ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

Katedra řídicí techniky

Název: Využití robotů LEGO Mindstorms při výuce – návrh soutěžních úloh

Bakalářská práce

Praha 2011

Student: Jakub Mrva

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

Abstrakt:

Tato práce pojednává o možnostech využití robotů LEGO Mindstorms při výuce. Zabývá se současným stavem, softwarovými a hardwarovými možnostmi stavebnice. Dále je součástí práce návrh dvou nových soutěžních úloh, včetně jejich řešení a specifikace pravidel. Závěrečná část této práce je věnována webovým stránkám k navrženým úlohám.

Abstract:

This thesis discusses the possibilities of using LEGO Mindstorms robots as an education tool. It considers the current state, software and hardware of this building blocks. This thesis also includes the design of two new competitive tasks with their solutions and rules included. The end of this thesis is devoted to the web pages of the designed tasks.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Jakub Mrva**

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

Název tématu: **Využití robota LEGO MINDSTORMS - návrh soutěžních úloh**

Pokyny pro vypracování:

1. Seznamte se s možnostmi robota LEGO Mindstorms (současný stav, HW a SW vybavení).
2. Vyberte vhodné prostředí pro programování robota.
3. Proveďte návrh dvou nových soutěžních úloh – automatické parkování a sledování automobilu (udržování konstantní vzdálenosti) 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 2011/2012

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 12. 10. 2010

Poděkování:

Rád bych touto cestou poděkoval svému bratru Matějovi za jeho bystré nápady a vydatnou pomoc při návrhu robotů a dále Martinu Tomkovi za jeho pomoc při vytváření webových stránek. Oba jste mi moc pomohli.

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 3.6.2011

podpis

1 Obsah

1 Úvod 8
1.1 Cíl práce 8
1.2 Obsah práce 8
2 Seznámení s možnostmi robota LEGO Mindstorms 9
2.1 Historie 9
2.2 Současný stav 9
2.3 Hardwarové vybavení 10
2.3.1 NXT inteligentní kostka 11
2.3.2 Servomotor 12
2.3.3 Dotykový senzor 12
2.3.4 Světelný senzor 13
2.3.5 Ultrazvukový senzor 13
2.3.6 Zvukový senzor 15
2.3.7 Hi-Tech senzory 15
3 Softwarové prostředky k programování robota 16
3.1 NXT-G 16
3.2 NXC 17
3.2.1 Základní příkazy pro obsluhu motorů a senzorů 18
3.3 leJOS – NXJ 20
3.3.1 Instalace prostředí Netbeans 21
3.3.2 Příkazy k ovládání motorů 22
3.3.3 Příkazy k ovládání senzorů 23
3.3.4 Pokročilejší třídy k ovládání robota 23
4 Návrh soutěžních úloh 24
4.1 Úvod k soutěžním úlohám 24
4.1.1 Typ podvozku 24
4.1.2 Schopnosti robotů 25
4.2 Výběr vhodného programovacího prostředí 25
4.2.1 Porovnání možností jednotlivých prostředí 25
4.2.2 Závěr porovnání 26
4.3 Úloha: Automatické parkování 26
4.3.1 Cíl úlohy 26
4.3.2 Technické specifikace 26

4.3.3 Pravidla 28
4.3.4 Konstrukční řešení robota 29
4.3.5 Programové řešení robota 33
4.3.6 Výsledek 40
4.4 Úloha: Sledování automobilu 41
4.4.1 Cíl úlohy 41
4.4.2 Technické specifikace 41
4.4.3 Pravidla 42
4.4.4 Konstrukční řešení sledovaného automobilu 44
4.4.5 Konstrukční řešení robota 45
4.4.6 Programové řešení robota 46
4.4.7 Výsledek 48
5 Webové stránky 48
6 Závěr 49
7 Literatura 50
8 Seznam obrázků 51
9 Seznam tabulek 52
10 Příloha 52

1 Úvod

1.1 Cíl práce

Cílem této bakalářské práce je seznámit se s možnostmi stavebnice LEGO Mindstorms, jak jejím hardwarovým, tak i softwarovým vybavením a možnostmi jeho programování. Dále je cílem vytvořit prototypy dvou nových soutěžních úloh určených pro předmět A3B99RO, ve kterém studenti prvního ročníku FEL ČVUT poznávají krásy jejich studijního oboru Kybernetika a Robotika. Poslední částí je vytvoření webových stránek k zadaným úlohám, které budou umístěny na webu katedry řízení.

Ze všech nabízených témat bakalářských prací jsem si vybral právě tuto, protože v sobě skloubí všechny oblasti mnou studovaného oboru Kybernetika a měření. Zároveň mám ke stavenici LEGO citový vztah z dětství a této nabídce nešlo odolat.

1.2 Obsah práce

Tato práce je kromě úvodu a závěru rozdělena do čtyř kapitol. Druhá kapitola poskytuje úvod do problematiky a seznamuje se stavebnicí. Třetí kapitola se věnuje možnostem jejího proramování. Čtvrtá a nejobsáhlejší kapitola obsahuje návrh a řešení dvou nových soutěžních úloh pro předmět Roboti. V páté kapitole se pak zabývá tvorbou webových stránek k navrženým úlohám.

2 Seznámení s možnostmi robota LEGO Mindstorms

2.1 Historie

Prvopočátky této stavebnice se datují do roku 1998, kdy spatřila světlo světa první verze s názvem Robotics Invention Systém (RIS), jehož hlavní částí byla „kostka“ nazvaná Robotics Command eXplorer (RCX).

V roce 2006 vyšla první verze LEGO Mindstorms NXT a o tři roky později, v srpnu roku 2009 novější verze LEGO Mindstorms NXT 2.0, která obohacuje předchozí například o barevný senzor a nové grafické programovací prostředí.

Obrázek 2.1 LEGO Mindstorms RCX

Obrázek 2.2 LEGO Mindstorms NXT

2.2 Současný stav

V současnosti se stavebnice LEGO Mindstorms těší velkému úspěchu. Je velmi oblíbená napříč generacemi. Spousta tatínků kupuje tuto stavebnici svým dětem, protože po ní touží oni sami.

Zároveň je to i skvělý výukový prostředek například do škol. A to nejen základních a středních. Mnoho vysokých škol a univerzit po celém světě používá tuto stavebnici pro výukové účely v motivačních předmětech pro zvýšení zájmu studentů. Ani České vysoké učení technické v Praze není výjimkou.

Její krása spočívá v jednoduchosti. Jak uvádí výrobce [1], funkčního robota si může každý postavit již za zhruba 30 minut. A to není nijak přehnané. K dispozici jsou přehledné návody (jako např. [2]), grafické vývojové prostředí NXT-G a i například volně dostupný softwarový simulátor stavebnice LEGO Digital Designer, kde si může každý zkusit postavit vlastního robota s bezednou zásobou dílů, případně zjistit, jaké díly na daného robota bude potřebovat.

S pomocí této stavebnice je pořádána také spousta nejrůznějších robotických soutěží. Většinou se jedná o úlohu robota, sledující černou čáru s možnými modifikacemi, jako je např. zastavení na konci čáry nebo objetí překážky na trati, nebo o úlohu projetí robota bludištěm či robotí zápasy sumo.

Všechny tyto různé soutěže a motivační předměty slouží k rozvíjení technických poznatků a formou „školy hrou“ úspěšně budí zájem mezi studenty i zvědavou veřejností.

V rámci desátého výročí stavebnice LEGO Mindstorms byla na přání jejích mnohých majitelů vydána limitovaná edice NXT inteligentní kostky v černé barvě (viz obrázek 2.3).

Obrázek 2.1 NXT inteligentní kostka v černém provedení (limitovaná edice)

2.3 Hardwarové vybavení

Kromě stavebních dílů, stavebnice LEGO Mindstorms nabízí i širokou škálu nejrůznějších senzorů a servomotory. Součástí stavebnice LEGO Mindstorms Education Set určené pro výukové účely a použité v této bakalářské práci jsou:

- NXT inteligentní kostku
- Tři servomotory se zabudovanými optickými inkrementálními snímači
- Dva dotykové senzory
- Světelný senzor
- Ultrazvukový senzor
- Zvukový senzor
- Spojovací kabely typu RJ-12 k propojení senzorů a motorů s NXT kostkou

2.3.1 NXT inteligentní kostka

Základem celé stavebnice je řídicí jednotka (na obrázku 2.2 a 2.3), které se říká *NXT inteligentní kostka*, nebo jen zkráceně *kostka.*

Obsahuje 32-bitový mikroprocesor Atmel ARM7 běžící na frekvenci 48MHz s 256kB flash paměti a 64kB RAM.

K ovládání motorů (a diod) slouží tři výstupní porty označené A, B a C.

Senzory se dají připojit na čtyři vstupní porty označené 1, 2, 3 a 4, které umožňují jak analogový, tak digitální přenos signálu.

Propojení s počítačem zajišťuje kabelové připojení přes port USB 2.0 nebo bezdrátové spojení pomocí Bluetooth. Pomocí Bluetooth lze také komunikovat s ostatními NXT inteligentními kostkami. Vzhledem k tomu, že přenos pomocí USB je oproti Bluetooth rychlejší, spolehlivější a bez nutnosti cokoli nastavovat, vřele doporučuji používat výhradně USB např. pro nahrávání programu do NXT kostky.

Kostka komunikuje s uživatelem přes maticový LCD displej o rozměrech 100 x 64 pixelů a čtyři tlačítka.

NXT kostka je napájena buď šesti AA bateriemi nebo dobíjecím Li-ion akumulátorem. Při stavbě robota je dobré myslet na to, že kostka s akumulátorem má větší rozměry, než kostka s bateriemi.

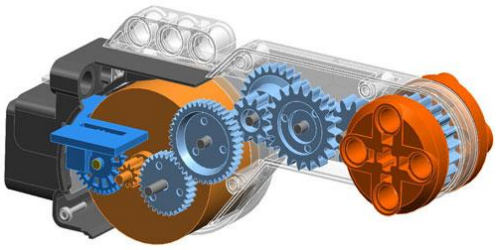
Obrázek 2.2 Li-ion akumulátor

2.3.2 Servomotor

LEGO Mindstorms používá stejnosměrné servomotory napájené až 9V (záleží na napětí baterií). Samotný servomotor je pro zajištění pomalejších pohybů zpřevodován. Při plně nabitých bateriích a bez zátěže je schopen vyvinout na připojené koncové hřídeli otáčivou rychlost až 900 stupňů za vteřinu.

Zároveň obsahuje optický inkrementální snímač s rozlišením 360 dílů na jedno otočení. Díky tomu lze řídit motor i na polohu a relativně přesně měřit a řídit jeho rychlost.

Obrázek 2.5 NXT Servomotor



Obrázek 2.6 Vnitřek motoru

2.3.3 Dotykový senzor

Nejzákladnějším z analogových senzorů je senzor dotykový. Funguje na principu sepnutí a rozepnutí elektrického obvodu. Jeho výstupem je sice hodnota od 0 do 1023, ale vzhledem k principu funkčnosti tohoto senzoru jej nelze použít jako tlakový senzor. Už první zapouzdření tohoto senzoru funkcí mění výstupní hodnotu mezi 0 a 1 při překročení prahové hranice 600 (hodnota menší než 600 odpovídá stisknutému senzoru, zatímco hodnota větší než 600 senzoru nestisknutému).

Tento senzor se dá využít pro detekci nárazu na překážku, nebo třeba při ladění programu jako externí tlačítko, pokud čtyři tlačítka na NXT kostce nestačí, nebo se nedají dobře mačkat, když je robot v pohybu.

Obrázek 2.3 Dotykový senzor

2.3.4 Světelný senzor

Tento senzor obsahuje fototranzistor, který měří intenzitu světla a přisvětlovací diodu. Senzor může fungovat ve dvou režimech. Když je přisvětlovací dioda vypnutá, senzor měří intenzitu okolního světla, což se dá využít například pro ovládání robota posvícením baterkou. V druhém režimu je přisvětlovací dioda zapnutá a robot tak může snímat i odražené světlo této diody od nějakého povrchu. V závislosti na typu a barvě povrchu pak senzor udává intenzitu odraženého světla.

Výstupem je hodnota mezi 0 a 1023, přičemž při vypnuté, resp. zapnuté přisvětlovací diodě odpovídají nízké hodnoty tmě, resp. černé barvě a vysoké hodnoty naopak intenzivnímu světlu, resp. bílé barvě.

Obrázek 2.4 Světelný senzor

2.3.5 Ultrazvukový senzor

Ultrazvukový senzor je jedním z digitálních senzorů. Je natolik složitý, že v sobě ukrývá vlastní obvod s mikroprocesorem, který řídí vysílání a příjem jednotlivých vln ultrazvuku. Propojení s NXT kostkou je zajištěno pomocí I^2C protokolu.

Senzor funguje na principu echolokace, kdy vyšle z vysílače ultrazvukovou vlnu a přijímač čeká na odražené vlny.

Senzor je schopen vyhodnotit až osm odražených signálů, ale kromě první získané vzdálenosti (první přijatý signál) mají další získané vzdálenosti jen velmi malou vypovídací hodnotu. Při detekci blízkého předmětu se většinou se jedná o vícenásobné odrazy od samotného senzoru a nelze tak případně určit, zda je před robotem jedna velká rovná deska, či více malých objektů.

Další funkcí, které tento senzor nabízí, je detekce ultrazvukových vln při vypnutém vysílači. Toho se dá využít v případě více ultrazvukových senzorů v místnosti pro vhodné zvolení vysílacích intervalů k zabránění interference.

Dosah tohoto senzoru je až 250 cm +/- 3 cm. Jeho přesnost již však značně závisí na velikosti a kolmosti odrazového povrchu. Od šikmých ploch se totiž vlny neodrážejí přímo zpět, ale pod úhlem stejným, jako je úhel dopadu. Tím robot získává schopnost vidět „za roh“, ale využitelnost této schopnosti je prakticky nulová. Rovněž malé předměty senzor (např. noha od stolu) není schopen detekovat a o trochu větší předměty, které nejsou úplně rovné, (např. dlaň ruky) detekuje velmi nespolehlivě.

Výstupem tohoto senzoru je většinou (závisí na výše zmíněných režimech) vzdálenost nejbližšího detekovaného předmětu v cm. Pokud však senzor nepřijme žádnou ultrazvukovou vlnu v určitém intervalu, vrací hodnotu 255, což je zároveň maximální výstupní hodnota tohoto senzoru. Toho se dá využít například pro detekci příliš blízkého předmětu (v případě, že nevadí, že se ho senzor přímo dotýká).

Obecně se tento senzor využívá pro detekci náhlé překážky, kdy stačí jediný odraz od blízkého předmětu pro jeho detekci. Pro sledování předmětu nebo prozkoumávání okolí se již musí počítat s jeho nevypočitatelnou nepřesností způsobenou výše zmíněnými odrazy a zhoršenou schopností určit vzdálenost od malého různorodého předmětu.

Obrázek 2.5 Světelný senzor

2.3.6 Zvukový senzor

Tento senzor měří intenzitu okolního hluku v decibelech. Je schopen rozlišit různé úrovně hlasitosti až do 90 dB. Ze všech senzorů je asi nejméně využitelný v nějaké sofistikované aplikaci, protože pro přesnou funkci vyžaduje tiché prostředí.

Lze využít hlavně v jednoduchých aplikacích, kdy robot například reaguje na tlesknutí apod.

Obrázek 2.6 Zvukový senzor

2.3.7 Hi-Tech senzory

Ve verzi LEGO Mindstorms NXT 2.0 se objevuje i barevný senzor, jenž je schopen rozpoznávat jednotlivé barvy, a díky němu lze řešit například populární úlohu skládání Rubikovy kostky.

Samostatně se pak dají zakoupit senzory jako gyroskop, akcelerometr, laserový dálkoměr, infračervený přijímač, teplotní čidlo, tlakový senzor, kompas a další, které jsou teprve ve vývoji.

3 Softwarové prostředky k programování robota

NXT inteligentní kostku lze programovat v nejrůznějších programovacích prostředích. Součástí originální distribuce je programovací prostředí NXT-G, vhodné pro začátečníky. Pak existují další jazyky, které s výhodou vycházejí z dobře známých programovacích jazyků. Jsou jimi jazyky NXC (Not eXactly C), jehož syntaxe je velmi podobná jazyku C, a leJOS NXJ (LEGO Java Operating System Not eXactly Java), který vyžaduje vlastní firmware.

Další, již méně používaný jazyk je například RobotC [12], který vychází z jazyka C a běží na optimalizovaném firmwaru, který umožňuje rychlejší běh aplikací.

Programovat lze také v MATLABu pomocí RWTH – Mindstorms NXT Toolboxu.

Všechny výše zmíněné jazyky poskytují vlastní vývojové prostředí (přičemž leJOS NXJ formou pluginů do známých prostředí Netbeans a Eclipse) a podporují vícevláknové aplikace a Bluetooth komunikaci.

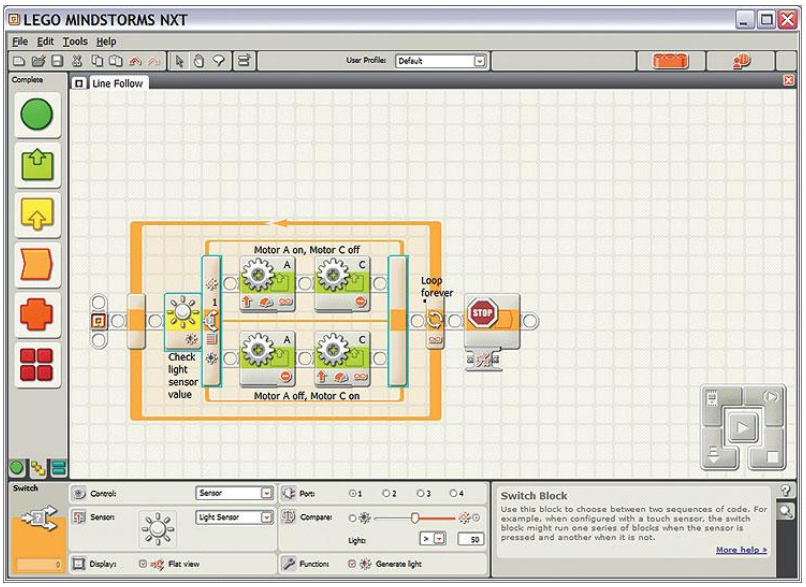
Podrobněji rozebereme pouze první tři zmíněná programovací prostředí.

3.1 NXT-G

Jak již bylo řečeno, NXT-G je grafické vývojové prostředí poskytované výrobcem. Je založeno na prostředí LabView a umožňuje vizuální programování metodou drag-and-drop („táhni a pusť“). Je velmi výhodné zvláště pro mladší generace, protože nevyžaduje znalosti programovacího jazyka. Spojováním bloků a nastavováním parametrů lze vytvořit jednoduchý algoritmus.

Bohužel pro náročnější aplikace je toto programovací prostředí naprosto nevhodné. Znesnadňuje práci s proměnnými a poli, nepodporuje čísla v plovoucí řádové čárce a psaní jakéhokoli složitějšího programu je časově o dost náročnější než v jiných prostředích.

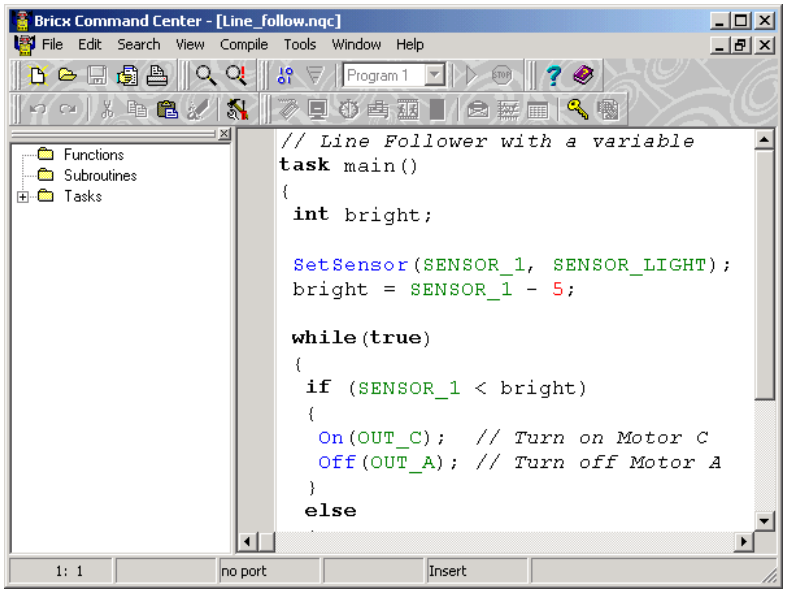
Ukázka tohoto grafického prostředí je na obrázku 3.1.



Obrázek 3.1 Grafické prostředí NXT-G

3.2 NXC

Programovat v jazyce NXC lze celkem pohodlně díky volně stažitelnému vývojovému prostředí Bricx Command Center. Ukázka vzhledu tohoto prostředí je na obrázku 3.2.



Obrázek 3.2 Programovací prostředí Bricx Comand Center

Jazyk NXC nepotřebuje ke svému chodu vlastní firmware. Ale pokud bychom chtěli používat například čísla v plovoucí řádové čárce, nebo nové funkce, které poskytují například řízení motoru na polohu, je třeba stáhnout nový „Enhanced NBC/NXC firmware“. Enhanced firmware (verze 1.28+) již podporuje datový typ *float*, jako ostatní jazyky. Nejnovější firmware (verze 1.31+) pak poskytuje i funkce k regulaci polohy motoru.

Program se dá členit do jednotlivých úloh, které mohou běžet paralelně. Toto umožňuje například jízdu robota a současné sledování blížícího se předmětu s pomocí dálkového senzoru, nebo vykonávání určitého úkolu vždy, když např. zvukový senzor zachytí tlesknutí nebo se stiskne dotykový senzor.

3.2.1 Základní příkazy pro obsluhu motorů a senzorů

Ke snadnému programování je již implementována řada maker a funkcí pro obsluhu motoru a senzorů. Výčet těch nejdůležitějších maker je v tabulce 3.1.

Makro	Vlastnost
OUT_A	Adresa motoru připojeného na výstup A
OUT_B	Adresa motoru připojeného na výstup B
OUT_C	Adresa motoru připojeného na výstup C
OUT_AB, OUT_AC, OUT_BC, OUT_ABC	Kombinace adres jednotlivých motorů umožňující řízení více motorů najednou
S1, S2, S3, S4	Adresy jednotlivých vstupních portů 1-4
SENSOR_TOUCH	Konfigurační makro pro dotykový senzor
SENSOR_LIGHT	Konfigurační makro pro světelný senzor
SENSOR_SOUND	Konfigurační makro pro zvukový senzor
SENSOR_LOWSPEED	Konfigurační makro pro digitální senzor (například ultrazvukový senzor)
SENSOR_1, SENSOR_2, SENSOR_3, SENSOR_4	Aktuální analogová hodnota (škálovaná pro senzor, který je připojen) na daném portu
SENSOR_MODE_RAW	Režim pro zobrazování neškálované hodnoty

Tabulka 3.1 Nejdůležitější makra

Výčet nejdůležitějších obslužných funkcí motoru je uveden v tabulce 3.2.

Funkce	Vstup(y)	Výstup / Chování
OnFwd (byte outputs, char pwr)	Adresy motorů; síla	Roztočí dané motory danou silou vpřed
OnRev (byte outputs, char pwr)	Adresy motorů; síla	Roztočí dané motory danou silou vzad
OnFwd Sync(byte outputs, char pwr, char turnpct)	Adresy motorů; síla; míra zatočení	Roztočí synchronně dané motory danou silou vpřed s možností jízdy po oblouku,

		pokud míra zatočení není nula
OnRevSync(byte outputs, char pwr, char turnpct)	Adresy motorů; síla; míra zatočení	Roztočí synchronně dané motory danou silou vzad s možností jízdy po oblouku, pokud míra zatočení není nula
Coast (byte outputs)	Adresy motorů	Odpojí dané motory od napájení (zastaví se setrvačností)
Off (byte outputs)	Adresy motorů	Zastaví dané motory pomocí elektrické brzdy
RotateMotor (byte outputs, char pwr, long angle)	Adresy motorů; síla; úhel natočení	Natočí dané motory danou silou o daný úhel
PosRegSetAngle (byte output, long angle)	Adresa motoru; cílový úhel	Reguluje natočení daného motoru na daný úhel (absolutně)
PosRegAddAngle (byte output, long angle_add)	Adresa motoru; přidaný úhel	Reguluje natočení daného motoru o daný úhel (relativně)

Tabulka 3.2 Funkce pro obsluhu motorů

Výčet nejdůležitějších obslužných funkcí senzorů je uveden v tabulce 3.3.

Funkce	Vstup(y)	Výstup / Chování
SetSensorTouch (const byte &port)	Adresa vstupního portu	Nastaví daný port jako dotykový senzor
SetSensorLight (const byte &port [, bool bActive=true])	Adresa vstupního portu; (přisvětlovací dioda zap/vyp)	Nastaví daný port jako světelný senzor s/bez přisvětlovací diody
SetSensorSound (const byte &port)	Adresa vstupního portu	Nastaví daný port jako zvukový senzor
SetSensorUltrasonic (const byte &port)	Adreas vstupního portu	Nastaví daný port jako ultrazvukový senzor
Sensor (const byte &port)	Adresa vstupního portu	Hodnota analogového senzoru na daném portu (stejné jako SENSOR_x)
SensorUs (const byte &port)	Adresa vstupního portu	Hodnota ultrazvukového

		senzoru na daném portu

Tabulka 3.3 Funkce pro obsluhu senzorů

Většinu nastavení lze provést několika způsoby. Snažil jsem se uvést ty nejkratší možné postupy. Například nastavení připojení dotykového senzoru na port číslo 1 lze zapsat jako:

SetSensorTouch(S1),

což má stejný efekt jako funkce

SetSensor(S1, SENSOR_TOUCH)

nebo volání dvou funkcí po sobě

SetSensorType(S1, SENSOR_TYPE_TOUCH),

SetSensorMode(S1, SENSOR_MODE_BOOLEAN).

Existují i další funkce, rozšiřující možnosti funkce *OnFwd* jako *OnFwdReg* a *OnFwdRegPID*, které umožňují zvolit typ regulace (od již zmíněné verze 1.31+ lze volit i regulaci polohy), potažmo konstanty PID regulátoru.

Pro zjišťování, resp. nulování otáček motoru slouží funkce *MotorTachoCount(byte output),* resp. *ResetTachoCount(byte output),* nebo funkce *OnFwdEx*, která s rozběhnutím motoru současně vynuluje čítač.

Celkově jazyk NXC nabízí pohodlné programovací prostředí pro programátory znalé jazyka C. Kód při psaní vypadá přehledně, nabízené funkce a makra zjednodušují jeho zápis na co nejmenší počet řádků. Na druhou stranu právě více možností zápisu ztěžuje orientaci v dokumentaci a nutí ne úplně zkušeného programátora se často rozhodovat, kterou metodu zápisu zvolit, což ve výsledku stojí čas.

Pro podrobnější seznámení s tímto prostředím doporučuji například pěkně vytvořený návod [11].

3.3 leJOS – NXJ

Programovací prostředí leJOS NXJ obsahuje vlastní firmware, na kterém běží kompaktní verze Java Virtual Machine. To umožňuje programovat kostku stejným způsobem jako běžné Java aplikace. Podporuje objektově orientované programování, datový typ *float*, vícevláknové aplikace, obsluhu událostí, a další a další možnosti známé z klasického programování v jazyku Java (více k dočtení například na oficiálních stránkách [3]).

Navíc, v porovnání s jazykem NXC, leJOS NXJ obsahuje o mnoho více již implementovaných uživatelských tříd a spoustu ukázkových programů.

Prostředí leJOS sice nenabízí vlastní vývojové prostředí jako NXT-G nebo NXC, ale nabízí zdarma stažitelné pluginy do existujících velmi úspěšných vývojových prostředí Netbeans a Eclipse.

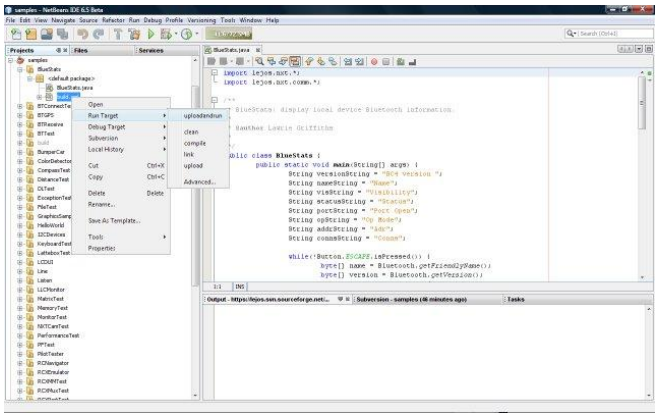
3.3.1 Instalace prostředí Netbeans

Vybral jsem si Netbeans, protože je mi bližší než Eclipse a mám s ním více zkušeností. Jeho instalace je více než snadná. Doporučuji přehledně zpracovaný návod na instalaci pluginu do Netbeans [13]. S jeho pomocí se dá postupovat bez obtíží při instalaci leJOS NXJ verze 0.8.5.

V květnu 2011 byla vydána nová verze 0.9.0, jejíž instalace probíhá stejně, ale je potřeba zmínit několik odlišností:

- cesta k leJOS balíčkům a třídám se při standardní instalaci není:
- *C:\Program Files\leJOS NXJ\lib\classes.jar*, ale
- *C:\Program Files\leJOS NXJ\lib\nxt\classes.jar*, nebo
- *...\lejosNXJProjects\classes\lib\classes.jar*
- V souboru build.properties se naopak nemusí měnit nic (pouze v případě, že se změní název projektu/třídy/package, musí se změnit I příslušné řádky)
- pro správnou funkčnost příkazu Run File (Shift+F6) je potřeba (u každého projektu zvlášť) v souboru:
  {jméno projektu}\nbproject\ide-file-targets.xml
- změnit v řádku 6:
- *name="main.class"* na
- *name="full.main.class"*

Po úspěšné instalaci vypadá prostředí jako na obrázku 3.3.



Obrázek 3.3 Vývojové prostředí Netbeans

3.3.2 Příkazy k ovládání motorů

Jak již bylo uvedeno výše, součástí instalace leJOS - NXJ je i mnoho již implementovaných funkcí, tříd a balíčků. V aktuálně nejnovější verzi 0.9.0 došlo oproti verzi 0.8.5 ke změně základních tříd, které ovládají motory.

- ***NXTRegulatedMotor***

Nejdůležitější třídou se tak stává nová třída *NXTRegulatedMotor*, která obsahuje regulátor, běžící ve vlastním vlákně s periodou 4 ms, který řídí komplexně všechny motory, dokáže řídit jejich polohu, rychlost i zrychlení, a při požadavku na novou rychlost nebo polohu uvažuje i s aktuálním pohybem motoru tak, aby výsledný pohyb byl co nejhladší, což se bude velmi hodit pro účely níže uvedených soutěžních úloh. Jediný problém vzniká, když jsou požadavky na změnu polohy volány až příliš často (s periodou řádově menší než 1 ms). Potom je ragulátor nestabilní a motor se otáčí velmi chaoticky. V praxi to ovšem nevadí, protože i jednoduchá operace jako výpis jednoho řádku na LCD trvá něco přes 1 ms. Hlavní metody třídy *NXTRegulatedMotor* přístupné uživateli jsou vypsány v tabulce 3.4. K instancím této třídy se přistupuje pomocí statických proměnných *Motor.A*, *Motor.B* a *Motor.C*.

Metoda	Poznámky
forward()	Roztočí motor vpřed
backward()	Roztočí motor vzad
changeDirection()	Změní směr otáčení
stop()	Zastaví motor okamžitě
flt()	Vypne motor, ale nazabrzdí ho
setSpeed(int speed)	Nastaví rychlost otáčení ve stupních za vteřinu (maximální rychlost je cca 900°/s)
getTachoCount()	Vrátí úhel natočení motoru ve stupních

rotate(int angle)	Natočí motor o daný úhel
rotateTo(int angle)	Natočí motor na daný úhel
rotateTo(int angle, boolean immediateReturn)	Vyšle požadavek na otočení na daný úhel, ale nečeká na jeho dokončení

Tabulka 3.4 Hlavní metody třídy NXTRegulatedMotor

3.3.3 Příkazy k ovládání senzorů

Pro senzory jsou připraveny jednotlivé třídy, odpovídající daným senzorům. Nový senzor se inicializuje vytvořením instance dané třídy se vstupním argumentem SensorPort.Sx (kde x je číslo vstupního portu). Jednotlivé třídy a jejich nejdůležitější funkce jsou uvedeny v tabulce 3.5.

Třída	Funkce	Poznámky
TouchSensor	isPressed()	Vrací *true*, pokud je senzor stisknut
LightSensor	getLightvalue()	Vrací úroveň intenzity světla v procentech
	setLow(int rawValue) setHigh(int rawValue)	Nastaví úroveň intenzity světla pro 0%, resp. 100%
SoundSensor	readValue()	Vrací úroveň intenzity zvuku
UltrasonicSensor	getDistance()	Vrací vzdálenost nejbližšího detekovaného objektu v cm

Tabulka 3.5 Základní funkce pro obsluhu senzorů

3.3.4 Pokročilejší třídy k ovládání robota

LeJOS – NXJ obsahuje i komplexnější třídy a rozhraní pro ovládání robotů, u kterých se předpokládá určité konstrukční řešení. Ve verzi leJOS – NXJ 0.9.0 jsou to zatím následující čtyři třídy.

- ***DifferentialPilot***

Pro začátečníky je nejjednodušší využít třídu *DifferentialPilot*, která je přizpůsobena k ovládání robotu s diferenciálním řízením (známým např. z tanku). Lze využít i návod na jeho sestavení [2]. Tato třída je, co se týče vývoje, zatím nejdále. Je považována za jakýsi základ řízení robota, protože konstrukční i programové řešení takovéhoto robota není nikterak obtížné. Obsahuje funkce, které umožňují kromě jízdy vpřed a vzad a rotací o daný úhel i jezdit po daném oblouku.

- ***CompassPilot***

Tato třída rozšiřuje výše zmíněnou třídu *DifferentialPilot* o další funkce, které využívají Hi-Tech kompas senzoru. To umožňuje robotu kromě jízdy „naslepo“ i korigovat vzniklé nepřesnosti v úhlech natočení.

- ***SegwayPilot***

Další velmi zajímavou třídou je třída *SegwayPilot*, která, jak název napovídá, je schopna řídit robota typu Segway v nestabilní vztyčené poloze. K tomu využívá Hi-Tech gyroskopický senzor a je propracována velmi důkladně. Na základě vstupních paremetrů je schopna řídit i roboty s různými rozměry kol. Ukázku takového robota lze spatřit například na videu (http://www.youtube.com/watch?v=V40ScvJeFxg).

- ***SteeringPilot***

Vzhledem k povaze níže uvedených soutěžních úloh se zdá být tato třída přímo ideální, avšak opak je pravdou. Po bližším prozkoumání implementace jednotlivých metod, které měly stejně jako u třídy *DifferentialPilot* umožnit i jízdu po daném oblouku, jsem zjistil, že zásadní funkce *steer(double radius)*, která je volána takřka v každé funkci, která má zajistit nějaké natočení, je implementována pouze napůl. Pokud je jako argument této funkce předáno nekonečno (nekonečně velký poloměr zatáčení odpovídá nulovému natočení předních kol), robot jede rovně, což odpovídá předpokladům. Pokud však chceme jet po kružnici s určitým konečným poloměrem, robot zkrátka natočí přední kola daným směrem o maximální možný úhel. Díky tomu není tato třída zatím příliš použitelná. Doufejme, že v prvním velkém vydání leJOS – NXJ (verze 1.0) již bude tento nedostatek napraven.

4 Návrh soutěžních úloh

4.1 Úvod k soutěžním úlohám

Součástí této bakalářské práce je i návrh dvou nových soutěžních úloh, které budou plnit studenti v rámci předmětu Roboti [6]. Jsou to úlohy „automatické parkování“ a „sledování automobilu“. Jak jejich název napovídá, cílem je vytvořit roboty, které simulují reálné automobily jak konstrukcí, tak i funkcí.

4.1.1 Typ podvozku

Co se konstrukce robotů týče, u obou úloh bude požadováno sestrojení dvounápravového čtyřkolového vozidla, jehož jediná otočná náprava je právě ta přední. Přesně po vzoru klasických automobilů (u přední nápravy se nemusí striktně otáčet každé kolo kolem vlastní

osy, ale je možno otáčet i celou nápravou kolem jedné osy). Rozchod předních a zadních kol by měl být přibližně stejný, aby konstrukce odpovídala automobilu, byť bude nastavena určitá malá tolerance vzhledem k omezeným možnostem LEGO dílů. Hnaná náprava může být přední i zadní, ale vzhledem k možným komplikacím při propojování přední otočné nápravy s hnací hřídelí doporučuji volit jako hnanou nápravu tu zadní, kde lze pohodlně umístit diferenciál pro lepší vlastnosti řízení.

4.1.2 Schopnosti robotů

Základem obou úloh bude schopnost robota sledovat 3 cm širokou černou čáru, která zde symbolizuje jízdu v jízdním pruhu. Konstrukční řešení podvozku tak může být v obou úlohách shodné. Zároveň bude potřeba v obou úlohách schopnost robota určovat vzdálenost od objektů před sebou nebo vedle sebe.

4.2 Výběr vhodného programovacího prostředí

Ve třetí kapitole jsou podrobněji rozebrána tři nejpoužívanější vývojová prostředí. Ne všechna jsou však k programování robotu řešícího níže uvedené úlohy vhodné.

4.2.1 Porovnání možností jednotlivých prostředí

- ***NXT-G***

Prostředí NXT-G je například naprosto nevhodné. Toto prostředí je určené hlavně k jednoduchému programování jednoduchých úloh. Jakékoli složitější algoritmy se zde programují jen velmi obtížně.

- ***NXC***

Jazyk NXC je již o poznání vhodnější. Avšak hned první problém nastal při řešení zdánlivě jednoduché úlohy sledování čáry.

U robotu s diferenciálním řízením se korekce směru jízdy robotu provádějí změnou rychlosti otáčení dvou motorů, což znamená, že je třeba regulovat jejich rychlosti. K regulaci rychlosti je tento jazyk dobře uzpůsoben, úloha je snadno řešitelná a robot jede podél čáry plynule.

Naproti tomu u robotu, jenž se podobá automobilu, se korekce směru jízdy provádějí změnou úhlu natočení přední nápravy, což vede na regulaci polohy jednoho motoru.

V jazyce NXC sice lze regulovat motor na požadovanou polohu, ale nelze tento požadavek (metodu *rotateTo(angle)*, která otočí motor o zadaný úhel) volit příliš často. Motor se snaží

zastavit na zadaném úhlu, přičemž další požadavky úspěšně ignoruje do té doby, než se ustálí (nutno podotknout, že se ustaluje s překmity a ustalování díky nim trvá cca půl vteřiny) a až poté přijme další požadavek na otočení na zadaný úhel. Výsledný pohyb proto není v žádném případě plynulý, ale skokový i při pomalé lineární změně požadovaného úhlu.

- ***leJOS – NXJ***

Toto prostředí se ukázalo být naprosto ideální. Výše zmíněný problém s regulací polohy v jazyce NXC je zcela vyřešen v nové verzi NXJ 0.9.0, která vyšla v květnu 2011.

Navíc toto prostředí poskytuje i přehlednou dokumentaci ke všem třídám včetně možnosti nahlédnout do jejich implementace. Tím se stává spolu s výhodami prostředí Netbeans či Eclipse velmi mocným nástrojem a programování jde opravdu velmi snadno.

4.2.2 Závěr porovnání

Vzhledem k výše uvedeným kladům a záporům jednotlivých programovacích prostředí, vychází jako nejlepší (a prakticky jediná) možnost prostředí leJOS – NXJ.

Protože programování v jazycích NXT-G a NXC se ukázalo být nesrovnatelně náročnější, než v jazyce NXJ, a protože sledování čáry je základem pro obě soutěžní úlohy, vybral jsem si jako programovací prostředí pro jejich řešení pouze jazyk NXJ. Řešení úloh ve zbylých dvou prostředích proto nejsou u úloh uvedena.

4.3 Úloha: Automatické parkování

Tato úloha vyžaduje velkou zdatnost. Je náročná jak na technické provedení, tak na jeho programování. Zároveň je potřeba obě části skloubit a při práci na jedné myslet i na tu druhou.

4.3.1 Cíl úlohy

Cílem této úlohy je sestavit a naprogramovat robota tak, aby co nejrychleji a nejpřesněji zaparkoval na jednom z volných parkovacích míst na soutěžním plánu. Případné dotyky nebo vyčnívání z řady budou penalizovány.

4.3.2 Technické specifikace

- ***Použité díly***

Týmy obdrží sadu základních dílů LEGO Mindstorms Education Set (9797), která obsahuje NXT inteligentní kostku, motory, senzory a další technické díly, dále sadu rozšiřujících dílů (9648), druhý světelný senzor (9844) a síťový adaptér (9833). Po skončení soutěže je vrátí zpět v původním a neporušeném stavu.

Žádné jiné díly nesmí být na stavbu robota použity.

- ***Konstrukce robota***

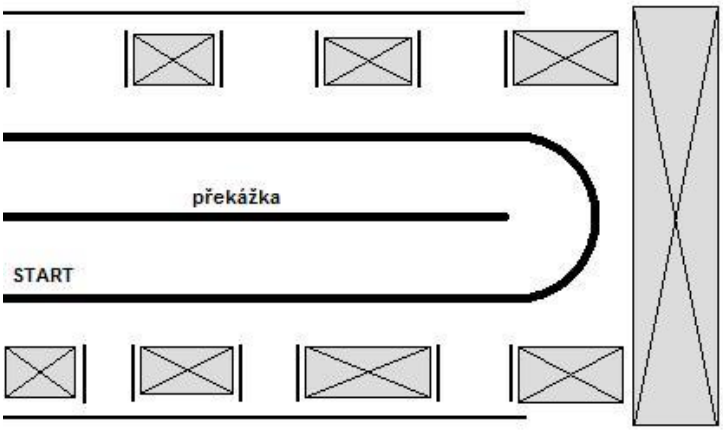
Robot musí v prvé řadě simulovat klasický automobil. To znamená, že musí mít čtyři kola stejného typu umístěná na dvou nezávislých nápravách. Právě přední náprava musí být otočná (lze otáčet každým kolem kolem vlastní osy nebo celou nápravou kolem jediné osy). Motorem poháněná náprava může být jak přední, tak i zadní.

Robot by měl měřit na délku alespoň 15 cm a na šířku alespoň 10 cm. Rozměry robotu nejsou shora omezeny, protože by stejně neposkytovaly žádnou výhodu. Výška robotu není omezena vůbec.

Robot musí být vybaven alespoň jedním světelným senzorem určeným ke sledování černé čáry.

- ***Soutěžní plán***

- Soutěžní plán (viz obrázek 4.1) má rozměry 250 x 150 cm.
- Dráha je tvořena 3 cm širokou černou čárou, která opisuje tvar písmene „U“.
- Začátek a konec dráhy jsou na koncích černé čáry. Zatáčka je levotočivá, z čehož lze usoudit, kde je začátek a kde je konec dráhy.
- Poloměr zatáčky není menší než 20 cm.
- Na vnější straně plánu (ve vzdálenosti od čáry 15 cm) jsou zaparkována auta (symbolizovaná kvádry o šířce 15-20 cm, výšce 10-15 cm a minimální délce 10 cm).
- Mezi auty jsou umístěna volná parkovací místa s postupně rostoucí velikostí mezer.
- Délky parkovacích míst se pohybují od 15 cm na začátku dráhy až do 40 cm na konci dráhy.
- Každé volné parkovací místo má na obou svých krajích, ve vzdálenosti 2 cm od aut před a za ním, 1 cm širokou a 20 cm dlouhou černou čáru.
- Pokud jsou za sebou zaparkovaná dvě auta, pak vzdálenost mezi nimi je vždy menší než 10 cm.
- Po celé délce oblouku zatáčky nejsou žádná volná parkovací místa.
- Mezi dvěma rovnoběžnými konci dráhy je překážka, zamezující robotům zkracovat si dráhu.
- Ve vzdálenosti 40 cm od okraje černé čáry je na vnější straně dráhy další černá čára široká 1 cm (symbolizující obrubník). Šířka parkovacích míst je tak 25 cm.



Obrázek 4.1 Možná varianta soutěžního plánu

4.3.3 Pravidla

Každý soutěžící tým má k dispozici dva pokusy, přičemž mezi jednotlivými pokusy nesmí provádět žádné programové úpravy robota. Může ale například přesunout či pootočit senzory nebo přesněji nastavit výchozí nulový úhel natočení přední nápravy.

Počítá se vždy lepší z obou časů.

- ***Průběh jízdy***
  - Soutěžící umístí robota na start, nejlépe tak, aby přední světelný senzor, který sleduje černou čáru, byl ve své ideální pozici.
  - Soutěžící robota spustí a po protnutí pomyslné čáry (kterou určí organizátor soutěže) se začne měřit čas.
  - Robot musí sledovat černou čáru, dokud nenalezne adekvátní parkovací místo. Pokud robot čáru ztratí a podaří se mu ji bez kolize opět najít, může pokračovat v jízdě. Pokud ji ztratí a viditelně nemá šanci se na dráhu vrátit (např. jezdí-li v kruhu nebo se zasekl o překážku), má soutěžící k dispozici ještě druhý a poslední pokus (začíná opět od začátku).
  - Po nalezení vhodného místa na zaparkování by měl robot vykonat sérii pohybů, které ho (nejlépe bez kolizí) umístí do volného parkovacího místa.
  - Po zaparkování by měl robot zastavit. Robot oznámí jasným zvukovým signálem (melodií), že už zaparkoval a dál se již pohybovat nebude. V ten okamžik se zastaví měřený čas, který se zapíše.
  - Robot by neměl žádnou svojí částí vyčnívat z řady zaparkovaných aut směrem k dráze, kterou tvoří pomyslná linie vzdálená 15 cm od okraje černé čáry. Neměl by ani stát na „obrubníku“, který je vzdálen 40 cm od černé čáry. V obou případech bude penalizován.
  - Pokud robot není schopen z jakýchkoli důvodů dokončit svoji jízdu, nebo je více než polovinou svého objemu mimo parkovací místo (má soutěžící k dispozici ještě druhý a poslední pokus (začíná ale opět od začátku).

- ***Hodnocení***

Hodnotí se především čas robota, potřebný k úspěšnému zaparkování. K tomuto času se přičtou případné penalizace nebo bonifikace.

- ***Penalizace***

Penalizuje se každý dotyk robota se zaparkovanými auty, přičemž záleží na tom, zda je dotyk řízený, nebo není.

Řízeným dotykem se myslí situace, kdy robot, používající dotykový senzor při parkování, dotyk očekává a ihned na něj reaguje. Náraz také nesmí být příliš silný. Tomu odpovídá tzv. parkování „na ťuknutí“. Řízený dotyk je penalizován přičtením tří vteřin k celkovému času.

Neřízeným dotykem se myslí nečekaná srážka robota se zaparkovaným autem nebo příliš silný dotyk. To je penalizováno přičtením pěti vteřin k celkovému času.

Další penalizace přičtením až 10 vteřin je možná při nesprávném zaparkování (např. vyčnívající část auta z řady nebo parkování na „obrubníku“). O této případné penalizaci rozhodne svým subjektivním hodnocením organizátor soutěže.

- ***Bonifikace***

Pokud robot zaparkuje například s obdivuhodnou lehkostí, nebo je ukázkově vyrovnán v řadě zaparkovaných aut, je možné výsledný čas zkrátit až o tři vteřiny. O tomto případném zvýhodnění rozhodne taktéž organizátor soutěže.

- ***Fair-play***

Robot musí být plně autonomní. Nesmí přijímat žádné příkazy přes Bluetooth, ani být řízen člověkem pomocí jiných senzorů.

Zároveň členové ostatních týmů nesmí vstupovat do dráhy ultrazvukového senzoru robotu, nebo záměrně vysílat rušivé ultrazvukové signály vlastním senzorem, čímž by ovlivnili průběh jízdy robota.

4.3.4 Konstrukční řešení robota

- ***Celkový náhled***

Pro tuto úlohu jsem se snažil vytvořit kompaktního robota. Při jeho stavbě jsem se zaměřil na to, aby robot byl co nejkratší, a aby měl co nejlepší manévrovací schopnosti (minimální poloměr otáčení). Výsledkem je robot, který zabírá včetně senzorů na délku 19 cm a na šířku 15 cm, dokáže plynule projíždět i zatáčky s poloměrem cca 10 cm (což je dvakrát méně, než

má minimální poloměr zatáčky na soutěžním plánu) a zvládne pohodlně zaparkovat i na parkovacím místě o délce 25 cm.

Celkový výsledek tvořivé práce je k vidění na obrázku 4.2, nebo v elektronické příloze ve složce Models jako trojrozměrný model vytvořený v programu LEGO Digital Designer.

Obrázek 4.2 Celkový náhled robota

- ***Zadní hnaná náprava***

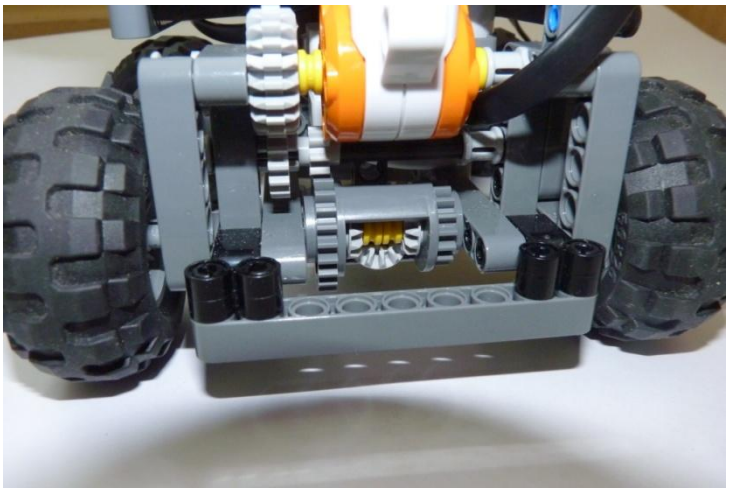
Základem zadní nápravy je diferenciál, který umožňuje efektivně rozdělovat sílu motoru na jednotlivá kola zadní nápravy a zlepšit tak jízdní vlastnosti v zatáčkách.

Protože diferenciál nelze pohánět přímo, ale je zapotřebí ozubených kol, a protože v této úloze není důležitá maximální rychlost robota, ale spíše jeho přesnost, použil jsem s ohledem na úspornost řešení systém ozubených kol, který zajišťuje poměr otáček kol k otáčkám motoru 5/12. Pouhým vzájemným prohozením dvou ozubených kol lze dosáhnout poměru 4/15, ale potom je již robot zbytečně pomalý.

Motor jsem umístil do svislé polohy nad zadní nápravu, protože na výšce robotu v tomto případě nezáleží.

Díky vůli v diferenciálu sice robot i při konstantní rychlosti motoru mírně poskakuje, avšak celkově je tato chyba zanedbatelná a výhody samotného diferenciálu ji dalece předčí.

Celá náprava je velmi pevná a „nárazník“, který zároveň spojuje oba konce nápravy, poskytuje mnoho možností pro připojení dalších senzorů. Detail samotné nápravy je zobrazen na obrázku 4.3.

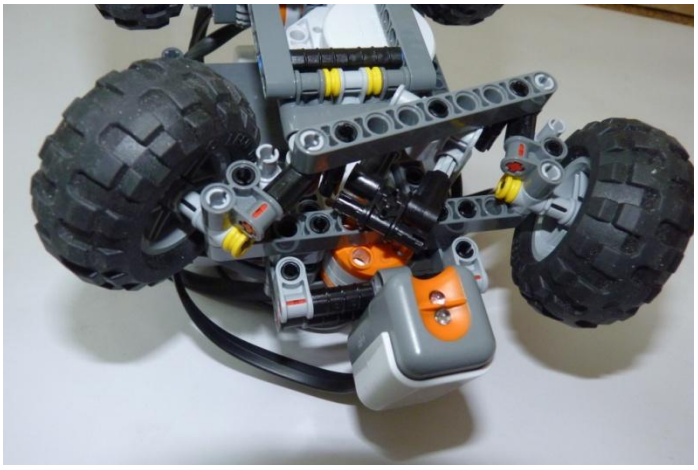


Obrázek 4.3 Diferenciál

- ***Přední otočná náprava***

Po několika nezdařených pokusech o vytvoření přední otočné nápravy s efektivním přenosem otáček motoru na úhel natočení nápravy se mi podařilo vytvořit dva druhy nápravy.

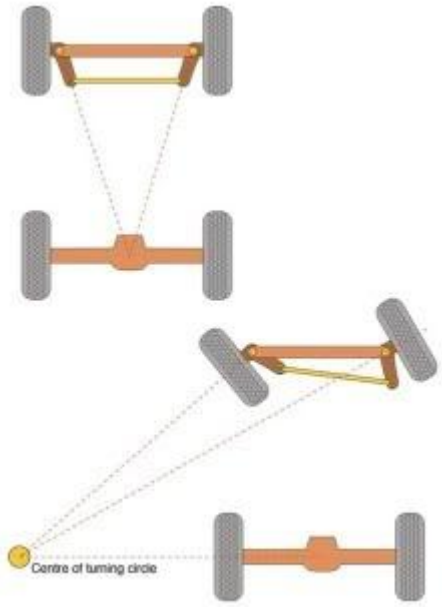
První je asi ta nejpevnější možná. Každé kolo se otáčí kolem vlastní, velmi blízké osy a k přenosu pohybu od motoru ke kolům je použito minimum dílů, což zajišťuje minimální vůli v uložení kol. Dva dlouhé rovnoběžné díly připojené přímo k motoru fungují i jako pevný doraz při maximálním úhlu natočení. Tato náprava je velmi stabilní, zabírá jen málo místa a umožňuje natočit přední kola až do úhlu cca 50°. Její hlavní (a snad jediná) nevýhoda je pozorovatelná při zatáčení robota pod velkým úhlem. Tím, že jsou obě kola vzájemně rovnoběžná, nemají společný střed kružnic, které opisují. Vnější kolo se tak smýká, což může na různých površích vyvolat i nežádoucí poskakování robota. Názorněji bude vysvětleno níže.



Obrázek 4.4 Přední náprava

Druhá a pokročilejší verze, jejíž detail je na obrázku 4.4, funguje obdobně. Osy otáčení jednotlivých kol jsou o trochu dále od samotných kol, je použito více dílů s větší vůlí, ale

zásadní rozdíl je v tom, že tato náprava odstraňuje nedostatek té předchozí. Zabránění smýkání kol v zatáčkách je dosaženo díky Ackermannově geometrii řízení, jejíž schematický princip funkčnosti je na obrázku 4.5.



Obrázek 4.5 Ackermannova geometrie řízení

S použitím této metody se tak všechny osy kol při zatáčení protínají v jednom bodě a žádné kolo se nesmýká.

Pro účely testování robota jsem používal výhradně druhou verzi nápravy, která se zdála být užitečnější.

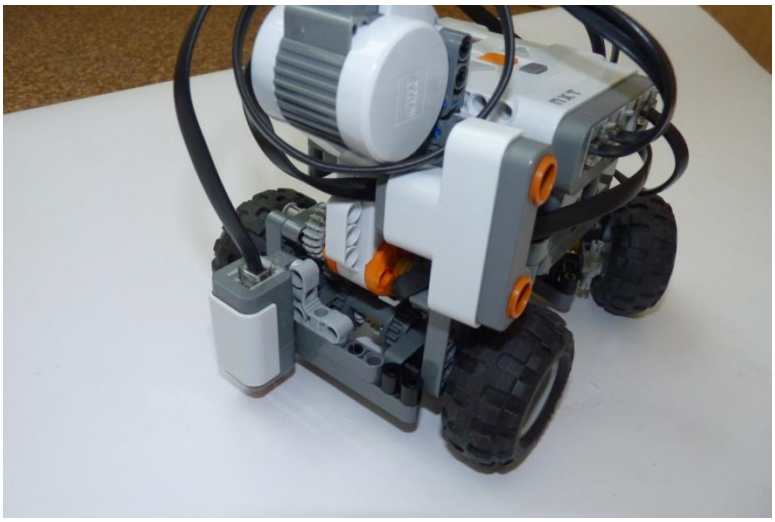
- ***Umístění senzorů***

Světelný senzor určený pro sledování čáry jsem umístil těsně před přední nápravu doprostřed (viz obr. 4.4).

Druhý světelný senzor, který je k dispozici, jsem umístil na zadní nárazník blíže levému zadnímu kolu. Tento senzor je využit pouze při parkování k detekci zadní černé čáry. K této čáře se robot přibližuje pod určitým úhlem levým kolem napřed, proto je žádoucí, aby byl senzor blíže tomuto kolu pro včasnější detekci černé čáry.

Ultrazvukový senzor, určený ke zkoumání, zda se na parkovacích místech vedle robota nachází, či nenachází auto, jsem umístil do prostoru u zadního motoru a NXT kostky. Senzor je ve svislé poloze jednak z čistě prostorových důvodů, ale hlavně kvůli přesnější detekci hran aut, které jsou také svislé. Detekovaná hrana je tak mnohem ostřejší.

Umístění senzorů na zadní části robota je zachyceno na obrázku 4.6. Žádné jiné senzory použity k řešení této úlohy nebyly.



Obrázek 4.6 Umístění senzorů

4.3.5 Programové řešení robota

- ***Rozbor úlohy***

Tato úloha se dá rozdělit do tří navzájem nezávislých dílčích částí. Každá část má jasně definované vstupní a výstupní veličiny:

- Sledování čáry
  - vstup: Světelný senzor
  - výstup: Úhel natočení přední nápravy a pohyb robota vpřed
- Sledování zaparkovaných aut
  - vstup: Ultrazvukový senzor
  - výstup: Velikost mezery mezi auty a její pozice
- Parkovací sekvence
  - vstup: Přední a zadní světelný senzor
  - výstup: Pojíždění dopředu / dozadu

- ***Sledování čáry***

Pro jednoduchost a modularitu programu jsem si vytvořil tři rozhraní:

- *SteeringVehicle* - poskytuje funkce pro pohyb robota s přední otočnou nápravou
- *Tachometer* - poskytuje měření ujetých vzdáleností v mm
- *LineFollower* – poskytuje schopnost sledovat barevné rozhraní

Dále jsem vytvořil třídu *SteeringLineFollower*, která všechna tato rozhraní implementuje. Instance této třídy představuje robota, který je schopen sledovat čáru. Pro uživatele poskytuje hlavně funkce:

- *forward (), backward(), stop()* – jízda vpřed, vzad a zastavení
- *setSpeed(int speed)* – změna rychlosti jízdy
- *followLine(), stopFollowing()* – zapnutí a vypnutí sledování čáry
- *getTraveledDistance(int from)* – měření ujeté vzdálenosti

Co se týče rozhraní *SteeringVehicle* a *Tachometer*, jejich implementace je značně intuitivní a všechny funkce mají nejvýše pár řádků. Nebudu je zde proto uvádět, ale jsou součástí elektronické přílohy v rámci kompletně zdokumentovaných zdrojových kódů.

Implementace rozhraní *LineFollower* je již o poznání zajímavější. Vyžaduje totiž vlastní regulátor, který na základě údajů ze světelného senzoru určuje úhel natočení přední nápravy. Světelný senzor jsem zkalibroval pomocí funkcí *setLow* a *setHigh* tak, že černé barvě čáry odpovídá hodnota 0% a bílé barvě podložky hodnota 100%. Robot se pak bude snažit držet rozhraní těchto barev, čemuž odpovídá referenční hodnota 50%.

Při pomalé jízdě (do cca 40 mm/s) bohatě postačí řízení odchylkou. Takový regulátor se nazývá proporcionální, neboli P regulátor, jehož rovnice má tvar

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

kde k_p je konstanta proporcionální složky, r je referenční hodnota (v tomto případě 50), y je aktuální zkalibrovaná hodnota senzoru (od 0 do 100) a u je akční zásah (v tomto případě úhel natočení motoru). Hodnota proporcionální konstanty k_p je určena tak, aby krajním hodnotám ze senzoru odpovídaly mezní úhly natočení nápravy (čímž je zajištěn plný rozsah).

Se zvyšující se rychlostí robota je třeba postupně omezovat proporcionální složku regulátoru (ve skutečnosti to odpovídá posilovačům řízení v automobilech, které rovněž omezují svůj zásah ve vysokých rychlostech při jízdě po dálnici), jinak se stává sledování nestabilním. Snížením vlivu proporcionální složky se ale také zmenší maximální rozsah úhlů natočení nápravy a zvětší minimální poloměr zatáček, které je robot schopen projet.

To lze kompenzovat integrační složkou regulátoru, ze kterého se stane tzv. PI regulátor, který má rovnici (pouze rozšířenou oproti předchozí)

$$u = k_p \cdot (y - r) + k_i \cdot \sum_{i=1}^{n} (y_n - r) \cdot dt$$

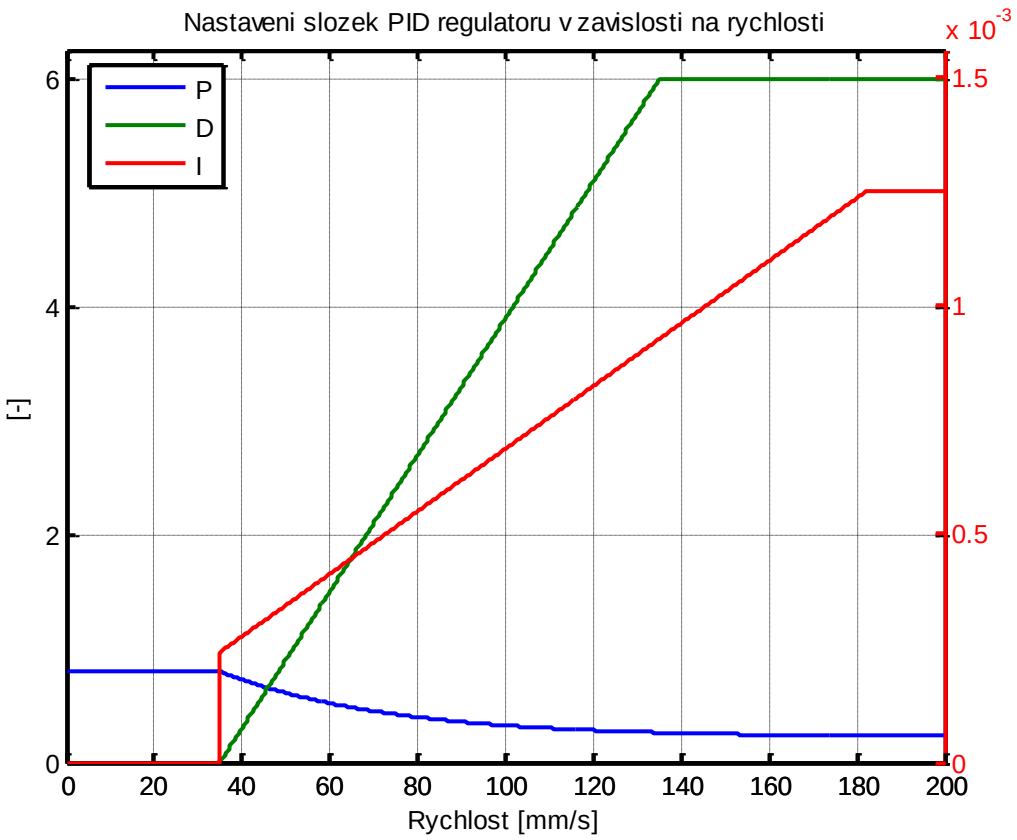
kde, kromě již uvedených proměnných a konstant, k_i je integrační konstanta a dt je perioda vzorkování. Suma v této rovnici je saturována maximální hodnotou tak, aby integrační část regulátoru svým rozsahem přesně doplňovala proporcionální část na maximální možný rozsah úhlů natočení nápravy.

Se zvyšující se rychlostí je také třeba přidat a postupně zvyšovat derivační složku regulátoru, která odhaduje na základě porovnání dvou posledních regulačních odchylek budoucnost, čehož je vzhledem k relativně úzkému barevnému rozhraní, a tudíž horší manévrovatelnosti ve vyšších rychlostech zapotřebí. Takovému regulátoru se říká PID regulátor a jeho rovnice má tvar

$$u = k_p \cdot (y - r) + k_i \cdot \sum_{i=1}^{n} (y_n - r) \cdot dt + k_d \cdot [(y - r) - (y_{-1} - r)]/dt$$

kde, kromě již zmíněných proměnných a konstant, k_d je derivační konstanta.

Konstanty PID regulátoru pro tuto úlohu jsou tedy závislé na rychlosti robotu. Po experimentálních měřeních a testování konstant při jednotlivých rychlostech jsem vytvořil aproximační funkce, které dobře určují průběh konstant při změně rychlosti v celém rozsahu rychlostí, jaké je robot schopen vyvinout. Jak lze vyčíst z grafu na obrázku 4.7, integrační a derivační konstanty se podařilo linearizovat v celém daném rozsahu a proporcionální konstanta odpovídá exponenciále. Všechny konstanty mají své omezující podmínky. Robot je schopen vyvinout maximální rychlost kolem 180 mm za vteřinu. Pro vyšší rychlosti (kterých již daný robot nedosáhne) nebyl regulátor testován a jeho konstanty jsou tak jen určitým odhadem.



Obrázek 4.7 Graf závislosti konstant PID regulátoru na rychlosti

V podúloze sledování čáry je tak jediným úkolem navrhnout vhodný regulátor k řízení úhlu natočení přední nápravy. Výše uvedený regulátor upravuje akční zásah každých 10 ms a je schopen řídit robota podle 3 cm silné černé čáry v plném rozsahu rychlostí, které je schopen vyvinout (v přepočtu až 180 mm za vteřinu).

Pro zvětšení přesnosti další podúlohy je vhodné nastavit přiměřenou a konstantní rychlost pohybu robota.

- ***Sledování zaparkovaných aut***

Pro přidání dalších schopností robotu je třeba třídu *SteeringLineFollower* rozšířit. Vytvořil jsem proto třídu *ParkingRobot*, která je potomkem třídy *SteeringLineFollower* a dědí od ní všechny výše uvedené užitečné funkce. Dále přidává funkce vlastní a umožňuje tak řešení této i následující podúlohy.

Mezi hlavní přidané (uživatelsky přístupné) funkce patří:

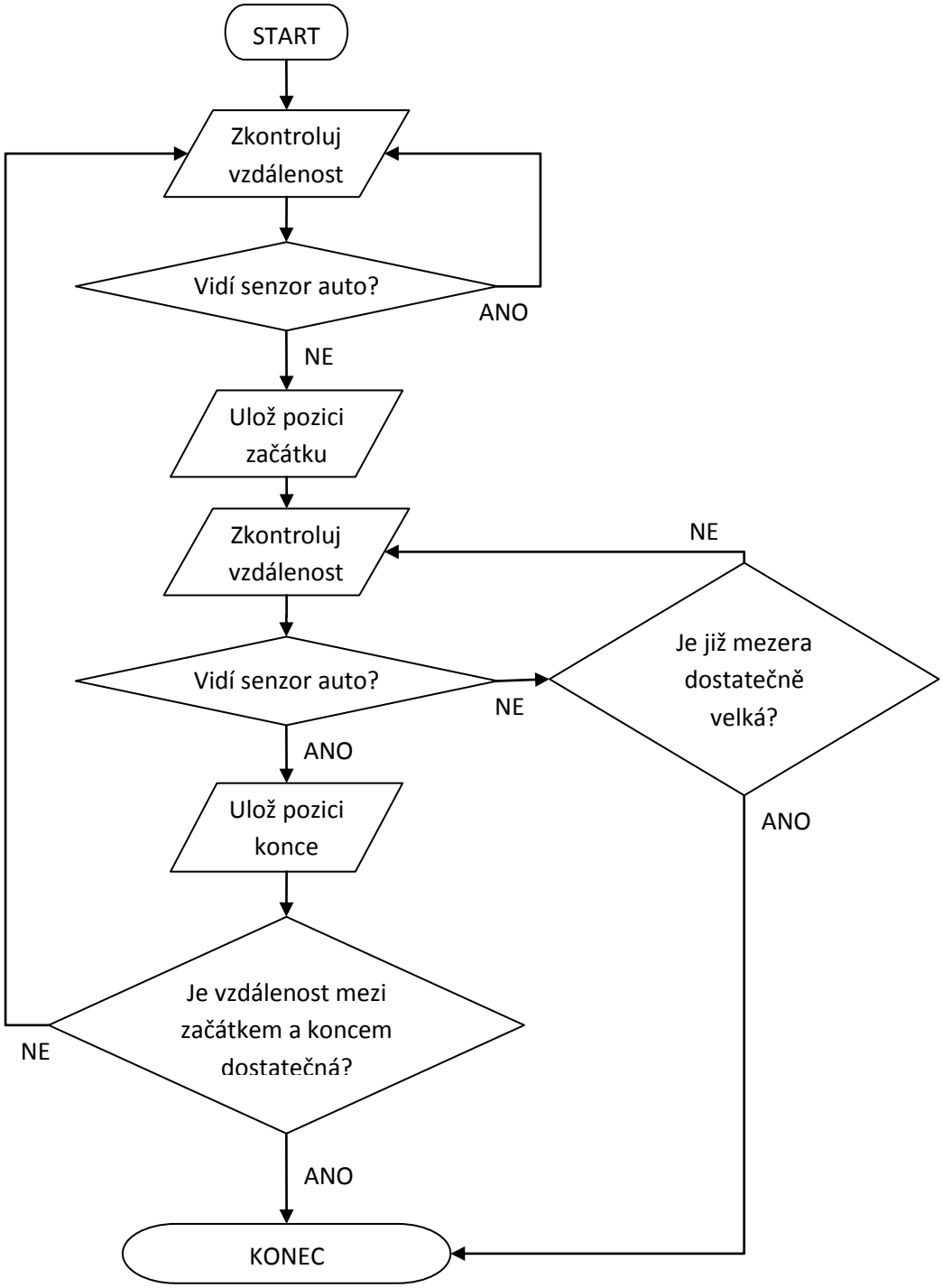
- *findParkingSpot()* – funkce běží tak dlouho, dokud nenalezne vhodné místo na zaparkování (potom je návratovou hodnotou jeho délka)

- *park(int length)* – spustí parkovací sekvenci (s možným využitím informace o délce parkovacího místa)

Ke sledování zaparkovaných aut slouží připojený ultrazvukový senzor. Ten je vzdálen od zaparkovaných aut cca 10 cm. Za předpokladu úspěšného splnění podúlohy sledování čáry a klidné jízdy podél ní je tento senzor schopen určit hrany zaparkovaných aut, a tím i zjistit (s pomocí funkcí rozhraní *Tachometer*), kde přesně začíná volné parkovací místo. Senzor se rozhoduje podle prahové hodnoty 20 cm, zda vedle dráhy je, či není volné místo.

Tato podúloha se spouští ve chvíli, kdy robot urazí zhruba 20 cm od začátku (což odpovídá momentu, kdy ultrazvukový senzor protne startovací linii), aby se zamezilo zaparkování robota mimo soutěžní plán. Celá podúloha běží paralelně s podúlohou sledování čáry. Při ukončení této podúlohy se rovněž ukončí i sledování čáry a započne koncová podúloha čítající toliko parkovací sekvenci.

Celý vývojový diagram této podúlohy znázorňující rozhodovací proces robota je vidět na obrázku 4.8.



Obrázek 4.8 Vývojový diagram podúlohy "Sledování zaparkovaných aut"

- ***Parkovací sekvence***

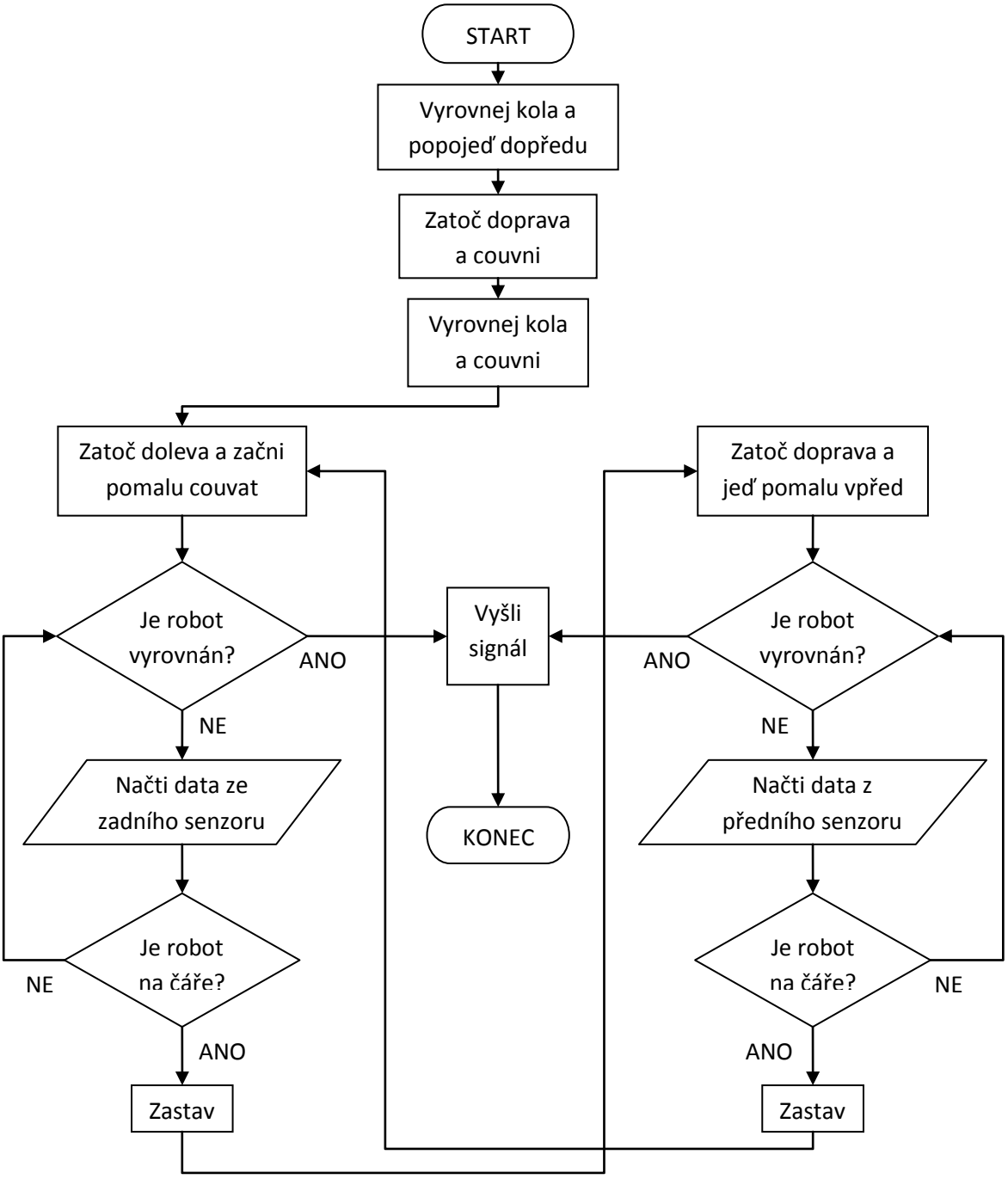
Ihned po nalezení vhodného parkovacího místa se spustí parkovací sekvence. Robot využije informaci, že zadní senzor právě nalezl hranu dalšího automobilu a ví tak, kde se automobil vpředu od parkovacího místa přesně nachází. Pak začne provádět sekvenci pohybů,

které ho navedou do volného prostoru. V poslední fázi parkování využívá robot světelných senzorů k detekci obou černých čar.

Robot se snaží vyrovnat do polohy rovnoběžné s tou, ve které parkovací sekvenci začal provádět. Toho je dosaženo jízdou po stejně dlouhých obloucích vzad s maximální výchylkou kol napřed vlevo a potom vpravo. Tím se robot otočí kolem své osy vždy o opačný úhel. Pokud během couvání (s maximální výchylkou kol vlevo) objeví senzor černou čáru, vyrovná úhel jízdou vpřed s maximální výchylkou kol vpravo. Pokud narazí naopak na černou čáru vpředu, opakuje robot tyto pohyby tak dlouho, dokud se úplně nevyrovná. Názornější popis je uveden ve vývojovém diagramu na obrázku 4.9.

K přesnější jízdě po oblouku je použita maximální výchylka kol zvětšená o 5 stupňů, čímž je dosaženo úrovně hardwarového dorazu (místo softwarového v ostatních částech úlohy). Tím se minimalizuje vliv špatného manuálního nastavení nulového úhlu před začátkem jízdy.

Po úspěšném zaparkování robot vyšle dva krátké zvukové signály pro zastavení měření času.



Obrázek 4.9 Vývojový diagram parkovací sekvence

4.3.6 Výsledek

Podařilo se velmi úspěšně navrhnout, postavit a naprogramovat robota, který řeší komplexní úlohu automatického parkování. Robot, jenž sám měří na délku 19 cm, je schopen v čase kolem 25 vteřin hladce zaparkovat i na místě dlouhém 25 cm. Ukázkový záznam z jízdy je umístěn ke shlédnutí na webové adrese http://www.youtube.com/watch?v=c54t-IIUXU4 a v elektronické příloze ve složce Videos.

Úloha byla naprogramována v jazyce NXJ, jehož verze 0.9.0 poskytuje tolik potřebné funkce pro efektivní řízení polohy motoru. Ve verzi 0.8.5 (a starší) tak přiložený kód nebude fungovat.

4.4 Úloha: Sledování automobilu

Tato úloha je zčásti podobná jako předchozí úloha automatické parkování. Vyžaduje podobné konstrukční řešení, ale už nemusí být kladen důraz na rozměry robota. Úloha také dovoluje rozmanitější způsoby řešení a tím se stává zajímavější. Zásadní je zde ovšem práce s ultrazvukovým senzorem, který ne vždy pracuje tak, jak bychom chtěli.

4.4.1 Cíl úlohy

Cílem této úlohy je vytvořit robota, který je schopen sledovat automobil jedoucí před ním. Robot by od něj měl udržovat rozumný odstup a přitom automobil neztratit z dohledu. V žádném případě by do něj neměl narazit.

4.4.2 Technické specifikace

- ***Použité díly***

Týmy obdrží sadu základních dílů LEGO Mindstorms Education Set (9797), která obsahuje NXT inteligentní kostku, motory, senzory a další technické díly, dále sadu rozšiřujících dílů (9648) a síťový adaptér (9833). Po skončení soutěže je vrátí zpět v původním a neporušeném stavu.

Žádné jiné díly nesmí být na stavbu robota použity.

- ***Konstrukce robota***

Robot musí v prvé řadě simulovat klasický automobil. To znamená, že musí mít čtyři kola stejného typu umístěná na dvou nezávislých nápravách. Právě přední náprava musí být otočná (lze otáčet každým kolem kolem vlastní osy nebo celou nápravou kolem jediné osy). Motorem poháněná náprava může být jak přední, tak i zadní.

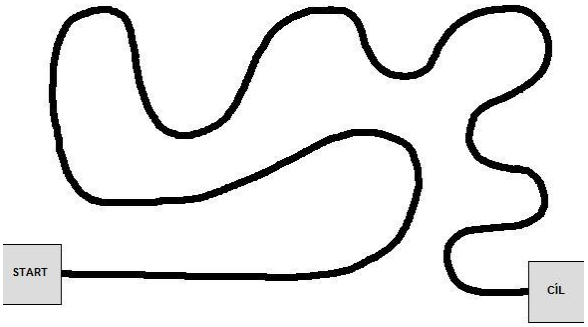
Minimální rozměry robota nejsou stanoveny, ale robot by neměl přesáhnout svojí délkou 40 cm ani svojí šířkou 30 cm. Výška robotu není omezena vůbec.

Robot musí být vybaven alespoň jedním světelným senzorem určeným ke sledování černé čáry a měl by být vybaven i senzorem ultrazvukovým pro sledování automobilu před sebou.

- ***Soutěžní plán***

Soutěžní plán tvoří deska o rozměru 250 x 150 mm. Na ní je nakreslena zakřivená černá čára. Může vypadat například jako na obrázku 4.10. Čára splňuje všechny následující požadavky:

- Šířka čáry je 3 cm po celé její délce.
- Minimální poloměr všech zatáček je 15 cm.
- Dráha se nepřiblíží sama sobě na menší vzdálenost, než je 10 cm.
- Vzdálenost dráhy od okraje soutěžního plánu není nikde menší než 15 cm.
- „Klikatost“ čáry se postupně od začátku k cíli zvětšuje, což má za následek rostoucí náročnost úlohy s ujetou vzdáleností.



Obrázek 4.10 Jedna z variant soutěžního plánu

- ***Sledovaný automobil***

Zde je ponechán určitý prostor. Ideální by bylo, kdyby sledovaným automobilem byl jiný robot, který sleduje čáru konstantní rychlostí, ale jsou možná i jiná řešení.

Organizátor při vyhlášení soutěže určí, jak bude sledovaný automobil nakonec vypadat. S tímto automobilem pak soutěžící testují na testovací dráze a stejný automobil bude použit i při samotné soutěži.

Rychlost sledovaného automobilu by měla být přiměřená a konstantní (soutěžícím však předem neznámá, známo pouze rozmezí – například 90 - 110 mm za vteřinu), přičemž jsou možné i různé modifikace této úlohy s proměnnou rychlostí automobilu.

4.4.3 Pravidla

Každý soutěžící tým má k dispozici dva pokusy. Mezi jednotlivými pokusy se nesmí provádět žádné programové úpravy.

- ***Průběh jízdy***
  - Nejprve soutěžící umístí robota na začátek čáry (nejlépe tak, aby světelný senzor byl ve své ideální sledovací pozici).
  - Sledovaný automobil je umístěn 20 cm před robota na rovném úseku dráhy.
  - Soutěžící poté svého robota spustí. Pokud je robot naprogramován správně, měl by buď zůstat stát, nebo se pouze přiblížit automobilu a pak se zastavit.
  - Poté se začne pohybovat i sledovaný automobil.
  - Robot musí po celou dobu pokusu sledovat černou čáru. Pokud ji ztratí nebo se od ní vzdálí o více než 20 cm, pokus končí a jako „dojezd“ je zaznamenáno poslední místo na dráze, kde robot ještě viditelně sledoval čáru.
  - Robot by měl sledovat automobil z rozumné vzdálenosti (zhruba od 10 do 70 cm). Pokud ho robot viditelně ztratí z dohledu na delší časový úsek (cca 10s), pokus končí a jako „dojezd“ se označí poslední místo, kde robot automobil ještě sledoval. V tomto případě určení „dojezdu“ závisí hlavně na organizátorovi soutěže.
  - Robot by neměl v žádném případě do automobilu narazit. Pokud se tak stane, je místo nárazu označeno jako „dojezd“.
  - Na konci dráhy sledovaný automobil zastaví. Robot by měl zastavit za ním ve vzdálenosti do 30 cm tak, aby do něj nenarazil. Pokud robot zastaví od automobilu dále, než je 30 cm, jako „dojezd“ je označena poloha robota, kde zastavil.
  - Pokud se robot nedopustil žádné z výše uvedených chyb a zastavil na konci dráhy, viditelně sledujíce automobil před sebou ve vzdálenosti menší, než je 30 cm, je hodnocen maximálním možným ohodnocením (vyšším než v kterémkoli jiném případě).

- ***Hodnocení***

Hodnotí se vzdálenost, po kterou byl robot schopen plnit tuto úlohu.

Vzdáleností se rozumí délka dráhy od začátku až do místa „dojezdu“, označeného při chybě robota.

Počítají se oba pokusy, přičemž horší ze dvou se uplatní při rozhodování mezi týmy se stejnou ujetou vzdáleností (může nastat, pokud více týmů zvládne projet celou drahou bez úhony). Pokud je stále mezi týmy nerozhodný stav, provedou se třetí pokusy u týmů, kterých se to týká. Pokud ani to nerozhodne, skončí tyto týmy na stejném místě.

- ***Fair-play***

Robot musí být plně autonomní. Nesmí přijímat žádné příkazy přes Bluetooth, ani být řízen člověkem pomocí jiných senzorů.

Zároveň členové ostatních týmů nesmějí vstupovat do dráhy ultrazvukového senzoru robotu, nebo záměrně vysílat rušivé ultrazvukové signály vlastním senzorem, čímž by ovlivnili průběh jízdy robota.

4.4.4 Konstrukční řešení sledovaného automobilu

Vzhledem k potížím při určování vzdálenosti ultrazvukovým senzorem od nerovného a nekolmého předmětu, jakým LEGO robot bezesporu je, se zde nabízejí určité alternativy vedle použití dalšího robota jako sledovaného automobilu.

První alternativou (použitou při testování robota v rámci této bakalářské práce) je nepoužít LEGO robota, ale jiný objekt srovnatelných rozměrů a pohybovat s ním ručně. Já jsem použil papírovou krabici na kolech, která připomíná návěs nákladního automobilu. Jeho zadní stěna poskytuje dostatečně velkou rovnou kolmou plochu, takže robot nemá problém s jeho lokalizací. Jak již bylo o ultrazvukovém senzoru řečeno v kapitole 2.3.5, problém nastává při zatočení automobilu již o cca 20°. Naměřenou vzdáleností pak není vzdálenost automobilu, ale prvního předmětu, ke kterému se signál ze senzoru dostane odrazem od automobilu.

Má-li tato úloha simulovat sledování automobilu jiným automobilem, a vezmeme-li v úvahu nedokonalost ultrazvukového senzoru a fakt, že reálné automobily mají v zadních světlech zabudovaná odrazová skla (např. tzv. koutové odražeče), považuji za logické opatřit sledovaný objekt také nějakými podobnými odraznými plochami. Klasické odrazky, známé z bicyklů, však použít nejdou, protože jejich vnitřní struktura odráží zpět pouze světlo, zatímco zvuk je odražen jejich hladkou plochou na povrchu.

Jako odrazovou plochu, poskytující pravoúhlé odrazy, jsem použil dřevěnou mřížku, která je dle obrázku 4.11 připevněna na zadní stěnu automobilu[1]. Tato jednoduchá úprava významně zlepšila schopnost robota správně detekovat vzdálenost od automobilu.

[1] Budu i nadále používat označení „automobil“, ikdyž se jedná spíše o „návěs“, kvůli zachování kontextu.

Obrázek 4.11 Sledované vozidlo s odrazovou mřížkou

Druhou alternativou je použít LEGO robota jako sledovaný automobil, ale opatřit jej podobnými odraznými plochami, jaké jsem uvedl výše. Není přitom nutné používat pouze LEGO díly.

4.4.5 Konstrukční řešení robota

- ***Celkový náhled***

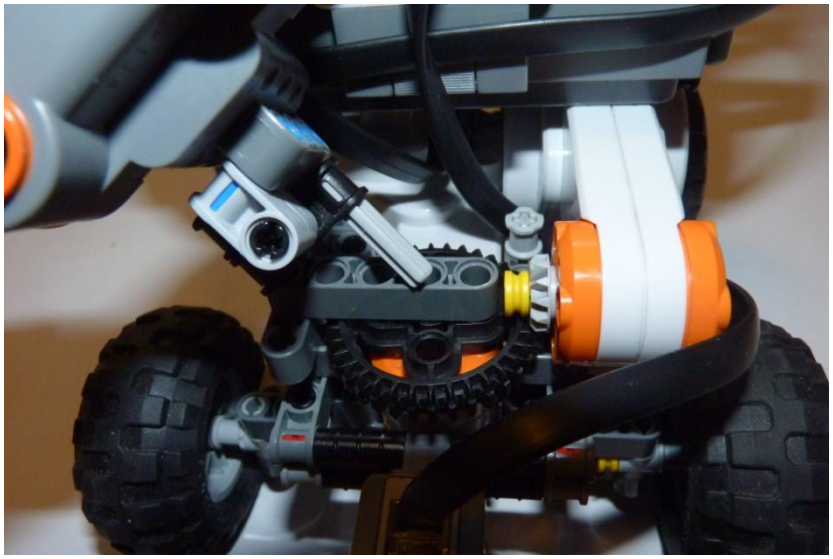
Konstrukční řešení robota řešícího tuto úlohu je velmi podobné tomu předcházejícímu (parkující robot). Bylo provedeno několik změn, hlavně v použití třetího motoru a odlišné umístění ultrazvukového senzoru. Celkový náhled na robota je na obrázku 4.12.

Obrázek 4.12 Celkový náhled na robota

- ***Otočný ultrazvukový senzor***

Na obrázku 4.13 je vidět detailní umístění ultrazvukového senzoru v přední části robota. Protože sledovaný automobil se nemusí nacházet jen před robotem, ale může i zatáčet, bylo potřeba použít i třetí motor, aby byl zajištěn širší zorný úhel robota. Samotný senzor sice má určitý zorný úhel (cca 20°), ale nyní byl úhel rozšířen až na cca 180°, což je dostatečné pro sledování automobilu.

Motor, který otáčí senzorem, jsem umístil na bok robota tak, aby nepřekážel přední ani zadní nápravě. Samotný senzor se otáčí podle stejné osy jako je osa otáčení motoru ovládajícího přední nápravu. Dokonce se ho i dotýká, ale není s tímto motorem nijak propojen (s jeho otáčivou částí).



Obrázek 4.13 Detail připojení ultrazvukového senzoru

4.4.6 Programové řešení robota

I tuto úlohu lze úspěšně rozdělit na několik dílčích, téměř nezávislých, podúloh. Každá podúloha má jasně definované vstupní a výstupní veličiny:

- Sledování čáry
  - vstup: Světelný senzor
  - výstup: Úhel natočení přední nápravy
- Sledování automobilu
  - vstup: Ultrazvukový senzor
  - výstup: Úhel natočení ultrazvukového senzoru
- Udržování konstantní vzdálenosti
  - vstup: Ultrazvukový senzor
  - výstup: Rychlost otáčení zadní hnané nápravy

- ***Sledování čáry***

Řešení této podúlohy je identické s řešením stejné podúlohy v případě automatického parkování v kapitole 4.3.5. Nebudu jej zde proto znovu uvádět.

- ***Sledování automobilu***

Ke sledování automobilu je potřeba rozšířit schopnosti robota sledujícího čáru. Pro ten účel jsem vytvořil třídu *FollowingRobot*, která je potomkem třídy *SteeringLineFollower*, a dědí tak od ní všechny funkce spojené s pohybem robota a sledováním čáry. Zbývá jen dopsat funkce ovládající třetí motor a ultrazvukový senzor.

Aby byl robot schopný sledovat automobil, je potřeba zareagovat, pokud zmizí ze zorného úhlu, a opět jej rychle najít. To provádí funkce *search(int lastDirection)*. Tato funkce běží v nekonečné smyčce po celou dobu běhu robota (do stisknutí tlačítka ESCAPE na NXT kostce). V každém cyklu této funkce se načtou data z ultrazvukového senzoru, která jsou filtrována (viz níže). Pokud je automobil nalezen ve vzdálenosti menší než 50 cm, je aktualizována proměnná *lastDistance*, která v sobě neustále nese informaci o poslední známé vzdálenosti automobilu. Pokud není ve vzdálenosti do 50 cm žádný objekt, automobil nejspíše odbočil a je potřeba ho nalézt.

Robot (jeho hlava tvořená ultrazvukovým senzorem) tedy začne vykonávat prohledávací proceduru:

- podívá se směrem, ve kterém byl při předchozím hledání úspěšný
- pokud nic nenajde, otočí se o stejný úhel, ale opačným směrem
- pokud opět nic nenajde, zvětší úhel a předchozí dva kroky opakuje
- pokud robot prohledal určitý úhel, nastaví se do původní pozice a celou proceduru opakuje

Tímto způsobem robot postupně prohledává prostor kolem původní pozice, dokud automobil nenajde. Výhodou tohoto řešení je, že když se automobil zatáčí, robot ho sleduje tím směrem, kterým zatáčí, má ho tak stále na dohled a je schopen regulovat jeho vzdálenost.

Vzhledem k tomu, že ultrazvukový senzor občas nenalezne žádnou odezvu (tehdy vrací hodnotu 255), i když je automobil jasně v jeho zorném poli, je zapotřebí takovéto výkyvy filtrovat, jinak by robot začal svou, výše uvedenou, prohledávací proceduru a automobil by akorát ztratil z dohledu. Pro filtrování jsem si vytvořil jednoduchou funkci, která ignoruje vždy první dvě získané vzdálenosti, které mají hodnotu právě 255.

```
private int getDistanceFiltered() {
```

```
    int badCount = 0;
    int dist = sonar.getDistance();
    while (dist == 255 && badCount < 2) {
        badCount++;
        dist = sonar.getDistance();
    }
    return dist;
}
```

- ***Udržování konstantní vzdálenosti***

Splnění této podúlohy zajišťuje PI regulátor, který podle poslední známé vzdálenosti od automobilu (která je uložena v proměnné *lastDistance*) určuje rychlost robota. Snaží se regulovat vzdálenost k hodnotě 30 cm. Regulátor běží s periodou 100 ms. Rychlejší perioda regulátoru nemá moc velký smysl, protože přibližně 100 ms rovněž trvá získání vzdálenosti pomocí ultrazvukového senzoru.

4.4.7 Výsledek

Výsledkem řešení této úlohy je robot, který dokáže celkem efektivně sledovat před ním jedoucí automobil jak na rovných úsecích tak i v zatáčkách a při různých rychlostech. Ukázka jízdy tohoto robotu je umístěna k shlédnutí na http://www.youtube.com/watch?v=Fe--cfQM208&NR=1 a v elektronické příloze ve složce Videos.

Největší úskalí této úlohy již několikrát zmiňovaný ultrazvukový senzor, který není dobře přizpůsoben k prohledávání okolí. To se dá částečně kompenzovat vhodnou volbou sledovaného automobilu.

5 Webové stránky

Součástí této práce jsou i webové stránky, týkající se dvou navržených soutěžních úloh, umístěné na webu Katedry řízení.

Vzhled stránek je převzat ze stávajících webových stránek předmětu [6] (jsou použity i stejné kaskádové styly [7]), protože je pouze doplňují.

Jejich obsah je výňatkem z kapitol 4.3.1 až 4.3.3 a 4.4.1 až 4.4.3 této práce.

Jedná se o soubory *index.html* a *uloha2.html*, které jsou k dispozici v elektronické příloze.

6 Závěr

V této bakalářské práci byl věnován větší prostor v jednotlivých kapitolách nejrůznějším problémům, se kterými bylo třeba se potýkat. Ve druhé kapitole se jedná například o nemalé potíže při používání ultrazvukového senzoru, v kapitole třetí pak o možnostech spolehlivého řízení polohy motoru v různých prostředích. Ve třetí kapitole byl rovněž obohacen stávající návod na instalaci prostředí leJOS NXJ o instalaci nové verze.

Dále byly navrženy dvě nové soutěžní úlohy včetně specifikace jejich pravidel a hodnocení. Základem obou úloh je snaha přiblížit se návrhu skutečných automobilů. První úloha má název Automatické parkování a druhá má název Sledování automobilu. Řešení obou úloh je založeno na konstrukci podvozku s přední otočnou nápravou a sledování čáry, která zde symbolizuje jízdní pruh.

Obě úlohy byly vyřešeny pouze v programovacím prostředí leJOS NXJ, protože ostatní prostředí (NXT-G a NXC) neumožňovala plynulé řízení otáčení přední nápravy.

K těmto úlohám byly také vytvořeny webové stránky, které doplňují ty již stávající. Byly proto vytvářeny s použitím stejných stylů.

7 Literatura

Internetové zdroje:

[1] LEGO.com MINDSTORMS : Home [online]. c2010 [cit. 2011-06-02]. Dostupné z WWW: http://mindstorms.lego.com/en-us/default.aspx

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

[3] LeJOS, Java for Lego Mindstorms [online]. 2006 [cit. 2011-06-03]. Introduction to leJOS NXJ. Dostupné z WWW: http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/Intro.htm

[4] LeJOS, Java for Lego Mindstorms [online]. 2006 [cit. 2011-06-03]. NXJ Downloads. Dostupné z WWW: http://lejos.sourceforge.net/nxj-downloads.php

[5] Welcome to NetBeans [online]. 2010 [cit. 2011-05-28]. Dostupné z WWW: http://netbeans.org/

[6] Roboti [online], Česke vysoke učeni technicke v Praze (2011-03-12), http://support.dce.felk.cvut.cz/roboti

[7] ROBOTI | _CVUT - Katedra řídící techniky [online]. 2009 [cit. 2011-05-25]. Roboti.css. Dostupné z WWW: http://support.dce.felk.cvut.cz/roboti/roboti.css

[8] ROBOTI | _CVUT - Katedra řídící techniky [online]. 2009 [cit. 2011-05-25]. Roboti head. Dostupné z WWW: http://support.dce.felk.cvut.cz/roboti/img/head cs.jpg

[9] Philo's Home Page [online]. 2007 [cit. 2011-05-29]. NXT Rmotor internals. Dostupné z WWW: http://www.philohome.com/nxtmotor/nxtmotor.htm

[10] HANSEN, John. NBC/NXC - NeXT Byte Codes and Not eXactly C [online]. 2007-10-10 [cit. 2011-06-02]. The NXC Guide. Dostupné z WWW: http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC Guide.pdf

[11] BENEDETTELLI, Danny. Programming Lego Robots using NXC [online]. 2007-10-10 [cit. 2011-06-02]. The NXC Guide. Dostupné z WWW: http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC Guide.pdf

[12] ROBOTC.net [online].Carnegie Mellon University Robotics Academy, 2009 [cit. 2011-05-03]. Download ROBOTC for LEGO MINDSTORMS. Dostupné z WWW: http://www.robotc.net/download/nxt/

Knižní zdroje:

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

[13] MARTINEC, Dan. Využití robota LEGO MINDSTORMS při výuce předmětu A3B99RO Roboti,
Praha 2010, bakalářská práce (Bc.). České vysoké učení technické v Praze. Fakulta řídící techniky

8 Seznam obrázků

Obrázek 2.1 LEGO Mindstorms RCX 9
Obrázek 2.2 LEGO Mindstorms NXT 9
Obrázek 2.3 NXT inteligentní kostka v černém provedení (limitovaná edice) 10
Obrázek 2.4 Li-ion akumulátor 11
Obrázek 2.5 NXT Servomotor 12
Obrázek 2.6 Vnitřek motoru 12
Obrázek 2.7 Dotykový senzor 13
Obrázek 2.8 Světelný senzor 13
Obrázek 2.9 Světelný senzor 15
Obrázek 2.10 Zvukový senzor 15
Obrázek 3.1 Grafické prostředí NXT-G 17
Obrázek 3.2 Programovací prostředí Bricx Comand Center 17
Obrázek 3.3 Vývojové prostředí Netbeans 22
Obrázek 4.1 Možná varianta soutěžního plánu 28
Obrázek 4.2 Celkový náhled robota 30
Obrázek 4.3 Diferenciál 31
Obrázek 4.4 Přední náprava 31
Obrázek 4.5 Ackermannova geometrie řízení 32
Obrázek 4.6 Umístění senzorů 33
Obrázek 4.7 Graf závislosti konstant PID regulátoru na rychlosti 36
Obrázek 4.8 Vývojový diagram podúlohy "Sledování zaparkovaných aut" 38
Obrázek 4.9 Vývojový diagram parkovací sekvence 40
Obrázek 4.10 Jedna z variant soutěžního plánu 42
Obrázek 4.11 Sledované vozidlo s odrazovou mřížkou 45
Obrázek 4.12 Celkový náhled na robota 45
Obrázek 4.13 Detail připojení ultrazvukového senzoru 46

9 Seznam tabulek

Tabulka 3.1 Nejdůležitější makra 18
Tabulka 3.2 Funkce pro obsluhu motorů 19
Tabulka 3.3 Funkce pro obsluhu senzorů 20
Tabulka 3.4 Hlavní metody třídy NXTRegulatedMotor 23
Tabulka 3.5 Základní funkce pro obsluhu senzorů 23

10 Příloha

Obsah CD

Součástí této práce je elektronická příloha, na které jsou všechny fotografie detailů robota, videozáznamy ukázkových jízd a zdrojové kódy programových řešení obou úloh.

Na disku jsou v kořenovém adresáři následující podadresáře:

- \Source – zde jsou uloženy zdrojové kódy
- \Videos – zde jsou umístěny videozáznamy
- \www – zde jsou webové stránky k navrženým úlohám