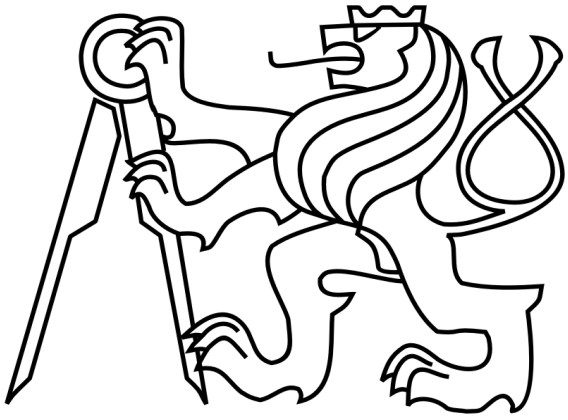
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

Katedra řídicí techniky



Využití robota LEGO Mindstorms – návrh a realizace úloh, návod pro programování v NXC

BAKALÁŘSKÁ PRÁCE

Praha 2010

Student: Ivan Moc

Vedoucí práce: Ing. Martin Hlinovský Ph.D.

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 10.5.2010

….....

podpis

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Martinu Hlinovskému Ph.D., za skvělou spolupráci, konstruktivní připomínky a zápal, se kterým k tomuto projektu přistupoval.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Ivan Moc**

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

Název tématu: **Využití robota Lego Mindstorms - návrh a realizace úloh, návod pro programování v NXC**

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 NXC (Not eXactly C)
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

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 27. 10. 2009

Abstrakt

Tato práce se věnuje výukovému prostředku LEGO Mindstorms a jeho využití v předmětu Roboti. Je zde popsána historie a hardware stavebnice, ale především se zabývá programováním robota pomocí jazyka NXC. Další částí práce jsou dvě soutěžní úlohy, včetně jejich pravidel a řešení. Poslední část práce se zabývá webovými stránkami k soutěžním úlohám.

Abstract

This bachelor thesis is about an education tool called LEGO Mindstorms and its usage in the subject called Robots. The introduction contains the description of its history and hardware, but the main part of the thesis covers the programming of robots in a language called NXC. The thesis also contains two competitive tasks including rules and solutions. The last part is about a web pages for competition.

Obsah

1 Úvod do bakalářské práce

1.1 Cíl práce

Cílem bakalářské práce je seznámit studenty předmětu Roboti se současným stavem robota LEGO Mindstorms, jeho hardwarem a softwarem. Připravit návod na programování robota v programovacím jazyku NXC pro úplné začátečníky, z tohoto návodu vytvořit prezentaci. Dalším cílem je návrh a realizace dvou soutěžních úloh, které budou studenti v rámci předmětu absolvovat. Posledním cílem práce je doplnit webové stránky o soutěžní úlohy.

Tuto práci jsem si vybral, protože chci přispět ke zkvalitnění výuky na Českém vysokém učení technickém v Praze. Robot LEGO Mindstorms se mi jeví jako ideální výukový prostředek, který si studenti oblíbí. V neposlední řadě jsem si tuto práci vybral, protože jako dítě jsem si s legem hrál velmi intenzivně.

1.2 Obsah práce

Práce je kromě úvodu a závěru rozdělena na čtyři části. Druhá kapitola pojednává o historii LEGO robotů a hardwarovém vybavení. V třetí kapitole je programátorská příručka jazyka NXC, obsahující veškeré informace potřebné pro programování robota. Čtvrtá kapitola obsahuje rozbor soutěžních úloh včetně pravidel, konstrukčního i programového řešení. Pátá kapitola se zabývá webovými stránkami.

2 Seznámení se s robotem LEGO Mindstorms

2.1 Úvod do druhé části

Robot LEGO Mindstorms představuje výukový prostředek používaný na mnoha světových univerzitách, mezi ně patří například:

Union College New York [5]

Tufts University Boston [6]

ETH Curych [7]

ČVUT Praha [8]

Baťova univerzita ve Zlíně [9]

Robot LEGO Mindstorms však neslouží pouze akademickým účelům, ale především dětem široké věkové kategorie, pro které byl stvořen. Na Bostonské univerzitě Tufts nabízejí semináře pro děti od předškolního věku, které jsou zaměřené na poznávání lega, přes semináře pro děti od sedmi do deseti let, které se setkávají s jednoduššími programovacími úlohami, až po semináře pro studenty univerzity, kteří řeší podobné úlohy jako studenti ČVUT FEL v předmětech Roboti či Systémy a modely.

Díky stavebnici LEGO Mindstorms se tak mohou studenti hravou formou seznamovat se základy programování, řízení, mechaniky a týmové práce v jakémkoliv věku.

2.2 Robot LEGO Mindstorms

Historie robota Mindstorms sahá do roku 1980, kdy byla ve společnosti LEGO založena divize vzdělávacích produktů. Ve spolupráci s Massachusetts Institute of Technology vznikla mezi roky 1988 a 1998 první inteligentní „kostka“, označená RXC. Programovala se v jazyce

NQC. V roce 2006 přišla nová řada označená NXT 1.0. Programovat se dá pomocí několika různých jazyků (LabView [13], C [11], Java [12], Matlab [13] a další). Od roku 2009 je v prodeji stavebnice označená jako NXC 2.0. Oproti verzi 1.0 se liší menším počtem LEGO dílků a světelný senzor byl nahrazen senzorem barev, více o rozdílech mezi stavebnicemi ve zdroji [14]. Kostka NXT 2.0 se liší použitím operací s plovoucí desetinnou čárkou, zatímco NXT 1.0 pracuje s integer operacemi [15]. Studenti v předmětu Roboti používají stavebnici NXT 1.0 v podobě Education setu.

2.3 Využití robota pro rozvoj schopností studentů

Robot LEGO Mindstorms je koncipován jako výukový prostředek. Studenti, kteří s ním pracují, se seznámí s různými obory elektrotechniky, díky kterým si rozšíří znalosti v oblastech, které je zajímají.

Programování, algoritmizace

Program je nezbytnou součástí každého robota. NXT je proto vhodným nástrojem pro výuku programování. Počínaje procedurálním programováním, přes objektově orientované až po speciální, jako například assambler, vícevláknové programy, či řízení dalších kostek. Programováním se studenti naučí vytvářet algoritmy, které si okamžitě mohou vyzkoušet v praxi s hmatatelným výsledkem.

Senzory

Díky použitým senzorům se studenti dostanou k tématům jako nejistoty měření, zpracování signálů, vnitřní struktura senzorů a další.

Řízení

Studenti se u robota nevědomky setkávají se zpětnou vazbou, která je základním kamenem řízení. Díky senzorům dostává robot informace ze

svého okolí a na algoritmu studenta je, aby ideálně reagoval pomocí této zpětné vazby.

Mechanika

Konstrukční možnosti robota jsou takřka neomezené. Díky množství zubatých kol, řetězů a dalších konstrukčních prvků jdou sestrojit složitá mechanická zařízení. Konstrukce vyžaduje určitou pevnost a stabilitu, lehčí robot ale může být rychlejší.

Týmová práce

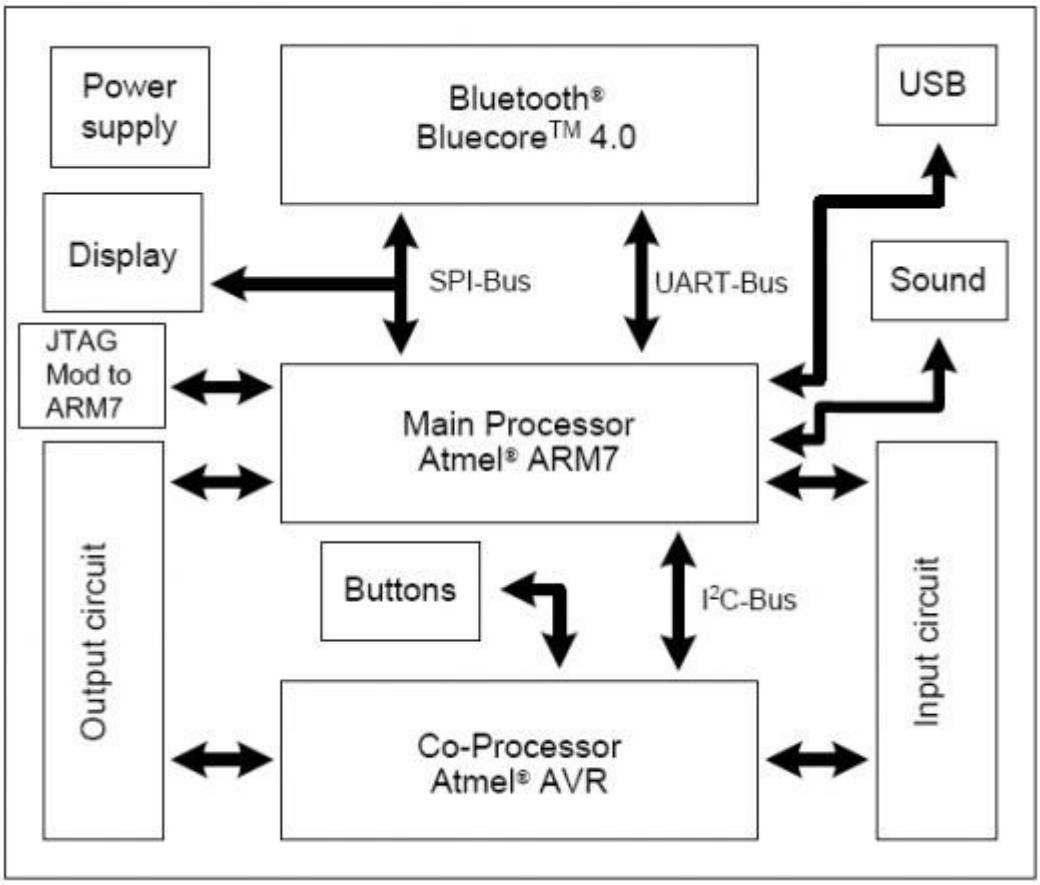
V neposlední řadě si studenti procvičí týmovou práci, soutěž mezi jednotlivými týmy tuto spolupráci ještě vylepší. Dlouhodobá práce na projektu naučí studenty efektivně plánovat a stanovovat priority.

2.4 Hardwarové vybavení robota

2.4.1 Řídicí jednotka

Řídicí jednotka, označovaná jako „kostka“, či „inteligentní kostka“ obsahuje 32-bitový RISC procesor Atmel ARM7 (AT91SAM7S256) běžící na 48Mhz s 256kB flash paměti a 64kB paměti RAM [16]. Dále je obsažen koprocesor Atmel AVR (Atmega48). Zobrazení blokového uspořádání kostky je uvedeno na Obr. 2.1.

Kostka komunikuje s počítačem pomocí technologie USB 2.0 (12Mbit/s), nebo pomocí Bluetooth (460Kbit/s). Displej na kostce je bodový, s rozlišením 100x64 pixelů. Napájení kostky zaručuje šestice AA baterií, nebo Li-Ion akumulátor.



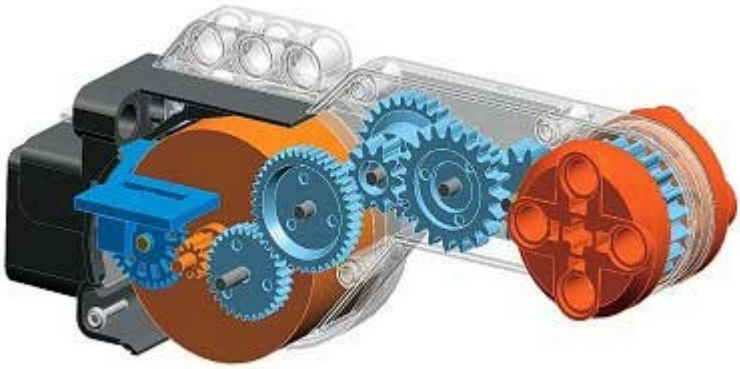
Obr. 2.1: Blokové schéma NXT kostky

Kostka obsahuje tři výstupní porty a čtyři vstupní. Na výstupní se připojují motory. Vstupní porty umožňují připojení jak analogového tak digitálního senzoru. V případě analogového senzoru je za vstupem 10-bitový A/D převodník a senzor je napájen ze zdroje proudu. V případě digitálního senzoru je komunikace zajištěna pomocí I^2C protokolu s rychlostí 9600bit/s.

Kostka je dále vybavena reproduktorem, s jehož pomocí může přehrávat zvukové stopy ve formátu .rso, nebo umí sama generovat tóny o určité frekvenci.

2.4.2 Motory

Na výstupní porty kostky lze připojit až tři stejnosměrné elektromotory. Napájení je 9V a otáčky jsou regulovány pomocí pulzně-šířkové modulace. Samotný elektromotor se může otáčet až 1500 otáček za minutu. Na motor je napojena série ozubených kol, které tvoří převod 1:48. V motoru je dále zabudovaný optický inkrementální snímač s rozlišením 360 hodnot na jednu otáčku, umožňující sledování otáček a pokročilé řízení. Podrobný vnitřek motoru a zátěžové charakteristiky uvádí zdroj [17].



Obr. 2.2: Vnitřní uspořádání NXT motoru

2.4.3 Senzory

Světelný senzor

Světelný senzor umožňuje měřit úroveň okolního osvětlení nebo měřit míru odraženého světla z přisvětlující LED diody. Tak můžeme detekovat změny v barvě měřeného objektu. Fototranzistor je citlivý především na infračervené záření.

Obr. 2.3: Světelný senzor

Dotykový senzor

Dotykový senzor umožňuje reagovat na přímý kontakt. Měří se změna napětí, které se mění spojováním a rozpojováním částí jednoduchého obvodu.

Obr. 2.4: Dotykový senzor

Ultrazvukový senzor

Ultrazvukový senzor je jediný digitální. To znamená, že obsahuje obvody pro zpracování hodnot ze samotných čidel a s kostkou komunikuje pomocí I^2C protokolu. Ultrazvukový senzor pracuje na bázi sonaru.
Z vysílací části senzoru je vypuštěna ultrazvuková vlna, která se odrazí od překážky a je zpátky zachycena přijímací částí senzoru. Ze známé rychlosti

šíření vlny a časového rozdílu je pak vypočtena vzdálenost překážky. Senzor měří až do vzdálenosti 2,5m s přesností ±3cm. Pokud senzor měří vzdálenost k dostatečně rozměrné překážce, tvořené plochou, pak je měření přesné i na vzdálenosti blížící se hornímu limitu použitelnosti. Při detekci menších objektů, které nejsou uniformní, je měření i na malé vzdálenosti značně nespolehlivé.

Obr. 2.5: Ultrazvukový senzor

Zvukový senzor

Zvukový senzor měří hladinu okolního hluku v dB, nebo dBA. Rozsah senzoru je přibližně 50-90dB [18].

Obr. 2.6: Zvukový senzor

3 Programování LEGO robota Mindstorms v NXC

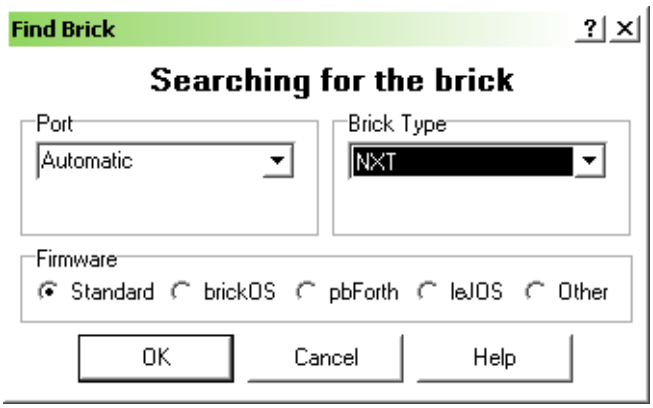
3.1 Úvod do třetí části

Tato část pojednává o programování LEGO robota v jazyku NXC pro úplné začátečníky. Uživatel se zde seznámí se základy jazyka, strukturou programu, ovládání motorů a senzorů. Celá část je zaměřená na příklady použití příkazů v reálném programu.

3.2 Vývojové prostředí NXC

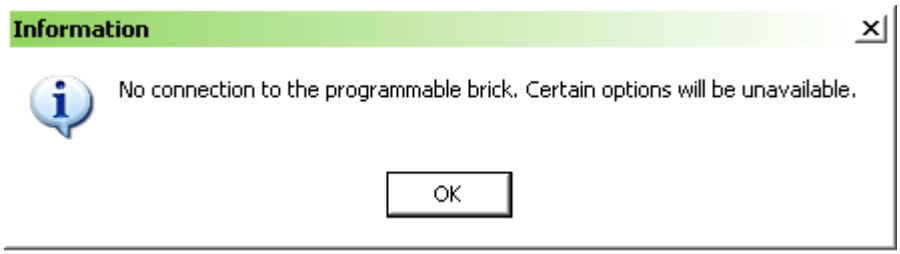
Jazyk NXC, který je odvozen od jazyka C, běží na standardním firmwaru LEGO robota. Pro navázání komunikace robota s PC proto stačí jen ovladač standardně dodávaný s robotem, nebo k dispozici na http://mindstorms.lego.com/en-us/support/files/Driver.aspx. Instalace ovladače je intuitivní a nevyžaduje podrobnější návod. Je však nutno vzít na vědomí, že ovladač je dostupný pouze pro operační systémy Microsoft Windows a MacOS a jen ve 32-bitové verzi.

Vývojové prostředí Bricx Command Centre, známé pod zkratkou BricxCC je volně dostupné na stránce http://bricxcc.sourceforge.net/. Instalace je stejně snadná jako u ovladače a program funguje pod všemi verzemi Microsoft Windows. Při každém spuštění vyzve program k identifikaci LEGO kostky, náhled je na Obr. 3.1.



Obr. 3.1: Výzva k identifikaci LEGO kostky

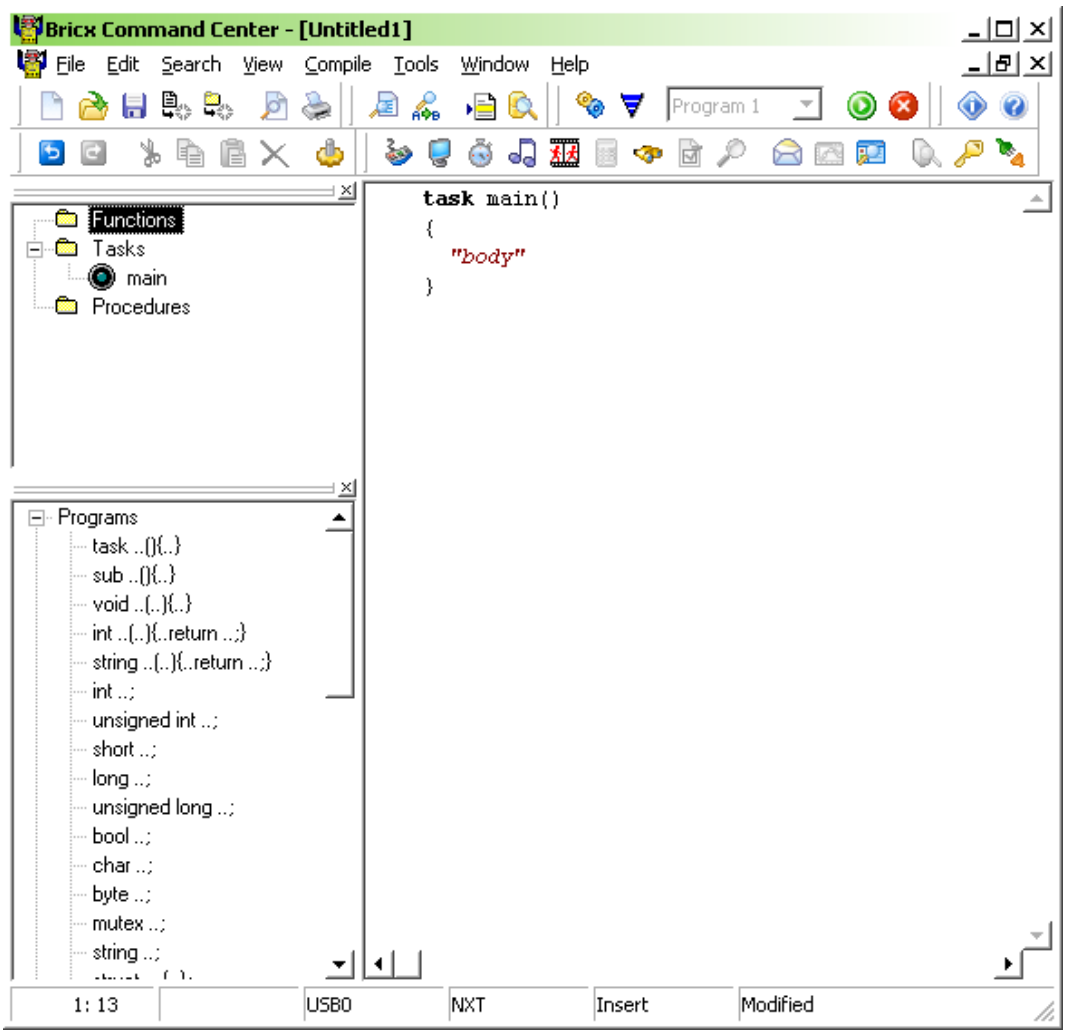
Port volíme buď automaticky, nebo USB, typ kostky NXT, firmware standard. Pokud nemáme připojenou kostku, nebo jsme špatně zvolili parametry, program zahlásí chybu, viz. Obr. 3.2.



Obr. 3.2: Chybová hláška při nepřipojené kostce

Poté se již dostaneme do samotného programovacího prostředí. Pokud není kostka stále inicializovaná, můžeme opětovnou inicializaci vyvolat pomocí menu Tools → Find Brick.

Hlavní okno BricxCC vypadá jako obyčejný textový editor. Do hlavního okna píšeme samotný program, vlevo nahoře je seznam funkcí, úkolů a procedur obsažených v našem programu. Vlevo dole se nachází seznam nejpoužívanějších výrazů uspořádaných podle funkcí. Tento seznam se zapíná pomocí klávesy F9 nebo přes menu View → Templates. Je to velice užitečná funkce, protože si programátor nemusí pamatovat exaktně syntaxi příkazů nebo kvůli nim často nahlížet do manuálu. Dvojklikem můžeme tyto funkce přidávat do těla programu. Náhled programovacího prostředí je na Obr. 3.3.



Obr. 3.3: Hlavní okno Bricx Command Centre

3.3 Představení NXC

Jazyk NXC vychází z jazyka C, zdědil z něj proto nejzákladnější pravidla. Jde o jazyk, který zohledňuje velká a malá písmena. To znamená, že proměnná *Abc* je jiná než *ABC*, stejně tak klíčová slova jako jsou *if* nebo *while* jsou pevně definována a výraz *IF* není považován za klíčové slovo, ale za proměnnou. Každá deklarace je zakončena středníkem a každá struktura je uzavřena dvojicí složených závorek.

```
task main(){     //struktura main
      int A=10000;     //deklarace proměnné
      OnFwd(OUT_AC,100); //deklarace pro spuštění
motoru
```

```
        Wait(A);   //deklarace pro čekání
        Off(OUT_AC);    //deklarace pro vypnutí
motoru
}     //konec struktury main
```

3.4 Komentáře

Každý kód, byť i sebejednodušší, by měl být okomentován. Při psaní delšího programu to usnadňuje orientaci v kódu a pro ostatní je jednodušší pochopit, jak programátor přemýšlel, když programoval. Komentáře jsou čistě pro potřeby programátora a interpret kódu je při překladu ignoruje. Existují dva druhy komentářů, jednořádkové a víceřádkové. Jednořádkový komentář má pouze nepárovou značku „//“, zatímco víceřádkový komentář používá značky párové a to „/*“ na začátku komentáře a „*/“ na konci. Vše, co je za značkou komentáře, je bráno jako prostý text.

```
OnRew(OUT_A, 50); //toto je komentář
/*toto je také komentář*/
```

3.5 Úkoly

Každý NXC program musí obsahovat alespoň jeden úkol, pojmenovaný *main*, který **musí** být umístěn v kódu jako poslední, jinak dojde k chybě překladu. Program může obsahovat nejvýše 255 běžících úkolů najednou.

Úkoly nejčastěji spouštíme pomocí příkazu *StartTask(jméno úkolu)*. Příkaz *Follows(úkol_1, úkol_2,...úkol_N)* spustí úkoly v řadě za sebou, poté co je předchozí úkol dokončen. Příkaz *Precedes(úkol_1, úkol_2,...úkol_N)* spustí úkoly souběžně, pokud tomu nebrání jiné okolnosti. Zastavit úkol můžeme pomocí příkazu *StopTask(jméno úkolu)*, všechny úkoly najednou zastavíme příkazem *StopAllTasks()*, tím se ale nezastaví samotný program. K tomu slouží příkaz *Stop()*. Pomocí příkazu *ExitTo(jméno dalšího úkolu)* ukončí současný úkol a zároveň se spustí specifikovaný.

Pokud úkoly přistupují ke společným zdrojům, např. motorům, pak musíme použít speciální typ proměnné nazvané mutex. Mutex je typ semaforu, který nedovolí přistupovat ke sdíleným prostředkům jinému úkolu než tomu, který má „zelenou“. Funkci úkolů a mutexu si ukážeme na příkladu:

```
mutex sem;
task pohyb(){
	while(true){
		Acquire(sem);
		OnFwd(OUT_AB,75);
		Wait(5000);
		OnRev(OUT_A,75);
		Wait(1000);
		Release(sem);
	}
}
task senzor(){
	while(true){
		if(SENSOR_1 == 1){
			Acquire(sem);
			OnRev(OUT_AB,75);
			Wait(1000);
			OnFwd(OUT_B,75);
			Wait(1000);
			Release(sem);
		}
	}
}
task main(){
	SetSensorTouch(IN_1);
	Precedes(pohyb,senzor);
}
```

Pro plné pochopení příkladu je vhodné nastudovat práci senzorů a motorů. Program se skládá ze tří úkolů. V hlavním *main*, nastavujeme dotykový senzor a spouštíme další dva úkoly. V úkolu *pohyb* je nekonečná smyčka, která způsobuje, že robot jezdí do čtverce. Třetí úkol *senzor*

kontroluje dotykový senzor, je-li stisknut, pak robot kousek zacouvá a zatočí. Semafor je zde použit, aby se oba úkoly nesnažily hýbat motory zároveň. Zavolá-li se funkce *Acquire,* pak druhý úkol nemůže přistupovat na motory. Funkcí *Release* jsou motory opět dány k dispozici všem úkolům, než se o ně přihlásí některý z nich pomocí *Acquire*.

3.6 Procedury

Procedury využíváme, když potřebujeme část kódu použít na více místech programu. Do procedury můžeme poslat vstupní argumenty, takže se pokaždé může chovat jinak, zatímco kód zůstává stejný. Procedury mohou také vracet hodnotu, která se uloží do proměnné, jež proceduru vyvolala. Toho využijeme např. pokud procedura provádí složitější výpočet.

```
sub zatoc(int tah){        //procedura pro zatočení
o 90°, směr volíme argumentem
      OnFwd(OUT_A, -tah);
      OnFwd(OUT_B, tah);
      Wait(200);
}
task main{
      OnFwd(OUT_AB,100);    //robot jede rovně
      Wait(5000);
      zatoc(75); //volání zatočení, směr zatočení
volíme argumentem
      OnFwd(OUT_AB,100);
      Wait(5000);
      zatoc(-75);     //opětovné volání zatočení,
dojde k otočení na opačnou stranu než v předchozím
případě
      Off(OUT_AB);
}
```

Syntaxe pro procedury je:

```
návratový typ jméno(argumenty)
```

Návratový typ - pokud procedura nevrací žádnou hodnotu, používáme *void* nebo *sub.* Pokud procedura vrací nějakou hodnotu, její typ je zde

definován.

```
int proc0(){    //tato procedura nemá žádný vstup
a výstupem je integer x
     int x=10;
     return x;       //pomocí příkazu return
určujeme kterou proměnnou procedura vrací
}
```

Pokud chceme takovouto proceduru zavolat, použijeme následující kód, do proměnné *a* se uloží výstup procedury, neboli *x*:

```
int a=proc0();
```

Argumenty - jsou uvedeny jako typ proměnné následované jménem.

```
void proc1(int a,int b,bool c)
```

Do této procedury vstupují tři argumenty a procedura nevrací žádná data. Zavoláme jí pomocí volání procedury:

```
proc1(10, 100, true);
```

Na odpovídajících místech v závorce musí být datové typy, které jsme stanovili u procedury.

3.7 Proměnné a jejich deklarace

V NXC existuje jedenáct typů proměnných, viz. Tabulka 3.1

Klíčové slovo proměnné	Informace
`bool`	true/false
`byte`	0 až 255
`char`	Znaky
`unsigned int`	0 až 65535
`int`	-32768 až 32767
`unsigned long`	0 až 4 294 967 295
`long`	−2 147 483 648 až 2 147 483 647
`mutex`	Speciální proměnná, viz. kapitola 3.5
`string`	Textové řetězce

`struct`	Uživatelem definovaná struktura
`klíčové slovo proměnné``[]`	Pole jakéhokoliv typu

Tab. 3.1. Typy proměnných v NXC

Proměnné jsou deklarovány pomocí klíčového slova proměnné, následovaného jménem proměnné, volitelně rovnítkem a hodnotou. Výraz je ukončen středníkem. Několik příkladů deklarace proměnné:

```
int a;
long b1,b2;
bool c=true;
int d=10;
```

Definujeme-li více proměnných, oddělujeme je čárkou. Program rozlišuje proměnné na lokální a globální. Lokální proměnné jsou dostupné pouze v proceduře či úkolu, kde jsou deklarovány. Co jsou úkoly a procedury se dočtete v kapitolách 3.5 a 3.6. Globální proměnné jsou viditelné všem úkolům a procedurám a musí být deklarovány mimo ně.

3.8 Struktury proměnných

Jazyk NXC umožňuje vytvářet uživatelské struktury více proměnných. Používají se na sdružení proměnných, které mají něco společného, uplatnění najdou především ve složitějších programech. Jak fungují si ukážeme na příkladu:

```
struct auto{    //vytvoření nové struktury typu auto
     int pocet_dveri;
     string znacka;
     int rok_vyroby;
};
```

Poté, co vytvoříme strukturu, můžeme na její proměnné přistupovat pomocí jména proměnné (struktury), tečky a jména vnitřní proměnné:

```
auto moje_auto;    //vytvoření nové proměnné typu auto
```

```
moje_auto.pocet_dveri=5;      /*přístup k proměnné
uvnitř struktury*/
moje_auto.rok_vyroby=1995;
```

3.9 Pole proměnných

NXC také podporuje pole proměnných. Pole deklarujeme podobně jako samostatné proměnné, jen za název přidáme dvojici hranatých závorek. Přidáním druhé dvojice hranatých závorek vytvoříme pole o dvou dimenzích. Největší počet dimenzí podporovaných NXC je čtyři. Prvky pole jsou číslovány od nuly.

```
int pole[];     //vytvoří pole s nula prvky typu
int
int pole2[] []; /*vytvoří pole dvou dimenzí s nula
prvky typu int*/
int pole3[5];      /*vytvoří pole s pěti prvky,
výchozí hodnotou jsou nuly*/
int pole4[]={1, 7, 3, 1, 4, 1};      /*vytvoří
pole, které je naplněno, pokud specifikujeme
hodnoty výčtem, tak nemusíme udávat délku pole
v hranatých závorkách */
pole[0]=12;      /*přiřadí do pole na nultou pozici
hodnotu, z pole se nyní stává pole délky 1 */
pole[1]=2;        //přiřadí do pole na pozici
s číslem 1 hodnotu, délka pole naroste na hodnotu 2
```

Pokud vytváříme prázdná pole, pak je před jejich použitím musíme inicializovat pomocí funkce ArrayInit a určit jejich velikost.

```
int vector[];
ArrayInit(vector, 1, 15);  //inicializace pole, do
proměnné vector se uloží patnáct jedniček
int vector[15]; //inicializace pole již není nutná,
pole má specifikovanou délku a na všech pozicích
jsou nuly
```

3.10 Přiřazení a další práce s proměnnými

Dosud jsme se setkali jen s přiřazením hodnoty do proměnné pomocí rovnítka. Proměnné ale můžeme mezi sebou porovnávat, sčítat a provádět

další operace.

Operátor	Funkce
=	Přiřazení výrazu do proměnné
+=	Přičtení výrazu k proměnné
-=	Odečte výraz od proměnné
*=	Vynásobí proměnnou výrazem
/=	Vydělí proměnnou výrazem
%=	Zbytek po dělení proměnné výrazem
&=	Bitový AND proměnné a výrazu
|=	Bitový OR proměnné a výrazu
^=	Bitový XOR proměnné a výrazu
||=	Absolutní hodnota výrazu
+-=	Do proměnné nastaví signum výrazu (-1, 0,1)
>>=	Bitový posun doprava o počet bitů určených výrazem
<<=	Bitový posun doleva o počet bitů určených výrazem
++	Zvýšení hodnoty proměnné o 1
--	Snížení hodnoty proměnné o 1

Tabulka 3.1: Operace s proměnnými

Příklady:

```
int x=15;
int y=7;
x %=y;      //x bude v tomto případě 1
x+=x;       //k x se přičte x, takže x=2
x*=y;       //do x se zapíše hodnota x*y neboli 14
x++;        //zvýšení hodnoty x o 1
```

Porovnávání proměnných se používá v případě podmínek, které probereme v následující kapitole

Operátor	Funkce
==	je rovno
<	je menší
>	je větší
<=	je menší nebo rovno
>=	je větší nebo rovno
!=	není rovno
true	ano
false	ne
&&	a zároveň
||	nebo

Tabulka 3.2: Porovnávání proměnných

3.11 Preprocesor

Preprocesor v NXC vychází z jazyka C. Preprocesor je část programu, která se zpracovává ještě před samotným překladem kódu. Používá se k implementování externích knihoven, definování maker či proměnných. NXC obsahuje všechny knihovny potřebné k běhu základního robota implicitně.

```
#include "knihovna.h" //příkaz pro implementaci knihovny
#define vzdalenost 15 //příkaz pro definování proměnné či makra
```

Preprocesor se chová odlišně než standardní kód. Při deklaraci proměnné neuvádíme typ proměnné, příkaz není ukončen středníkem a hodnota proměnné nejde během běhu programu změnit.

3.12 Podmínky a cykly

Podmínky a cykly jsou nejpoužívanější konstrukce programu, umožňují nám vykonávat části programu za předpokladu, že je splněna daná podmínka a v případě cyklů periodické opakování kódu.

3.12.1 Podmínka if a else

Je-li podmínka stanovená u příkazu splněna, pak se vykoná kód, který následuje. Není-li splněna, pak se kód v podmínce přeskočí. Podmínek může být více, spojených operátory && nebo ||.

```
if(a==10){ //podmínka
    /*zde jsou procedury, které se vykonají, je-li
podmínka splněna*/
}
if(a==5 && b|=true){
   //if s více podmínkami
}
```

Výraz *if* může být doplněn volitelně výrazem *else*, který spustí kód obsažený v proceduře else pokud není splněna podmínka v *if*

```
if(a==10){
    /*zde jsou procedury, které se vykonají, je-li
podmínka splněna*/
}else{
    /*zde jsou procedury, které se vykonají, není-
li podmínka splněna*/
}
```

3.12.2 Cyklus while a do-while

Cyklus *while* se vykonává stále dokola, je-li splněna podmínka pro jeho běh.

```
while(a==10){          //podmínka
    /*zde jsou procedury, které se vykonají, je-li
podmínka splněna*/
}
```

Dáme-li do podmínky klíčové slovo *true*, pak cyklus běží donekonečna. Z takového cyklu můžeme vyskočit např. příkazem *break*.

Obdobou cyklu *while* je cyklus *do-while*, který se liší od cyklu *while* tím, že je vždy spuštěn alespoň jednou, nezávisle na podmínce, která se kontroluje až po prvním průchodu. Cyklus *while* kontroluje podmínku před

průchodem, proto se nemusí spustit nikdy.

```
do{
    /*zde jsou procedury, které se vykonají, je-li
podmínka splněna a při prvním průchodu*/
}while(x==true)
```

3.12.3 Cyklus for

Dalším cyklem je příkaz *for,* jež má tři části:

```
for(výraz1; podmínka; výraz2){
     //tělo cyklu
}
```

Při běhu programu se nejprve jednou zavolá *výraz1*, poté se kontroluje podmínka. Je-li splněna, vykonají se procedury uvnitř cyklu a nakonec cyklu se vykoná *výraz2.* Poté cyklus končí a znovu se kontroluje podmínka, pomocí cyklu *while* bychom to zapsali:

```
výraz1;
while(podmínka){
    //tělo cyklu
    výraz2;
}
```

A reálné použití cyklu for

```
for(int i=0; i<10; i++){
    //tělo cyklu
}/*tento cyklus proběhne 10x, začíná s i=0, které
se při každém cyklu zvýší o 1*/
```

3.12.4 Cyklus repeat

Jednoduchým cyklem je příkaz *repeat*, kterým opakujeme vykonávání kódu.

```
repeat(4){
     //tělo cyklu, které se vykoná 4x
}
```

3.12.5 Přepínač switch

Přepínač neboli *switch* je příkaz, pomocí kterého můžeme spouštět různý kód v závislosti na podmínce. Po příkazu *switch* následuje výčet případů *case 1* až *case n*, volitelně doplněných příkazem *default*, který se spustí, není-li splněna ani jedna podmínka daná příkazy *case*.

```
switch(Y){
      case 1:{
            //kód, který se vykoná v případě Y=1
      }
      case 2:{
            //kód který se vykoná v případě Y=2
      }
      case 3:{
            //kód který se vykoná v případě Y=3
      }
      default:{
                  //kód který se vykoná, je-li Y
různé od 1,2,3
      }
}
```

3.12.6 Funkce until

Předposledním druhem cyklu je funkce *until*, která je velmi užitečná a používá se ve spojení se senzory. Dokud není splněna podmínka specifikovaná v *until*, program čeká.

```
task main(){
      SetSensor(IN_1,SENSOR_TOUCH);    //nastavení
senzoru
      OnFwd(OUT_AC,100);    //zapnutí motoru
      until(SENSOR_1 == 1);/*robot jede, dokud není
stisknut dotykový senzor*/
      Off(OUT_AC);    //vypnutí motoru
}
```

3.12.7 Funkce goto

Posledním cyklem je příkaz *goto*, kterým se můžeme vrátit na specifikované místo kódu. Používání příkazu *goto* bychom se měli vyvarovat, klasické cykly *while*, *for* nebo *if* jsou z hlediska přehlednosti

kódu mnohem vhodnější.

```
cyklus:
      x++; //tělo cyklu
      goto cyklus;    //vrácení se na cyklus
```

3.13 Časové funkce

V programu často potřebujeme, aby se s vykonáváním dalších příkazů počkalo, než robot někam dojede či se otočí. K tomu slouží funkce *Wait(čas v ms)*.

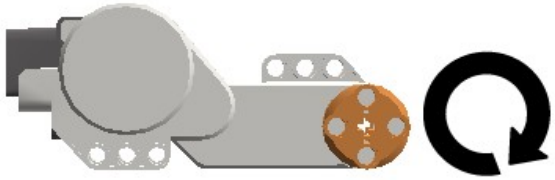
```
task main(){
      OnFwd(OUT_AB, 75);    //robot se rozjede rovně
      Wait(3000);      //program čeká 3s než vykoná
další řádek kódu, tj. robot pojede 3s než se
zastaví
      Off(OUT_AB);    //vypnutí motorů
}
```

3.14 Práce s motory

NXT podporuje až tři motory zapojené do kostky. Každý motorový výstup má svou identifikační proměnnou označenou *OUT_A*, *OUT_B* a *OUT_C*, které jsou přiřazeny odpovídajícím výstupům na kostce. Výstupy můžeme mezi sebou libovolně kombinovat a tím je spouštět najednou: *OUT_AB, OUT_AC, OUT_BC, OUT_ABC*. Každý motor má v sobě senzor otáček, který se využívá při pokročilejším řízení běhu motorů.

3.14.1 Základní pohyb

Pohyb vpřed provedeme funkcí *OnFwd(výstup, tah)*, na obrázku 3.4 je znázorněn směr rotace motoru.



Obr. 3.4: Směr otáčení motoru při použití příkazu OnFwd

Tah je v rozmezí -100 až 100, motor se tedy může pomocí této funkce točit na obě strany. Z hlediska přehlednosti programu je ale vhodnější, pokud pro couvání použijeme funkci *OnRev(výstup, tah)*. Motor zastavíme funkcí *Off(výstup)*. Funkce *Off* okamžitě zastaví motor pomocí brzdy. Pokud nepotřebujeme, aby robot zastavil přesně na místě, je vhodnější použít funkce *Float(výstup)*, která motor pouze odpojí od napájení a ten se zastaví setrvačností. V následujícím příkladě robot jede 10x do čtverce.

```
task main(){
      repeat(10){
            repeat(4){
                  OnFwd(OUT_AB,75);     //pohyb
dopředu, zapnuty jsou motory na výstupech A a B
                  Wait(1000);
                  OnRev(OUT_A,75);//zatočení, jeden
motor se točí na druhou stranu
                  Wait(500); //POZOR, čas potřebný
pro zatočení závisí na tahu motoru, konstrukci
robota, povrchu podložky a dalších faktorech. Vždy
je potřeba experimentovat s hodnotami tak, aby bylo
dosaženo kýženého výsledku!
            }
      }
      Float(OUT_AB);        //vypnutí motorů
}
```

3.14.2 Pokročilé metody pohybu

Motory NXT v sobě obsahují senzor polohy hřídele, takže umožňují přesné řízení rychlosti a polohy hřídele pomocí PID (proporciálně-integračně-derivačního) regulátoru. *OnFwdReg(výstupy, rychlost, mód regulátoru)* řídí motor podle módu regulátoru. *OUT_REGMODE_IDLE* je

mód, při kterém je regulátor nečinný a motor se chová jako při použití funkce *OnFwd*. *OUT_REGMODE_SPEED* reguluje rychlosti motorů tak, aby se točili stejně rychle. Pokud se jedno kolo začne točit pomaleji (např. v důsledku překážky), regulátor zasáhne a zvýší jeho tah, aby byla rychlost konstantní. *OUT_REGMODE_SYNC* synchronizuje otáčky motorů, aby byly stejné. Pokud jeden motor zastaví, druhý zastaví automaticky také. Regulační mód funguje i pro pohyb dozadu, pouze klíčové slovo je *OnRevReg*. Všechny tyto módy fungují pouze při volbě kombinovaných výstupů.

RotateMotor(výstup, tah, úhel) otočí motor o požadovaný úhel (ve stupních).

```
RotateMotor(OUT_A, 75, 45);      //otočí motor o 45° po směru otáčení
RotateMotor(OUT_A, -75, 45);     //otočí motor o 45° proti směru otáčení
```

RotateMotorPID(výstup, tah, úhel, P, I, D) otočí motor o požadovaný úhel pomocí PID regulátoru. Konstanty P-proporcionální, I-integrační a D-derivační určují parametry otočení, jako je rychlost, doba ustálení, překmit a další.

3.14.3 Motor jako senzor

Díky zabudovanému snímači můžeme využít další funkce. Před prací s optickým inkrementálním snímačem je vhodné resetovat snímač pomocí funkce *ResetTachoCount(výstup)*. Hodnotu senzoru načteme funkcí *MotorBlockTachoCount(výstup)*. Program pro měření otáček motoru za jednu minutu pak může vypadat takto:

```
ResetTachoCount(OUT_A);
int rot_start=MotorBlockTachoCount(OUT_A);
Wait(1000);
int rot_end=MotorBlockTachoCount(OUT_A);
```

```
speed=(rot_end-rot_start)*60/360; //motor má
rozlišení 360 hodnot na otáčku, proto je hodnotu
nutné vydělit 360ti
```

Pro přesnější měření je vhodné spustit program v cyklu, hodnoty ukládat do pole a rychlost počítat z většího objemu dat.

3.15 Práce se senzory

NXT podporuje až čtyři senzory zapojené do kostky, číslované od 1 do 4. Proto se používají proměnné určující port IN_1, IN_2, IN_3, IN_4 a k nim proměnné určující hodnotu ze senzoru SENSOR_1, SENSOR_2, SENSOR_3 a SENSOR_4 (případně S1, S2, S3 a S4) . Načíst hodnotu analogového senzoru můžeme takto:

```
x= SENSOR_1;    //přečte hodnotu senzoru na portu 1
a zapíše jí do x
x= Sensor(S1);        //stejná funkce, jiný zápis
```

Hodnotu z digitálního ultrazvukového senzoru musíme číst pomocí funkce:

```
x= SensorUS(S1);      //přečte hodnotu UZ senzoru
na portu 1 a hodnotu zapíše do x
```

Hodnoty ze senzorů můžeme vymazat pomocí funkce ClearSensor, ta má smysl jen u senzorů, které inkrementují hodnoty.

```
ClearSensor(S2);      //vymaže nainkrementovanou
hodnotu senzoru na portu 2
```

3.16 Typy a módy senzorů

Základní NXT obsahuje senzor světelný, dotykový, ultrazvukový, zvukový a senzory otáček v motorech. Kromě ultrazvukového senzoru jsou všechny analogové, to znamená, že informaci z nich přijatou můžeme zpracovat několika způsoby. Každý z těchto analogových senzorů může pracovat v různých módech, pro různou aplikaci se hodí jiný mód. Ultrazvukový senzor je digitální, obsahuje již obvody pro zpracování

a odesílání dat. Při použití senzoru je nutná nejprve jeho identifikace pomocí fukce *SetSensorType*:

```
SetSensorType(proměnná portu, typ senzoru);
```

kde port je výše uvedený *IN_1* až *IN_4* a typ senzoru udává, jaký senzor je použit:

Senzor	Klíčové slovo
Dotykový	SENSOR_TYPE_TOUCH
Optický s aktivním osvětlováním	SENSOR_TYPE_LIGHT_ACTIVE
Optický bez aktivního osvětlování	SENSOR_TYPE_LIGHT_INACTIVE
Zvukový (dB)	SENSOR_TYPE_SOUND_DB
Zvukový (dBA)	SENSOR_TYPE_SOUD_DBA
Ultrazvukový	SENSOR_TYPE_LOWSPEED_9V

Tabulka 3.3: Typy senzorů

Pomocí funkce *SetSensorMode* poté můžeme nastavit chování senzoru.

```
SetSensorMode(proměnná portu, mód);
```

Některé módy mají smysl jen pro určitý senzor.

Klíčové slovo módu	Výstup senzoru
SENSOR_MODE_RAW	Hodnota od 0 do 1023
SENSOR_MODE_BOOL	Sepnuto ano/ne
SENSOR_MODE_EDGE	Počet sepnutí
SENSOR_MODE_PULSE	Počet sepnutí a rozepnutí
SENSOR_MODE_PERCENT	Hodnota od 0 do 100

Tabulka 3.4: Módy senzorů

Výstupem RAW módu je hodnota mezi 0 a 1023. Význam této hodnoty se liší u každého senzoru. U dotykového senzoru znamenají

hodnoty kolem 1000 rozepnuté tlačítko, hodnoty pod 50 sepnuté[1]. U světelného senzoru se hodnoty pohybují v rozmezí 300 (velmi světlá) až po 800 (velmi tmavá). BOOL mód funguje nad RAW módem - hodnoty pod 562 jsou *true* a hodnoty nad jsou *false*. BOOL mód je výchozím módem dotykového senzoru, ale může být použit i u jiných senzorů. PERCENT mód je výchozí pro světelný a zvukový senzor a funguje nad RAW módem, ze kterého převádí hodnotu na procentuální vyjádření.

Takováto práce se senzory je často zdlouhavá a nepotřebná, proto v NXC existují samostatné funkce pro každý senzor, které ho automaticky nastaví do výchozího módu. Tyto funkce spolu s příklady si ukážeme dále.

3.17 Dotykový senzor

Ukážeme si několik příkladů s použitím dotykového senzoru. V následujícím příkladu pojede robot rovně, dokud není sepnut senzor. Senzor pracuje v BOOL režimu, který je výchozí, proto ho není potřeba nastavovat.

```
task main(){
      SetSensor(IN_1,SENSOR_TYPE_TOUCH);
      //inicializace senzoru
      OnFwd(OUT_AB,75);      //pohyb vpřed
      until(SENSOR_1==1);          //dokud není
splněna podmínka
      Off(OUT_AB);     //vypnutí motoru
}
```

Funkce *SetSensor(IN_1, SENSOR_TYPE_TOUCH)* se dá nahradit funkcí *SetSensorTouch(IN_1)*, která se chová totožně. Podobná syntaxe funguje i u ostatních senzorů. V následujícím příkladu program vylepšíme, robot po nárazu kousek couvne, otočí se a pojede dál.

```
task main(){
```

1 Tyto hodnoty jsou individuální a liší se senzor od senzoru. V žádném případě nelze měřit míru stisku tlačítka, pouze zda je sepnuté či nikoliv.

```
      SetSensorTouch(IN_1); //inicializace senzoru
      OnFwd(OUT_AB,75);     //robot jede dopředu,
dokud nenastane podmínka v if
      while(true){
            if(SENSOR_1==1){//senzor je sepnut
                  OnRev(OUT_AB,75);     //couvnutí
                  Wait(300);
                  OnRev(OUT_A,75);//zatočení
                  Wait(800);
                  OnFwd(OUT_AC);        //robot
pokračuje rovně dokud není opětovně splněna
podmínka v if
            }
      }
}
```

3.18 Světelný senzor

Světelný senzor může pracovat ve dvou režimech - s aktivním svícením LED diodou, kdy snímá množství odraženého světla, nebo v režimu bez přisvětlování, kdy snímá množství okolního světla. První režim využijeme např. při oblíbeném sledování čáry. Princip je jednoduchý, robot se snaží udržet svou pozici na rozhraní barev.

```
task main(){
      SetSensorLight(IN_2); //nastavení senzoru
      int intenzita=40;     //referenční hodnota, na
které se robot snaží udržet
      OnFwd(OUT_AB,75);
      while(true){
            if(SENSOR_2 > intenzita){
                  OnRev(OUT_B,75);
                  until(SENSOR_2 <= intenzita);
                  OnFwd(OUT_AB);
            }
      }
}
```

Robot sleduje rozhraní barev a pokud je hodnota odraženého světla větší než 40, couvne jedním motorem do té doby, než je robot opět na trati. Program má jednu nevýhodu, umožňuje pohyb pouze na jednu stranu.

Pokud by se čára klikatila, je nutný složitější algoritmus, který je rozebrán v kapitole 4.3.8. Při použití senzoru je nejprve nutné provést kalibrační měření. Výstupní hodnota záleží na odrazivosti materiálu, okolním osvětlení a vzdálenosti čidla od snímaného objektu. Hodnoty z uniformní bílé plochy tak mohou nabývat hodnot 45%, při oddálení senzoru o 1 cm od plochy už ale jen 39%, přechod černá/bílá se může jevit jako hodnota 42%, resp. 35%. Jednoduchý kalibrační program nám umožní zjistit tyto hodnoty tak, že dané intenzity vypíše na displej robota. Měření je vhodné několikrát opakovat, aby se snížila nejistota měření.

```
task main(){
      SetSensorLight(IN_1);
      int intenzita;
      while(true){
            intenzita=SENSOR_1;
            NumOut(0,0,intenzita);
            Wait(3000);
      }
}
```

Samotná kostka obsahuje v nabídce View → Reflected Light stejný měřící postup.

3.19 Zvukový senzor

Zvukový senzor se svým chováním podobá světelnému senzoru. V následujícím příkladu se robot po hlasitém zvuku rozjede a při dalším zvuku zastaví.

```
task main(){
      SetSensorSound(IN_3); //nastavení senzoru
      while(true){
            until(SENSOR_3 > 40); //nic se neděje
dokud nepřijde zvuk
            OnFwd(OUT_AB,75);     //robot jede
            Wait(300);
            until(SENSOR_3 >40);  //dokud nepřijde
```

```
        další zvukové znamení
                Off(OUT_AB);     //vypnutí motoru
                Wait(300);
        }
}
```

Příkazy *Wait* jsou v kódu nutné, NXT totiž vyhodnotí příkazy tak rychle, že první zvuk by měl vliv i na druhé *until*, takže by se robot rozjel a hned zastavil.

3.20 Ultrazvukový senzor

Jak už bylo výše uvedeno, ultrazvukový senzor je digitální. Senzor umožňuje měřit vzdálenost v rozmezí přibližně 3 cm až 2,5 m s přesností ±3 cm od překážky. V následujícím příkladě se robot bude snažit vyhnout překážkám. Pokud k nějaké dojede na vzdálenost 15 cm, tak se otočí a pojede dál, dokud nenarazí na další překážku.

```
task main(){
      SetSensorLowspeed(IN_4);   //inicializace
senzoru
      int VZDALENOST=15;    //vzdálenost v cm
      while(true){
            OnFwd(OUT_AB,75);
            if(SensorUS(IN_4) < VZDALENOST){
      //načtení hodnoty senzoru, je-li vzdálenost
menší než 15cm, pak robot zatočí
                  Off(OUT_AB);
                  OnRev(OUT_B,75);
                  Wait(1000);
            }
      }
}
```

3.21 LCD displej

LCD displej na NXT je bodový s rozlišením 100x64 pixelů. Pomocí funkcí v NXC na něj můžeme nechat vykreslit text, čísla, jednoduchou

grafiku v podobě čar a oblouků až po složitější bitmapovou grafiku. Výpis hodnot proměnné se hodí v případě ladění programu. Můžeme si tak např. nechat vypisovat aktuální vzdálenost z ultrazvukového senzoru a tím si ověřovat jeho přesnost. Při použití jakékoliv kreslící funkce se zadávají souřadnice, kam má být text či grafika umístěna. Pro usnadnění výpisu textu je displej rozdělen na osm textových řádků, označených *LCD_LINE1* až *LCD_LINE8*. Tyto konstanty použijeme místo *y* souřadnice. Souřadnice (0,0) odpovídají levému spodnímu rohu displeje.

```
ClearScreen();  //vymaže displej
NumOut(x,y,jméno proměnné);     //vypíše hodnotu v proměnné na souřadnice x,y
TextOut(x,y,text);    //obdoba NumOut pro výpis textu
LineOut(x1,y1,x2,y2); //úsečka z bodu (x1,x2) do bodu (y1,y2)
CircleOut(x,y,poloměr);    //kružnice se středem v (x,y) a poloměrem
PointOut(x,y);        //bod na souřadnicích (x,y)
ResetScreen();        //resetuje displej do výchozího stavu
```

3.22 Komunikace mezi roboty

NXT umožňuje komunikaci mezi roboty pomocí bluetooth. Hlavní kostka, označovaná jako master, může ovládat až tři další kostky označované jako slave, na kanálech 1 až 3, master má kanál č.0. Pro zprovoznění bluetooth komunikace je nutné připojit všechny kostky k počítači s aktivovaným bluetooth menu.

Master jednotka může ovládat ostatní pomocí přímých příkazů, slave jednotky tak nemusí mít nahraný žádný program. Komunikace je značně krkolomá, pro běh motoru je zde pouze jeden příkaz, zahrnující všechny možnosti. Nastavit lze i senzory, pro vyhodnocení jejich výstupů už musí slave jednotka obsahovat i kód pro příjímání a posílání čísel.
V následujícím příkladu je motor slave robota uveden do chodu pomocí

master kostky:

```
#define BT_CONN 1
#define MOTOR(p,s) RemoteSetOutputState(BT_CONN, p,
s,
OUT_MODE_MOTORON+OUT_MODE_BREAK+OUT_MODE_REGULATED,
OUT_REGMODE_SPEED, 0 OUT_RUNSTATE_RUNNING, 0)

sub BTCheck(int conn){
      if(!BluetoothStatus(conn)==NO_ERR){
            TextOut(0,0,“Bluetooth error“);
            Wait(3000);
            Stop(true);
      }
}

task main(){
      BTCheck(BT_CONN);
      MOTOR(OUT_A,100);
      Wait(1000);
      MOTOR(OUT_A,0);
}
```

Na začátku programu definujeme makro, protože příkaz na spuštění motoru je značně komplikovaný a pokaždé ho v kódu kopírovat přinese jen chaos. U příkazu *RemoteSetOutputState* nás zajímají hlavně první tři parametry: adresa slave jednotky, port motoru a jeho tah, které pak nastavujeme v programu individuálně. Procedura *BTCheck* kontroluje stav spojení, v případě jeho přerušení ukončí program.

4 NÁVRH SOUTĚŽNÍCH ÚLOH

4.1 Úvod do čtvrté kapitoly

Soutěžní úlohy, jak jsou zde popsány, budou řešit studenti ve skupinách v rámci předmětu Roboti. Seznámí se zde s cílem úlohy, pravidly a návrhem řešení pomocí programovacího jazyka NXC.

4.2 Úloha č.1: Sumo

4.2.1 Cíl úlohy

Cílem úlohy je sestavit a naprogramovat robota tak, aby samostatně vytlačil soupeřova robota z kruhu za hraniční čáru a přitom v kruhu sám zůstal.

4.2.2 Vybavení pro řešení úlohy

Každému studentskému týmu zapůjčí Katedra řídicí techniky jednu soupravu LEGO Mindstorms Education (9797), doplňující soupravu dílů (9648) a síťový adaptér (9833), které v kompletním a neporušeném stavu vrátí po skončení soutěže. NXT kostka bude při odevzdání obsahovat standardní firmware.

4.2.3 Konstrukce robota

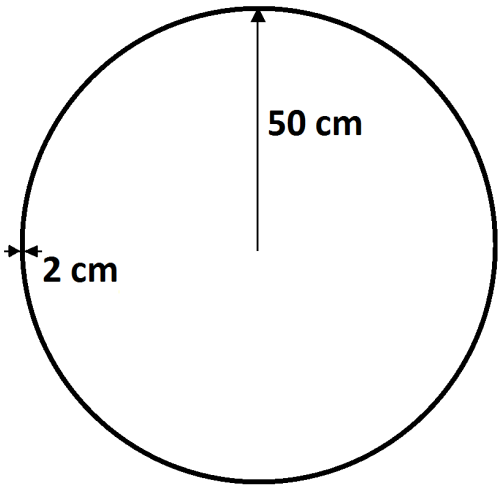
Ke konstrukci robota lze využít pouze dílů z výše uvedených stavebnic s výjimkou vázacích drátů, provázků či gumiček. Legové díly mohou držet pohromadě pouze pomocí standardních spojovacích prvků LEGO, použití lepidel, šroubů a jiných spojovacích materiálů není povoleno. Půdorys robota se musí vejít do čtverce o straně 25 cm, výškové omezení není. Po zahájení zápasu může robot tento limit překročit. Váhové omezení robota je 1 kg.

4.2.4 Programování robota

Program robota může být napsán v jakémkoliv programovacím jazyce. Robot se po zahájení soutěže musí pohybovat samostatně, ovládání robota pomocí hlasu, bluetooth či jiných komunikačních kanálů není dovoleno.

4.2.5 Hrací plocha

Hrací plocha je kruhová s průměrem 1 m. Na okraji je černá čára o šířce 2 cm, zbytek kruhu je bílý.



Obr. 4.1: Soutěžní plán k úloze sumo

4.2.6 Průběh soutěže a další pravidla

Před zahájením soutěže jsou na hrací plochu umístěni dva soupeřící roboti ve vzdálenosti 10 cm od středu hrací plochy tak, aby jejich kola byla vodorovně. Roboti jsou k sobě umístěni zádí. Kde má robot záď určuje dohlížející komise. Příď robota je typicky strana s radlicí, záď obsahuje zatáčecí kolo apod. Po umístění robotů na hrací plochu je majitelé naráz spustí. Robot **musí** čekat 4 vteřiny před tím, než začne vykonávat

jakoukoliv aktivitu.

Jedno kolo soutěže trvá 1 minutu nebo dokud není jeden či oba z robotů vyřazen. K tomu dojde v momentě, kdy je robot za černou čarou hrací plochy všemi hnacími koly či je neschopen dalšího pohybu (je povalen). Po vyřazení robota běží kolo ještě dalších 5 vteřin. Při souboji robotů **nesmí** dojít k poničení vlastního ani cizího robota.

4.2.7 Bodování

Vítězný robot (soupeřův robot je vyřazen a vlastní robot zůstane na hrací ploše) získává 2 body. V případě remízy (oba roboti vyřazeni nebo uplyne časový limit) získávají oba roboti 1 bod. Poražený robot nezískává žádný bod.

4.2.8 Konstrukční řešení úlohy

Konstrukční řešení robota pro sumo hraje podstatnou roli v cestě za vítězstvím. Pří řešení úlohy jsem experimentoval s asi desítkou robotů. První robot, kterého jsem se snažil postavit, se měl chovat podobně jako buldozer. Pomocí radlice vytlačil soupeře z hrací plochy. V některých případech se mu i soupeře podařilo převrátit. Hned v počátku jsem zjistil, že bez převodu rychlosti motoru to nepůjde. V této úloze není tak důležitá rychlost jako právě tah motorů. Nejdříve jsem začal s převodem 3:1, který se ale ukázal nedostatečný. V momentě, kdy protivníkův robot stál na místě, ho můj robot nebyl schopen posunout po podložce. Pouze stál na místě a kola se mu protáčela. Pokračoval jsem s převodem 5:1, který se ukázal být dostatečným pro vytlačení, robot byl díky němu ale už moc pomalý. Rozhodl jsem se proto pro robota, který bude aktivnější v boji se soupeřem.

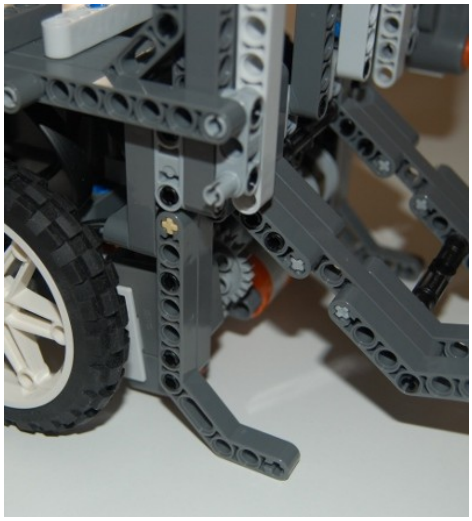
Konstrukční řešení se podobá vysokozdvižnému vozíku. Celkový

pohled na robota je na Obr. 4.2. Konstrukce vychází z robota ze zdroje [19].

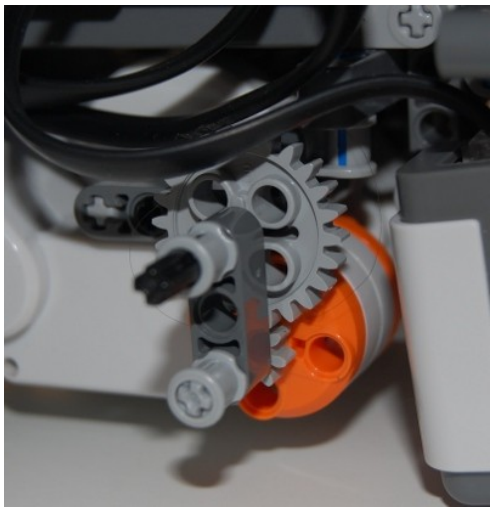
Obr. 4.2: Konstrukční řešení pro úlohu SUMO: vysokozdvižný vozík

Robot využívá dvou motorů s převodem 3:1 k pohybu, viz Obr. 4.4. Třetí motor je také s převodem 3:1 a slouží jako naviják pro lanko, které zvedá vidle, viz Obr. 4.5. Pomocí nich se robot snaží protivníka nadzvednout a odtlačit, ideálně převrátit. Celá konstrukce je co možná nejvíce robusní s velkým množstvím vzpěr a výztuh. Aby se robot při zvedání břemene nepřevážil dopředu, jsou na něm umístěny vzpěry, které se neustále dotýkají země, viz Obr. 4.3. Podobné vzpěry jsou umístěny i na bocích, zde je to však obranný mechanismus.

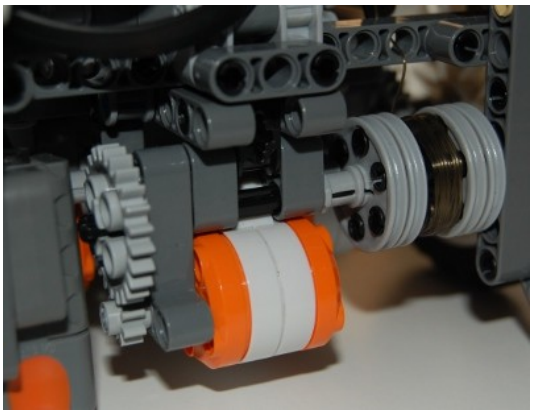
Důležitý je také správný výběr kol, pomoci v tomto ohledu může zdroj [20], ve kterém je proveden test efektivity trakce. Robot využívá kol 81,6x15, které podávají nejlepší výsledky.



Obr. 4.3: Detail vzpěry, která brání převážení robota při zvedání břemene

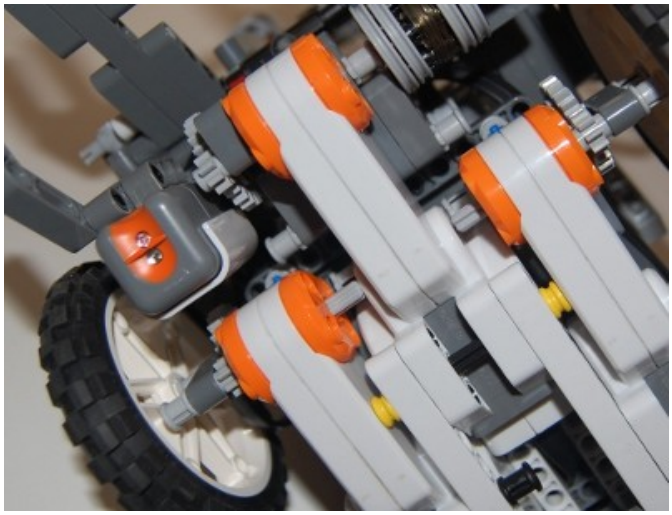


Obr. 4.4: Detail převodu motoru na kolo



Obr. 4.5: Detail navíjecího ústrojí

Robot musí být vybaven světelným senzorem, který by měl být dostatečně předsunut před koly tak, aby byl schopen detekovat hraniční čáru v dostatečném předstihu. Ideálně by měl být umístěn uprostřed robota. Toho se mi bohužel nepodařilo dosáhnout, uprostřed je umístěn třetí motor. Senzor navíc nemůže být moc předsunutý před kola, protože by kolidoval se zvedacím mechanismem. Zvolil jsem proto kompromis, který je vidět na Obr. 4.6. Robot díky tomu má problém v případě, že najede k čáře pod velkým úhlem levým kolem napřed. Senzor pak většinou zastaví robota příliš pozdě. Ultrazvukový senzor je také nepostradatelný, s jeho pomocí robot hledá protivníka. Jeho umístění by mělo být ideálně ve výšce kol a na středu robota, přičemž výškové umístění je opravdu stěžejní. Bez správně umístěného senzoru robot nenajde protivníka. Dále je použit dotykový senzor pro indikaci horního dorazu zvedacího mechanismu. Původně jsem chtěl použít dotykový senzor i pro detekci spodní polohy, ale díky nízké váze ústrojí to nebylo možné, senzor se nikdy nesepnul.



Obr. 4.6: Detail umístění světelného senzoru

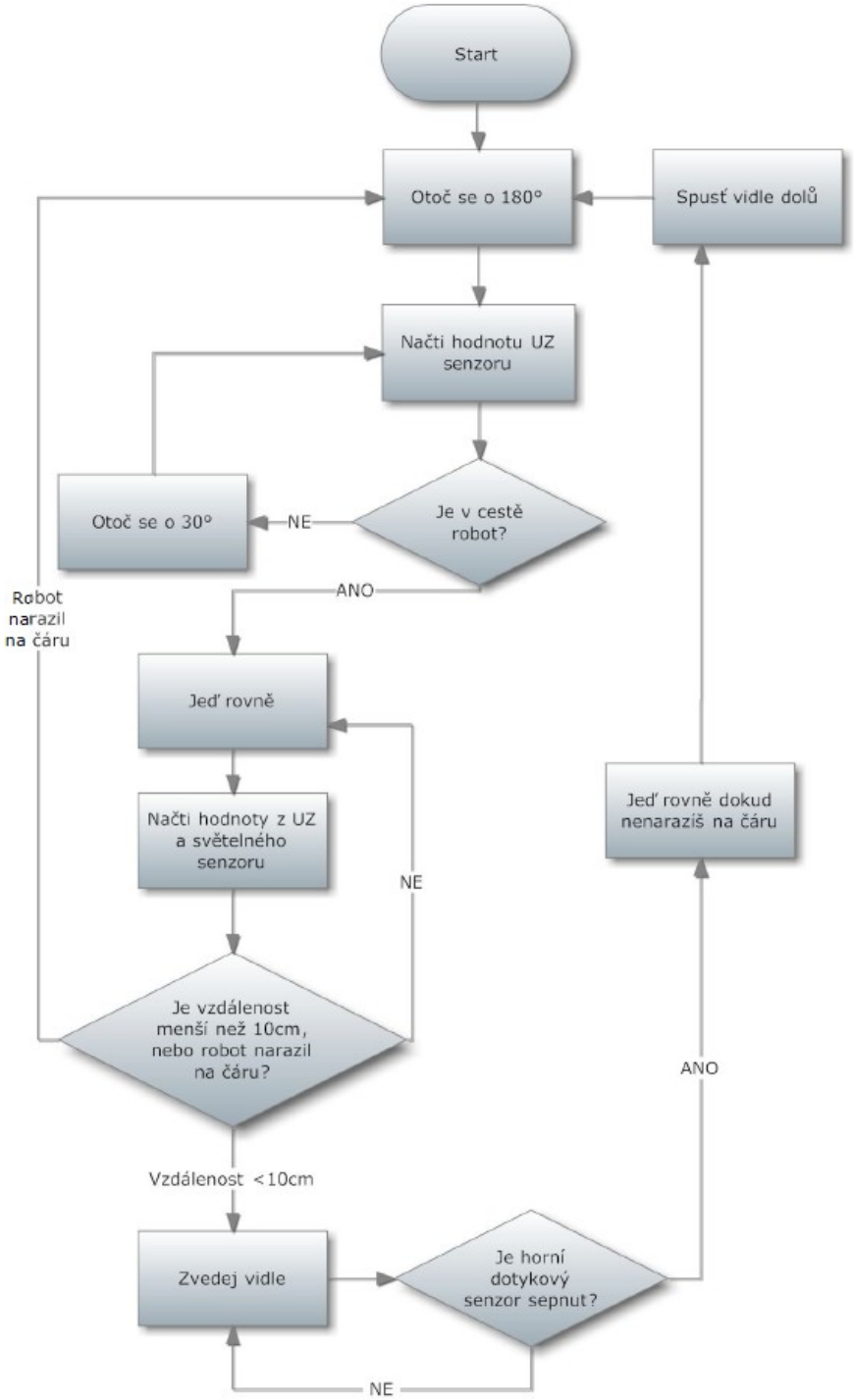
4.2.9 Programové řešení úlohy

Vývojový diagram pro robota typu vysokozdvižný vozík je na Obr. 4.7. Program se skládá ze dvou procedur, jedna pro otočení o 180° a druhá pro otočení o 30°.

```
sub spin180(){
      OnFwd(OUT_A,100);
      OnRev(OUT_B,100);
      Wait(1280);
      Off(OUT_AB);
}
sub spin30(){
      OnFwd(OUT_A,100);
      OnRev(OUT_B,100);
      Wait(300);
      Off(OUT_AB);
}
```

Dále program obsahuje tři úkoly. První ovládá zvedací mechanismus, ve druhém běží samotný algoritmus a ve třetím pouze nastavujeme senzory a spouštíme ostatní úkoly. Z důvodu konstrukce robota je pro pohyb vpřed nutné používat funkci *OnRev* a naopak, je to důsledek převodu, který obrací směr otáčení. Tento fakt se dá obejít několika způsoby. Použitím makra, kdy si vytvoříme vlastní funkci pro pohyb, používáním záporných hodnot tahu motoru, nebo na tuto skutečnost budeme pamatovat a upozorníme na to v komentáři.

```
task lift(){
      if(nahoru==true){
            OnFwd(OUT_C,100);
            until(Sensor(IN_3)<1023);
            Off(OUT_C);
      }else{
            OnRev(OUT_C,100);
            Wait(1800);
            Off(OUT_C);
      }
}
```



Obr. 4.7: Vývojový diagram pro robota "vysokozdvižný vozík"

```
bool nahoru;    //globální proměnná
task run(){
     int dist=40;    //vzdálenost při hledání
soupeře
     int li=62; //intenzita světla
     int ndist=10;   //vzdálenost
     bool otoc=true;
     while(true){
          if(otoc==true){
               spin180();
          }
          if(SensorUS(IN_2)<dist){
               OnRev(OUT_AB,100);
               until(Sensor(IN_1)<li ||
SensorUS(IN_2)<ndist){
                    if(SensorUS(IN_2)<ndist){
                         nahoru=true;
                         StartTask(lift);
                         until(Sensor(IN_1)<li;
                         Off(OUT_AB);
                         OnFwd(OUT_AB,100);
                         Wait(1000);
                         Off(OUT_AB);
                         otoc=true;
                         nahoru=false;
                         StartTask(lift);
                         Wait(2000);
                    }
                    else{
                         OnFwd(OUT_AB,100);
                         Wait(1000);
                         Off(OUT_AB);
                         otoc=true;
                    }
               }
               else{
                    spin30();
                    otoc=false;
                    Wait(500);
               }
```

```
            }
        }
    }

    task main(){
        SetSensorLight(IN_1);
        SetSensorLowSpeed(IN_2);
        SetSensorType(IN_3, SENSOR_TYPE_TOUCH);
        SetSensorMode(IN_3, SENSOR_MODE_RAW);
        StartTask(run);
    }
```

4.2.10 Závěr k úloze sumo

Úlohu „Sumo“ se podařilo úspěšně navrhnout i vyřešit. Videozáznam s průběhem souboje je k vidění na stránce: http://www.youtube.com/watch?v=ZmrDUOK96W4 či http://www.youtube.com/watch?v=4qXAk3znR0w . Úloha je především o konstrukčním řešení robota. Hlavní úskalí je v ultrazvukovém senzoru, který není vhodný pro hledání protivníkova robota, ale spíše pro detekci větších objektů (zdi). Velká část práce spočívá v cyklickém ladění programu. Úlohu jsem naprogramoval pouze v jazyku NXC, protože jazyk NXT-G není vhodný pro programování složitějších úloh a jazyk NXJ se od jazyka NXC liší svými možnostmi natolik, že by to vyžadovalo úplně jiný přístup.

4.3 Úloha č. 2: Sledování čáry a vyhýbání se objektu

4.3.1 Cíl úlohy

Cílem úlohy je dojet z jednoho konce čáry na druhý v co nejkratším čase, aniž by robot povalil překážku na trati.

4.3.2 Vybavení pro řešení úlohy

Každému studentskému týmu zapůjčí Katedra řídicí techniky jednu

soupravu LEGO Mindstorms Education (9797), doplňující soupravu dílů (9648) a síťový adaptér (9833), které v kompletním a neporušeném stavu vrátí po skončení soutěže. NXT kostka bude při odevzdání obsahovat standardní firmware.

4.3.3 Konstrukce robota

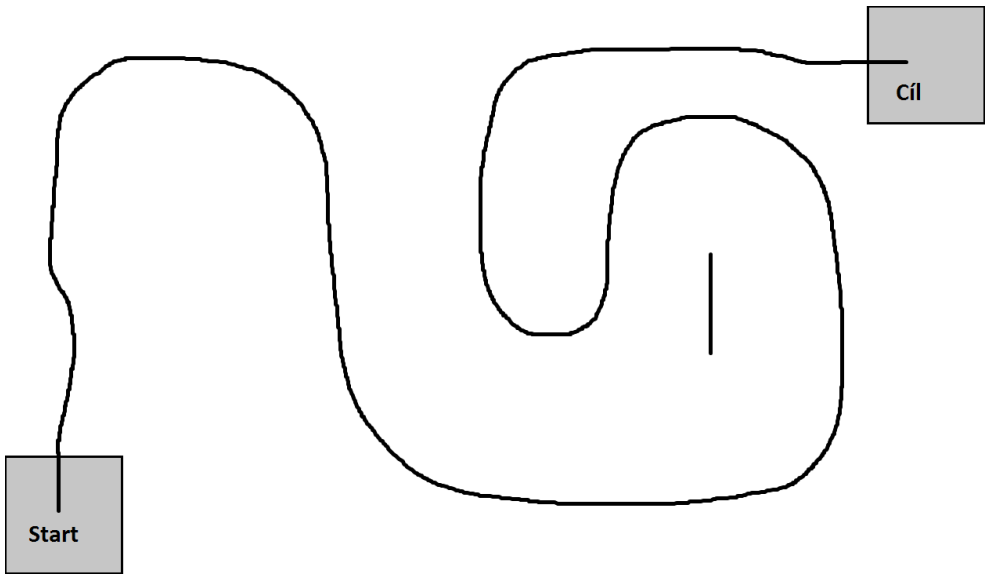
Ke konstrukci robota lze využít pouze dílů z výše uvedených stavebnic. Legové díly mohou držet pohromadě pouze pomocí standardních spojovacích prvků LEGO, použití lepidel, šroubů a jiných spojovacích materiálů není povoleno.

4.3.4 Programování robota

Program robota může být napsán v jakémkoliv programovacím jazyce. Robot se po zahájení soutěže musí pohybovat samostatně, ovládání robota pomocí hlasu, bluetooth či jiných komunikačních kanálů není dovoleno.

4.3.5 Uspořádání soutěžního plánu

Soutěžní plán obsahuje dvě ohraničená pole označená START a CÍL, která jsou spojena černou čárou minimální tloušťky 2cm. Poloměr zatáček je minimálně 20cm.



Obr. 4.8: Soutěžní plán k úloze sledování čáry a vyhýbání se objektu - jedno z možných uspořádání

4.3.6 Průběh soutěže a pravidla

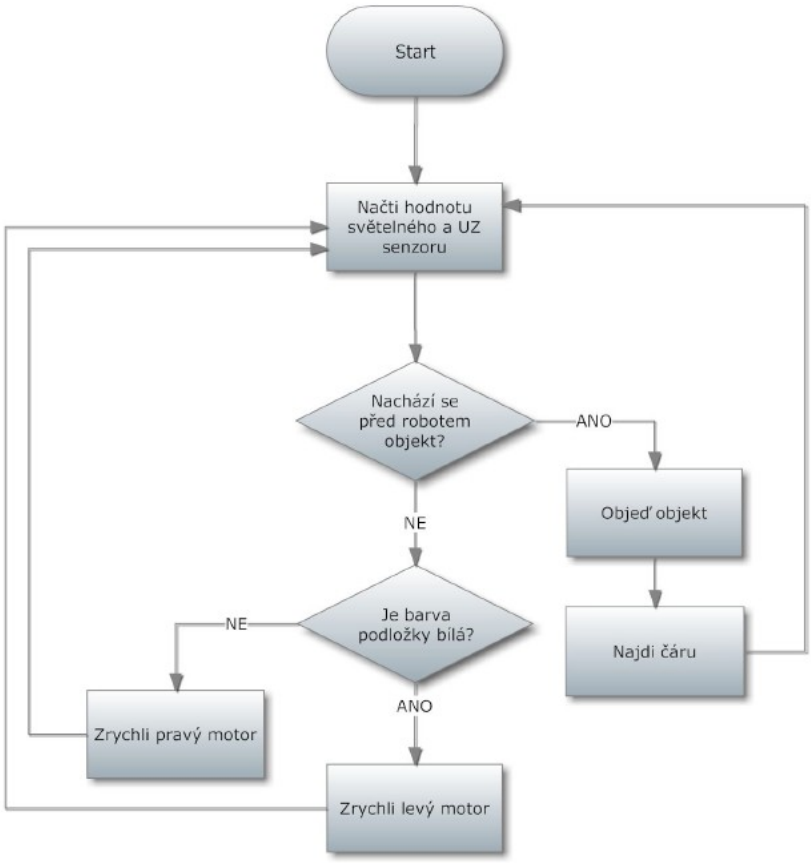
Robot může být umístěn kamkoliv na čáru ve startovním poli. Poté musí samostatně dojet do cílového pole. Na jakémkoliv místě dráhy je umístěna alespoň jedna překážka, kterou robot musí objet tak, aniž by jí převrátil.

4.3.7 Konstrukční řešení úlohy

Konstrukční řešení v této úloze hraje oproti sumo robotovi nepodstatnou roli. Pro vyřešení úlohy postačí jednoduchý robot se světelným a ultrazvukovým senzorem. Světelný senzor je vhodné umístit do osy otáčení robota, prakticky to znamená mezi hnací kola. Ultrazvukový senzor je umístěn na třetím motoru tak, aby se mohl volně otáčet kolem dokola, navíc by měl být dostatečně nízko, aby byl schopný zaregistrovat překážku.

4.3.8 Programové řešení úlohy

Vývojový diagram na Obr. 4.9 zobrazuje algoritmus při řešení úlohy.



Obr. 4.9: Vývojový diagram pro robota sledující čáru

Algoritmus sledování čáry je jednoduchý. Načteme hodnotu ze světelného senzoru, který pracuje v režimu RAW, kvůli větší přesnosti. Experimentálně zjistíme, jaké hodnoty senzor vrací. Obvykle se barevný přechod jeví jako hodnota 500. Tuto hodnotu budeme považovat za referenci pro náš regulátor. Nejjednodušší regulátor typu P má tvar:

$$u(t) = k_p \cdot e(t)$$

Směrodatnou odchylku e(t) vypočítáme jako rozdíl mezi aktuální hodnotou senzoru a referencí:

$$u(t) = k_p \cdot (y - r)$$

Pokusíme-li se tento regulátor implementovat, zjistíme, že robot

nepříjemně kmitá. To odstraníme přidáním derivační složky. Vytvoříme tím PD regulátor, který bude mít tvar:

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

Konstanty k_p a k_d musíme určit experimentálně. Pokud potřebujeme hodnoty k_p či k_d menší jak 1, pak nezbývá než vydělit akční zásah, jak je uvedeno v příkladě. Hodnotu akčního zásahu je také vhodné omezit pomocí saturace tak, aby generované hodnoty posílané do motorů byly v rozmezí ± 100. Kód v NXC pak vypadá takto[2]:

```
sub FollowLinePD(){
      int ref=490;     //reference světelného
senzoru
      int kp=1;  //konstanta proporcionální složky
      int kd=1;  //konstanta derivační složky
      int sVal;  //proměnná pro hodnoty senzoru
      int sValOld=490;//proměnná pro hodnotu
senzoru v předchozím průchodu
      int u;
      while(SensorUS(IN_2)>7){   //procedura běží,
dokud US senzor nenajde objekt blíž než 7 cm
            sVal=Sensor(IN_1);    //načtení hodnoty
senzoru
            u=(kp*(sVal-ref))/9+kd*((sVal-ref)-
(sValOld-ref)); //výpočet akčního zásahu
            if(u>60){  //omezení akčního zásahu
                  u=60;
            }
            if(u<-60){
                  u=-60;
            }
            OnFwd(OUT_A,40+u);
            OnFwd(OUT_B,40-u);
            sValOld=sVal;
      }
      Off(OUT_AB);
}
```

2 Popsané programové řešení je pouze orientační. Hodnoty konstant regulátorů, čekací časy a další proměnné jsou silně individuální. Závisí na konstrukci robota, okolních podmínkách a dalších faktorech.

Dojede-li robot k překážce, musí jí objet. K tomu využijeme ultrazvukový senzor umístěný na třetím motoru. Jakmile robot narazí na překážku, otočíme robota i senzor o 90° tak, aby senzor směřoval k objektu. Nyní využijeme další regulátor. Bude typu P a akční zásah budeme generovat z hodnot od ultrazvukového senzoru. Robot se bude snažit stále zůstat od překážky stejně daleko a bude kolem ní kroužit do té doby, než narazí na čáru. Poté se na místě otočí doprava a pojede rovně, dokud nenajde čáru. Pak běží celý algoritmus od začátku.

```
sub circle(){
	int ref=9; //referenční hodnota pro UZ senzor
	int u;
	int k=4;//konstanta proporcionální složky
	while(Sensor(IN_1)>530){   //procedura běží, dokud robot nenarazí na černou čáru
		u=k*(SensorUS(IN_2)-ref);  //výpočet akčního zásahu
		OnFwd(OUT_A,30+u);
		OnFwd(OUT_B,30-u);
	}
	Off(OUT_AB);
}
```

Procedura pro zatočení robota o 90°. Argumentem volíme směr.

```
sub spin90(pwr){
	OnFwd(OUT_A,-pwr);
	OnFwd(OUT_B,pwr);
	Wait(260);
	Off(OUT_AB);
}
```

V úkolu main pak nastavíme senzory a provedeme zbytek algoritmu.

```
task main(){
	SetSensorType(IN_1, IN_TYPE_LIGHT_ACTIVE);
	SetSensorMode(IN_1, IN_MODE_RAW);
	SetSensorLowspeed(IN_2);
	while(true){
		FollowPD();
		OnRev(OUT_AB,40);
```

```
                Wait(300);
                spin90(100);
                RotateMotor(OUT_C,-40,95);
                Wait(100);
                OnFwd(OUT_AB,40);
                Wait(250);
                Off(OUT_AB);
                circle();
                RotateMotor(OUT_C,40,95);//otočení UZ
senzoru o 90° tak, aby směřoval na objekt
                Wait(100);
                spin90(80);
        }
}
```

4.3.9 Závěr k úloze sledování čáry a vyhýbání se objektu

Úlohu se podařilo úspěšně navrhnout i vyřešit. Videozáznam jejího řešení, tak jak je popsáno výše, lze nalézt na adrese: http://www.youtube.com/watch?v=Encs3Yokb7U . Úloha je zajímavá především z hlediska řízení, kdy je pro vyřešení nutno aplikovat nějaký regulátor. Velkou část úlohy pak tvoří cyklická optimalizace zmíněných regulátorů. Vliv konstrukčního řešení se naopak v této úloze projeví zcela minimálně, pro úspěšné zvládnutí úlohy postačí libovolný robot. Úlohu jsem naprogramoval pouze v jazyku NXC, protože jazyk NXT-G není vhodný pro programování složitějších úloh a jazyk NXJ se od jazyka NXC liší svými možnostmi natolik, že by to vyžadovalo úplně jiný přístup.

5 Webové stránky pro soutěžní úlohy

5.1 Úvod do páté kapitoly

Cílem této kapitoly je rozšíření stávajících webových stránek předmětu A3B99RO Roboti o webové prezentace dvou soutěžních úloh navržených v předešlých kapitolách. Webové stránky jsou dostupné na odkazu: http://support.dce.felk.cvut.cz/roboti/

5.2 Obsah a forma stránek

Webová prezentace soutěžních úloh je ve čtyřech souborech:

- content_project_sumo_cs.php
- content_project_sumo_en.php
- content_project_cara_s_objektem_cs.php
- content_project_cara_s_objektem_en.php

ke každé úloze jeden český a jeden anglický ekvivalent.

Obsah stránek je převzat z kapitol 4.2.1 až 4.2.7 a 4.3.1 až 4.3.6, v anglické mutaci je tentýž obsah.

Formátování stránky využívá CSS stylů ze současných stránek a dodržuje vzor již hotových webových prezentací starších úloh.

6 Závěr

Úspěšně se podařilo splnit všechny body zadání bakalářské práce. Práce obohatí čtenáře především třetí kapitolou, které je věnováno nejvíce prostoru a obsahuje veškeré potřebné informace pro programování robota v jazyku NXC. Návod je koncipován pro uživatele všech úrovní. Jak pro ty, kteří se setkávají s programováním prvně, tak pro zkušenější programátory, kteří hledají další možnost v uplatnění jejich nadání. Čtvrtá kapitola je neméně důležitá. Obsahuje návrh, řešení a dokumentaci ke dvěma soutěžním úlohám, které budou řešit studenti předmětu Roboti.
U jednotlivých úloh jsou popsána pravidla, konstrukční i programové řešení. Cílem bylo ukázat studentům jeden z mnoha postupů a předat jim co možná nejvíce zkušeností, které jim mohou pomoci při překonávání překážek.

Seznam použitých zdrojů

Monografie

[1] HANSEN, John. Not eXactly C (NXC) Programmer's Guide, 2007

[2] BENEDETTELLI, Daniele. Programming LEGO NXT Robots using NXC, 2007

[3] ISOGAWA, Yoshihito, LEGO Technic Tora no Maki, 2007

Bakalářské práce

[4] TROJÁNEK, Pavel. Využití robota LEGO MINDSTORMS při výuce, Praha 2009, bakalářská práce (Bc.). České vysoké učení technické v Praze. Fakulta řídicí techniky

Internetové zdroje

[5] Robots Rule! [online], Union Collage (2010-04-12), <http://cs.union.edu/csc104/>

[6] LEGO Engineering [online], Tufts university (2010-04-12), <http://legoengineering.com/>

[7] LEGO Mindstorms NXT [online], ETH University Zurich (2010-03-12), < http://www.tik.ee.ethz.ch/mindstorms/sa_nxt/index.php?page=print>

[8] Roboti [online], České vysoké učení technické v Praze (2010-03-12), < http://support.dce.felk.cvut.cz/roboti>

[9] Roboti NXT Mindstorms [online], Univerzita Tomáše Bati ve Zlíně (2010-03-12), < http://vyuka.fai.utb.cz/course/enrol.php?id=99>

[10] LEGO Mindstorms NXT – Powered by NI LabView [online], National Instruments (2010-03-12), <http://www.ni.com/academic/mindstorms/>

[11] Bricx Command Center [online], (201-03-12), <http://bricxcc.sourceforge.net/>

[12] LeJOS, Java for LEGO Mindstorms [online], (2010-03-12),

<http://lejos.sourceforge.net/>

[13] LEGO Mindstorms NXT software for Matlab & Simulink [online], The MathWorks, (2010-03-12), <http://www.mathworks.com/academia/lego-mindstorms-nxt-software/>

[14] Using the NXT 2.0 Kit to build NXT 1.X projects [online], (2010-03-12), < http://www.nxtprograms.com/help/parts/8547.html>

[15] LEGO Education School [online], LEGO group, (2010-03-12), < http://www.lego.com/education/school/default.asp?locale=2057&pagename=fqnxt&l2id=3_2&l3id=3_2_5&l4id=3_2_5_1#n5>

[16] Atmel products [online], Atmel, (2010-03-12), < http://www.atmel.com/dyn/products/Product_card.asp?part_id=3524>

[17] NXT Motor internals [online], (2010-03-12), < http://www.philohome.com/nxtmotor/nxtmotor.htm>

[18] NXT Sound sensor [online], (2010-03-12), < http://www.convict.lu/htm/rob/NXT_sound_sensor.htm>

[19] NXT Forklift [online], (2010-03-12), < http://www.nxtprograms.com/forklift/index.html>

[20] Wheels, tyres and traction [online], (2010-03-12), < http://www.philohome.com/traction/traction.htm>

Seznam ilustrací

Seznam tabulek

Obsah elektronické přílohy

Elektronická verze dokumentu bakalářské práce

Zdrojové kódy k vyřešeným úlohám

Videa vyřešených úloh

Prezentace programovacího jazyka NXC

Webové prezentace k soutěžním úlohám v českém a anglickém jazyce