesný sled instrukcí, které jsou postupně vykonávány, jak robot projíždí skladem. Bohužel toto řešení naráží na nedokonalost použitého ultrazvukového senzoru při detekci malých objektů, jako je tenisový míček. V praxi by toto omezení příliš nevadilo, protože se ve skladu obvykle převážejí mnohem větší objekty, než je tenisový míček nebo plechovka od nápoje.

Obě tyto úlohy byly vyřešeny v programovacích prostředích LeJOS a NXC. Vývoj jednotlivých programů byl zevrubně popsán slovně i formou vývojových diagramů a okomentované zdrojové kódy jsou k nalezení v příloze na CD. V grafickém prostředí NXT-G úlohy nebylo možné naprogramovat, protože byly příliš komplikované a jejich

zdárné řešení vyžadovalo užití vyššího programovacího jazyka.

V práci je rovněž uvedeno hardwarové řešení obou úloh. Dokumentace o konstrukci robota byly převzaty z různých internetových stránek, na které jsou uvedeny odkazy. Případné odlišnosti byly vysvětleny a výsledky fotograficky zaznamenány.

Pro obě úlohy byly vytvořeny webové stránky, které seznamují se základními pravidly a ukazují možné řešení jednotlivých úloh. Styl těchto webových stránek byl vytvořen tak, aby se nechaly přidat k již existujícím internetovým stránkám o předmětu A3B99RO Roboti. Z toho důvodu byly při tvorbě stránek použity stejné kaskádové styly a stejná hlavička, jaké jsou využity na těchto internetových stránkách.

Literatura

[1] LeJOS, Java for Lego Mindstorms [online]. 2006 [cit. 2010-05-03]. Introduction to leJOS NXJ. Dostupné z WWW: ⟨http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/Intro.htm⟩.

[2] ROBOTC.net [online].Carnegie Mellon University Robotics Academy, 2009 [cit. 2010-05-03]. Download ROBOTC for LEGO MINDSTORMS. Dostupné z WWW: ⟨http://www.robotc.net/download/nxt/⟩.

[3] Developer Resources for Java Technology [online]. c2010 [cit. 2010-05-03]. Java SE Downloads - Sun Developer Network (SDN). Dostupné z WWW: ⟨http://java.sun.com/javase/downloads/index.jsp⟩.

[4] LeJOS, Java for Lego Mindstorms [online]. 2006 [cit. 2010-05-03]. NXJ Downloads. Dostupné z WWW: ⟨http://lejos.sourceforge.net/nxj-downloads.php⟩.

[5] Welcome to NetBeans [online]. c2010 [cit. 2010-05-09]. Dostupné z WWW: ⟨http://netbeans.org/⟩

[6] LEGO® MINDSTORMS Education NXT Building Instructions [online]. [s.l.] : LEGO, 2006 [cit. 2010-04-30]. LEGO® MINDSTORMS Education NXT Building Instructions Driving Base, s. . Dostupné z WWW: ⟨http://cache.lego.com/downloads/education/BI_Driving_Base.pdf⟩.

[7] LEGO® MINDSTORMS Education NXT Building Instructions Light Module Down [online]. [s.l.] : LEGO, 2006 [cit. 2010-04-30]. LEGO® MINDSTORMS Education NXT Building Instructions, s. . Dostupné z WWW: ⟨http://cache.lego.com/downloads/education/BI_Light_Module_Down.pdf⟩.

[8] MARTINEC Dan. Youtube [online]. 2010 [cit. 2010-05-01]. NXT Line Follower With Crossroad Race. Dostupné z WWW: ⟨http://www.youtube.com/watch?v=QWUiX_Egaa0⟩.

[9] MARTINEC Dan. Youtube [online]. 2010 [cit. 2010-05-01]. NXT Zig-Zag Double Line Follower With Crossroad Race. Dostupné z WWW: ⟨http://www.youtube.com/watch?v=3MyySutVGHI⟩.

[10] MARTINEC Dan. Youtube [online]. 2010 [cit. 2010-05-07]. NXT Warehouse Automatic Electric Trolley Displacer. Dostupné z WWW: ⟨http://www.youtube.com/watch?v=0myJS6ED-B4⟩.

[11] ROBOTI — ČVUT - Katedra řídící techniky [online]. 2010 [cit. 2010-05-05]. Fotogalerie AMPER_2010. Dostupné z WWW: ⟨http://support.dce.felk.cvut.cz/roboti/fotogalerie/AMPER_2010/thumb.html⟩.

[12] NXT Programs - Fun Projects for your LEGO Mindstorms NXT [online]. 2007 [cit. 2010-05-04]. NXT Castor Bot. Dostupné z WWW: ⟨http://www.nxtprograms.com/castor_bot/steps.html⟩.

[13] NXT Programs - Fun Projects for your LEGO Mindstorms NXT [online]. 2007 [cit. 2010-05-04]. NXT Ball Hunter. Dostupné z WWW: ⟨http://www.nxtprograms.com/ball_hunter/steps.html⟩.

[14] ROBOTI — ČVUT - Katedra řídící techniky [online]. 2009 [cit. 2010-05-05]. Roboti.css. Dostupné z WWW: ⟨http://support.dce.felk.cvut.cz/roboti/roboti.css⟩.

[15] ROBOTI — ČVUT - Katedra řídící techniky [online]. 2009 [cit. 2010-05-05]. Roboti head. Dostupné z WWW: ⟨http://support.dce.felk.cvut.cz/roboti/img/head_cs.jpg⟩.

[16] ROBOTI — ČVUT - Katedra řídící techniky [online]. 2009 [cit. 2010-05-05]. Dostupné z WWW: ⟨http://support.dce.felk.cvut.cz/roboti/⟩.

Internetové zdroje:

[17] LeJOS, Java for Lego Mindstorms [online]. 2006 [cit. 2010-05-03]. Dostupné z WWW: ⟨http://lejos.sourceforge.net/⟩.

[18] LEGO.com MINDSTORMS : Home [online]. c2010 [cit. 2010-05-09]. Dostupné z WWW: ⟨http://mindstorms.lego.com/en-us/default.aspx⟩

[19] For LEGO Enthusiast's: BrickEngineer [online]. 2008-09-05 [cit. 2010-05-09]. LEGO NXT Motor Wiring — BrickEngineer. Dostupné z WWW: ⟨http://www.brickengineer.com/pages/2008/09/05/lego-nxt-motor-wiring/⟩.

[20] Team Hassenplug [online]. c2007 [cit. 2010-05-09]. NXT Programming Software. Dostupné z WWW: ⟨http://www.teamhassenplug.org/NXT/NXTSoftware.html⟩

[21] Robot Magazine - The latest hobby, science and consumer robotics, artificial intelligence [online]. c2009 [cit. 2010-05-09]. Programming Solutions for the LEGO Mindstorms NXT. Dostupné z WWW: ⟨http://www.botmag.com/articles/10-31-07_NXT.shtml⟩

[22] Philo's Home Page [online]. 2007 [cit. 2010-05-09]. NXT® motor internals. Dostupné z WWW: ⟨http://www.philohome.com/nxtmotor/nxtmotor.htm⟩

[23] SLUKA, Jim. Jim Sluka [online]. 2009 [cit. 2010-05-03]. PID Controller For Lego Mindstorms Robots. Dostupné z WWW: ⟨http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html⟩.

[24] HANSEN, John. NBC/NXC - NeXT Byte Codes and Not eXactly C [online]. 2007-10-10 [cit. 2010-05-03]. The NXC Guide. Dostupné z WWW: ⟨http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf⟩.

[25] BENEDETTELLI, Danny. Programming Lego Robots using NXC [online]. 2007-10-10 [cit. 2010-06-07]. The NXC Guide. Dostupné z WWW: ⟨http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf⟩.

Knížní zdroje:

[26] TROJÁNEK, Pavel. Využití robota LEGO MINDSTORMS při výuce. Praha, 2009. 94 s. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, Katedra řídící techniky. Dostupné z WWW: ⟨http://support.dce.felk.cvut.cz/e-kurzy/file.php/24/BP_Pavel_Trojanek.pdf⟩.

[27] BAGNALL , Brian. Maximum LEGO NXT : Building Robots with Java Brains. 2nd edition. [s.l.] : Variant Press, 2010. 524 s. ISBN 0973864958, ISBN-13: 978-0973864953.

[28] FERRARI, Mario; FERRARI, Guilio. Building Robots with LEGO Mindstorms NXT. Burlington (Massachusetts) : Syngress, 2007. 524 s. ISBN 1597491527, ISBN-13: 978-1597491525.

[29] GASPERI, Michael; HURBAIN, Philippe E.; HURBAIN, Isabelle L. Extreme NXT : Extending the LEGO MINDSTORMS NXT to the Next Level. [s.l.] : Apress, 2007. 312 s. ISBN 1590598180, ISBN-13: 978-1590598184.

Příloha A

Příloha CD

K této práci je přiloženo CD, na kterém je uložena elektronická verze této práce, vytvořené webové stránky, vzniklé zdrojové kódy a videa dokumentující vyřešení jednotlivých úloh.

- \Codes
  - \JAVA
    Tento adresář obsahuje všechny zdrojové kódy napsané v jazyce JAVA, které byly použity v této práci.
  - \C
    V tomto adresáři se nachází všechny zdrojové kódy napsané v jazyce NXC, které byly použity v této práci.
- \Documents
  Zde lze nalézt elektronickou verzi této práce, návod na instalaci LeJOS, ukázky nejdůležitějších metod z prostředí LeJOS, prezentaci o prostředí LeJOS a naskenované zadání bakalářské práce.
- \Video
  Tento adresář obsahuje videa, které dokumentují řešení jednotlivých úloh.
- \Web
  V tomto adresáři jsou uloženy webové stránky vytvořené v rámci této práce.