SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-123283-76935

Programovanie NXT Robotov

Bakalárska práca

2017 **Zuzana Dudáková**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta chemickej a potravinárskej technológie

Evidenčné číslo: FCHPT-123283-76935

Programovanie NXT Robotov

Bakalárska práca

Študijný program: automatizácia, informatizácia a manažment v chémii a potravinárstve
Študijný odbor: 5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky
Vedúci záverečnej práce: Ing. Juraj Holaza
Konzultant: Ing. Richard Valo, PhD.

Bratislava 2017 **Zuzana Dudáková**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2016/2017
Evidenčné číslo: FCHPT-123283-76935

STU
FCHPT

ZADANIE BAKALÁRSKEJ PRÁCE

Študentka:	**Zuzana Dudáková**
ID študenta:	76935
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Vedúci práce:	Ing. Juraj Holaza
Konzultant:	Ing. Richard Valo, PhD.
Miesto vypracovania:	Oddelenie informatizácie a riadenia procesov

Názov práce: **Programovanie NXT Robotov**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Projekt je zameraný na osvojenie si programovacích zručností pri tvorbe riadiacich algoritmov robotov LEGO-Mindstorms NXT.

Čiastočné ciele práce sú:
- Syntéza riadiacich algoritmov pre problém sledovania žiadanej trajektórie.
- Validácia navrhnutých algoritmov prostredníctvom simulácie riadenia v prostredí NXC editora.
- Experimentálna implementácia navrhnutých algoritmov.
- Tvorba podporných edukačných materiálov.

Rozsah práce: 30

Zoznam odbornej literatúry:

1. HANSEN. J. NXC Programmer's Guide, [online 2011-10-17] http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf
2. Sourceforge.net, NXCeditor,[online 2017-02-13] http://nxceditor.sourceforge.net/index.html

Riešenie zadania práce od: 13. 02. 2017

Dátum odovzdania práce: 14. 05. 2017

Zuzana Dudáková
študentka

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Poďakovanie

Chcela by som sa poďakovať školiteľovi Ing. Jurajovi Holazovi a konzultantovi Ing. Richardovi Valovi, PhD. za ich odborné vedenie, metodickú pomoc a cenné rady, ktoré mi poskytli pri vypracovaní bakalárskej práce.
Za veľkú pomoc a ochotu ďakujem môjmu bratovi Matejovi, z ktorého skúseností a poznatkov som čerpala.

Abstrakt

Práca sa zaoberá programovaním NXT lego robotov používaním nxcEditora, ktorý predstavuje prostredie, v ktorom sa pomocou programovacieho jazyka NXC (Not eXactly C) tieto roboty programujú. V teoretickej časti je spracované využitie stavebnice Lego Mindstroms NXT a popis jednotlivých mechanických komponentov sústavy Lego Mindstorms, a to senzorov a elektromotorov. V druhej časti práce je uvedený popis inštalácie nxcEditora spolu s nxcSimulátorom pod operačným systémom Linux.

Pravidlá písania algoritmov pomocou NXC jazyka , funkcie a štruktúra programu sú uvedené vo štvrtej kapitole a v piatej kapitole je nasledovne predstavená praktická časť práce, a to návrh a bližšie vysvetlenie NXC algoritmov pre tri druhy robotov sledujúcich čiernu čiaru. V piatej kapitole je tiež vysvetlený princíp proporcionálno-integračno-derivačného regulátora (PID).

Súčasťou práce sú videoukážky zverejnené na youtube [A, B, C, D], tutoriál podrobnej inštalácie nxcEditora, ktorý je k zhliadnutiu na Optiblogu [E] a algoritmy robotov sledujúcich čiernu čiaru.

Kľúčové slová: NXT; NXC; Lego Mindstorms; nxcEditor; nxcSimulator; PID.

Abstract

The thesis deals with the programming of NXT lego robots using nxcEditor, which represents the environment in which these robots are programmed using the NXC programming language (Not eX-actly C). In the theoretical part, the Lego Mindstroms NXT and the Lego Mindstorms mechanical components sensors and electric motors, are described. The second part describes how to install nxcEditor along with the nxcSimulator under the Linux operating system.
The rules for writing algorithms using NXC language, functions, and program structures are discussed in the fourth chapter and in the fifth chapter, practical part of the work is presented, especially the design and detailed explanation of NXC algorithms for three types of black line tracking robots. The fifth chapter also explains the principle of proportional-integral-derivative regulator (PID).
Part of the work are videos published on the internet [A, B, C, D], a detailed installation guide for the nxcEditor, which is available on Optiblog [E] and line following robots algorithms.

Key words: NXT; NXC; Lego Mindstorms; nxcEditor; nxcSimulator; PID.

Obsah

1. Úvod

1.1 NXT

Lego Mindstorms NXT (Obr.1.1) je stavebnica lega vydaná firmou Lego, Inc v júli 2006. Táto stavebnica nahradila prvú generáciu Lego Mindstorms stavebnicu RCX, ktorá bola na trh uvedená v roku 1998. Poslednou generáciou Lego Mindstorms je momentálne EV3, ktorá bola predstavená v septembri 2013 [1].

Obr.1.1: Stavebnica Lego Mindstroms NXT.

Lego Mindstorms ponúka možnosti modelovania procesov prostredníctvom jednoduchých lego súčiastok. Pomocou senzorov a vytvoreného programu je možnosť riadiť reakcie robota alebo vykonávať automatické meranie, napríklad teploty a osvetlenia.

Stavebnica má veľké využitie vo výučbe už na základných školách. Pôvodne sa stavebnica používala ako vzdelávací nástroj vďaka partnerstvu medzi Lego a MIT (Massachusetts Institute of Technology) [1]. Študenti sa môžu hravou formou zoznámiť so základnými súčiastkami, pomocou ktorých si poskladajú robota, čo rozvíja ich manuálne zručnosti. Algoritmické a analytické zručnosti si študenti osvoja vytvorením programu, ktorý je nutné prispôsobiť konštrukcii robota. Avšak najväčšou výhodou je, že programy si študenti overia v praxi s hmatateľným výsledkom, vyskúšajú si prácu v tíme a prostredníctvom súťaží si tiež osvoja zdravú súťaživosť.

Študenti sa na základe svojej vekovej kategórie môžu zúčastňovať nasledujúcich súťaží: The FIRST LEGO League (FLL), Junior *FIRST* LEGO League (Jr.FLL), RoboCup Junior (RCJ), World Robot Olympiad (WRO) [2].

Na súťažiach sa študenti snažia navrhnúť čo najlepší algoritmus, a tak uspieť v širokej konkurencii. Viaceré vývojové firmy vysielajú na spomenuté súťaže svojich zamestnancov, ktorí hľadajú nových nádejných programátorov pre svoje firmy.

Čo sa týka samotných súčiastok, hlavným prvkom stavebnice je 32-bitová riadiaca jednotka. Kvôli obdĺžnikovému tvaru býva v literatúre označovaná ako programovateľná kocka (Obr.1.2). Kocka je základom každého skonštruovaného robota a obsahuje 3 výstupné porty A, B a C, slúžiace na prepojenie s výstupnými zariadeniami a 4 vstupné porty, port 1, 2, 3 a 4 pre napájanie senzorov.

Obr.1.2: Kocka NXT.

Celá stavebnica obsahuje 577 dielov zahrňujúc 3 servomotory (Obr.1.3) a štyri senzory.
Servomotory predstavujú akčné jednotky v LEGO stavebniciach. Všetky akčné jednotky je možné pripojiť k portom A, B a C. Servomotory slúžia k rozpohybovaniu robota na základe prijímaných inštrukcií z programovateľnej kocky. Majú zabudovaný rotačný senzor pre meranie rýchlosti, vzdialenosti a spätnú väzbu k NXT. Riadenie motora je možné prevádzať s presnosťou otočenia na jeden stupeň. Na rovnakú presnosť otáčania je možné naprogramovať aj viac motorov.

Obr.1.3: Interaktívny servomotor.

Senzory slúžia k získavaniu údajov z okolia robota a môžu byť využité pri riadení robota. Získané hodnoty je možné uložiť, prípadne preniesť do počítača pre ich ďalšie spracovanie. Senzory sa pripájajú k portom 1 až 4 na NXT kocke pomocou káblov, ktoré sú súčasťou stavebnice. Lego Mindstorms obsahuje 4 druhy senzorov.

Dotykový senzor

Dotykový senzor (Obr.1.4) pracuje ako binárne tlačidlo vracajúce logickú hodnotu True/False (pravda/nepravda). Tento senzor sa môže použiť na ovládanie alebo spustenie nahratého programu, môže slúžiť ako senzor pre náraz do prekážky alebo ním dokážeme zisťovať prítomnosť objektov. Na základe toho, aká akcia sa vykoná, priraďujeme senzoru nasledovné logické hodnoty: Pri stlačení snímača je výstup „1“ a v opačnom prípade „0“ [3].

Obr.1.4: Dotykový senzor.

Zvukový senzor

Zvukový senzor (Obr.1.5) sníma úroveň hlasitosti, t.j. meria úroveň intenzity zvuku v dB a dBA. (Frekvencie v dBA predstavujú percentuálne vyjadrenie hlasitosti zvukov počuteľných ľudským uchom). Zvukový senzor meria všetky zvuky s rovnakou citlivosťou, to znamená, že senzor dokáže zaznamenať aj frekvencie ľudským uchom nepočuteľné.

Maximálna intenzita zvuku, ktorú je možné odmerať je 90 dB. Úroveň hlasitosti meria senzor v %, pričom 1% zodpovedá 0,9 dB [3].

Obr. 1.5: Zvukový senzor.

Svetelný senzor

Svetelný senzor (Obr.1.6) nedokáže rozpoznávať farby, umožňuje zistiť iba intenzitu osvetlenia, to znamená, že vidí iba v odtieňoch šedej (Obr.1.7) [3]. Farbu objektov teda rozoznáva pomocou zabudovaného zdroja svetla podľa intenzity odrazeného svetla. Intenzita svetelného žiarenia i farebných odtieňov je načítaná v percentách. Čím menšie číslo je namerané, tým je väčšia tma. Svetelný senzor sa často používa v úlohách, kde sa robot musí pohybovať po predom vyznačenej stope (sledovanie čiernej čiary).

Obr.1.6: Svetelný senzor.

Obr.1.7: Rozoznávanie farieb svetelným senzorom.

Ultrazvukový senzor

Ultrazvukový senzor (Obr.1.8) umožňuje robotovi vidieť a rozpoznať predmety, vyhýbať sa prekážkam, merať vzdialenosť a detekovať pohyb.

Využíva rovnaký princíp ako využívajú netopiere. Vzdialenosť meria na základe času návratu akustickej vlny o vysokej frekvencii odrazenej od prekážky naspäť k senzoru [3]. Veľké predmety s hrubými stenami sa pomerne ľahko rozoznávajú, naopak malé, zaoblené alebo predmety s mäkkým povrchom sú detekované menej spoľahlivo.

Senzor meria vzdialenosť buď v centimetroch alebo inchoch. Senzor dokáže merať vzdialenosť od 0 po 255 centimetrov s presnosťou +/- 3 cm [3].

Obr.1.8: Ultrazvukový senzor.

Komunikácia NXT kocky s počítačom prebieha káblom cez USB alebo pomocou Bluetooth. Pripojením USB kábla je možné nahrávať programy z počítača do NXT a tiež opačne zasielať dáta z robota do počítača. Alternatívou prepojenia NXT s počítačom je bezdrôtová Bluetooth komunikácia. NXT kocka je vybavená maticovým displejom (60 x 100 bodov) a 8 kHz reproduktorom. Na kocke sa nachádzajú štyri tlačidlá, pomocou ktorých je možné pripojené zariadenia testovať a veľmi obmedzene aj riadiť. Na napájanie sú vhodné AA tužkové monočlánky – 6 ks (9 V) alebo dobíjateľná batéria.

Programovať NXT je možné priamo, pomocou tlačidiel NXT a LCD displeja alebo nepriamo s využitím rôznych programovacích jazykov postavených na syntaxe jazyka C.

V tejto práci sa budem bližšie venovať programovaniu NXT robotov používaním NXC programovacieho jazyka.

1.2 NXC (Not eXactly C)

Jedným z programovacích jazykov najčastejšie používaných pri programovaní NXT robotov je NXC (Not eXactly C), ktorého syntax je podobná s programovacím jazykom C. Autorom NXC je John Hansen.

Čo sa týka programovania lego robotov, NXC sa používa v programovacom prostredí Bricx (BricxCC – Bricx Command Center), spustiteľnom na operačnom systéme Windows a tiež v nxcEditore, čo je verzia určená pre operačný systém Linux.

BricxCC a nxcEditor pracujú podobne ako textový editor, avšak majú niekoľko rozšírení [4]; možnosť exportovania programov do HTML, kompilovanie a sťahovanie programov, AutoLayout funkciu, ktorá v prípade programovanie pomocou jazyka C, C++ alebo NXC obnoví program, Code assist, ktorý vyhľadá zhodu s už existujúcimi príkazmi príslušného programovacieho jazyka, položku Help zahŕňajúcu zoznam NXC príkazov, ukážky niektorých algoritmov, sprievodcu príkazmi a informácie o editore.

2. nxcEditor

NxcEditor je voľne dostupný softvér, ktorého autorom je učiteľ na gymnáziu v nemeckom Rahden, Frank Knefel [5]. Tento editor bol vytvorený pre učebné účely na školách. Prostredníctvom nxcEditora je možné na operačnom systéme Linux programovať nielen NXT, ale aj BrickPi, Raspberry Pi, EV3 a Arduino roboty. NxcEditor podporuje viacero programovacích jazykov, ktoré sa používajú v závislosti od toho, akú robotickú platformu chcete použiť:

- Arduino dosky s USB portom (C/C++)
- EV3 s ev3dev OS (C/Python)
- NXT (NXC)
- BrickPi (C/Python)
- Raspberry Pi (C/Python) [6]

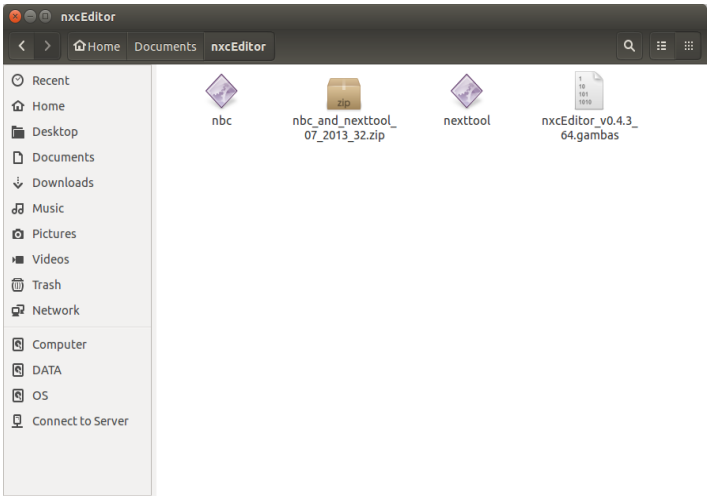
Program napísaný v jazyku C pre BrickPi môže byť naprogramovaný buď lokálne na Raspberry Pi alebo cez sieť použitím kompilátora na Raspberry Pi.
Program napísaný pomocou NXC je spustiteľný na reálnom robotovi, ale aj v nxcSimulátori, ktorý je súčasťou nxcEditora. Priamo je možné program spustiť na Linuxe, pri ostatných operačných systémoch je možné používať nxcEditor prostredníctvom virtuálneho počítača. V nasledujúcich kapitolách Vás prevedieme inštaláciou, možnosťami a využitím nxcEditora spolu s nxcSimulátorom, ktoré budú demonštrované na ukážke robota sledujúceho čiernu čiaru.

2.1 Inštalácia nxcEditora

nxcEditor je možné nainštalovať iba na operačnom systéme Linux. V nasledujúcich bodoch je podrobne opísaná inštalácia nxcEditora pre 32 a 64 bitové počítače a mala by byť pochopiteľná aj pre tých, ktorí s operačným systémom Linux zatiaľ nemajú skúsenosti.
Na inštaláciu nxcEditora budeme používať Terminál, poprípade Konzolu. Jedná sa o terminálovú aplikáciu, avšak s iným názvom, záleží od typu Linuxu. Terminál je aplikácia Linuxu, ktorá poskytuje textové rozhranie pre spúšťanie programov. V termináli je automaticky spustený program Shell, ktorý umožňuje spúšťanie zabudovaných príkazov (ako napr. `cd` pre zmenu adresára, `cp` pre kopírovanie súborov, atď..), užívateľských programov a skriptov.

Najnvošia verzia nxcEditora je na stiahnutie dostupná na stránke www.sourceforge.net/projects/nxceditor/files/nxcEditor. Z tej istej stránky je potrebné si stiahnuť aj

nbc_and_nexttool zip súbor. Treba dbať na stiahnutie vhodnej verzie pre počítač. (32 / 64 bit). Po stiahnutí zazipovaný súbor „nbc_and_nexttool“ rozbalíme (Obr.2.1).



Obr. 2.1: Stiahnuté a rozbalené súbory.

Po stiahnutí všetkých súborov budeme ďalej pracovať v termináli zadávaním príkazov:
Väčšine príkazov musí predchádzať príkaz `sudo`, ak pracujete s adresármi alebo súbormi, ktoré nie sú vo vašom vlastníctve (väčšinou všetky súbory mimo vášho domovského adreára). `Sudo` teda znamená spustenie príkazov so zvýšenými oprávneniami.
Aby sme zabezpečili aktualizovanie softvéru (Obr.2.2), začneme príkazom:

```
sudo add-apt-repository ppa:gambas-team/gambas3
```

„PPA“ (personal package archives) je softvér, ktorý nie je v Linuxe pôvodne zahrnutý. Väčšinou sa tieto repozitáre sústreďujú na jeden program, alebo môže zahŕňať aj viac programov. PPA zabezpečuje aktualizovanie softvéru oveľa rýchlejšie ako Linux a používa sa v príkaze `add-apt-repository ppa.`

Obr. 2.2: Aktualizovanie softvéru.

Nasledujúcim príkazom aktualizujeme balíčky Linuxu (Obr. 2.3):

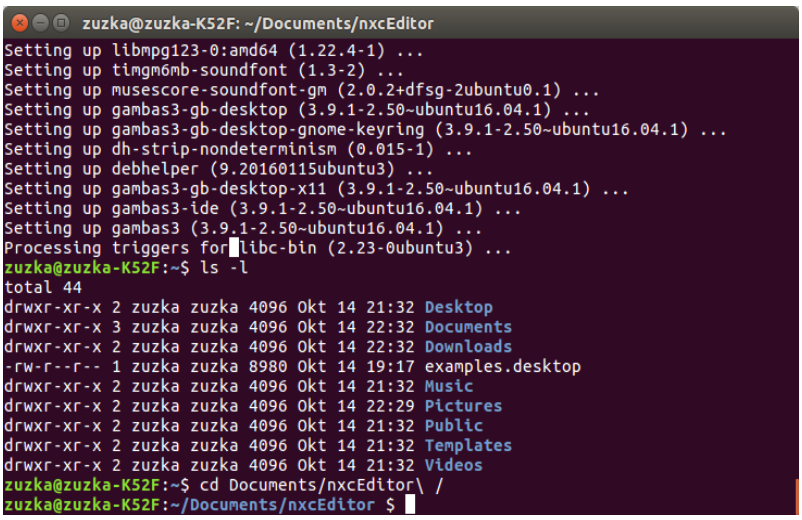
```
sudo apt-get update
```

Obr. 2.3: Aktualizovanie balíčkov Linuxu.

Po aktualizácii systému treba nainštalovať Gambas (Obr. 2.4), čo je voľne dostupné grafické vývojové prostredie. Aplikácie sa v Linuxe inštalujú príkazom `sudo apt-get install.`

```
sudo apt-get install gambas3
```

Ďalej musíme nbc a nexttool súbory skopírovať do „ /usr/local/bin“ priečinka, to znamená , že sa v termináli navigujeme použitím `cd` príkazu do priečinka, kde sme rozbalili nbc a nexttool súbory (pre zistenie aktuálneho adresáru sa používa príkaz `ls -l`) (Obr.2.4):

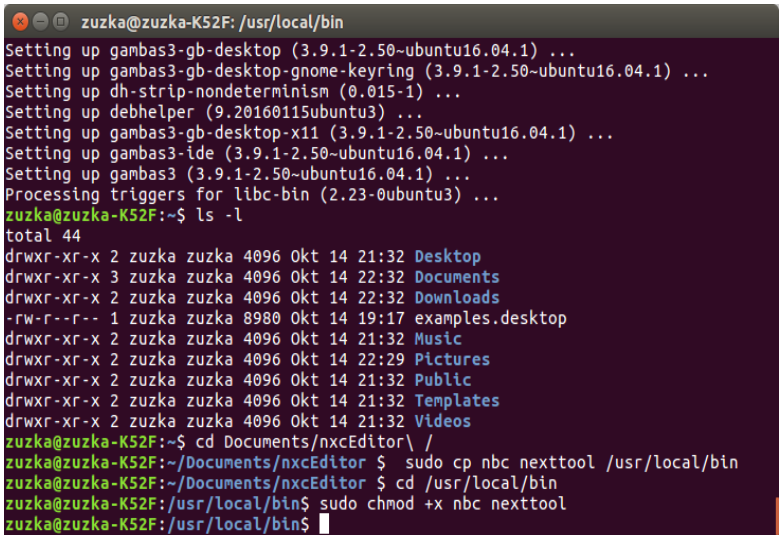


Obr. 2.4: Inštalácia grafického prostredia Gambas a použitie príkazu `ls -l`.

Na skopírovanie nbc a nexttool použijeme príkaz `sudo cp nbc nexttool /usr/local/bin` (Obr. 2.5). Teraz sa dostaneme do '/usr/local/bin' priečinka : `cd /usr/local/bin` a zmeníme povolenia súborov (Obr. 2.5).

Prístupové práva súborov meníme príklazom „chmod". Prepínač "-x" v príkaze `chmod -x` znamená, že súbor je povolené spustiť.

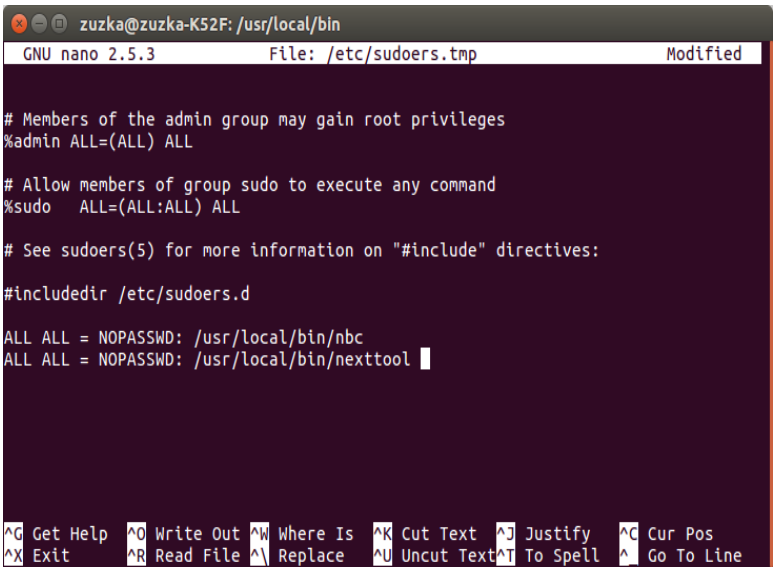
```
sudo chmod +x nbc nexttool
```



Obr. 2.5: Skopírovanie a zmena prístupových práv súborov.

Ďalej pokračujeme príkazom `sudo visudo`**.** Použitím príkazu `visudo` môžeme meniť „sudo súbory". a na spodok okna terminálu pridáme:

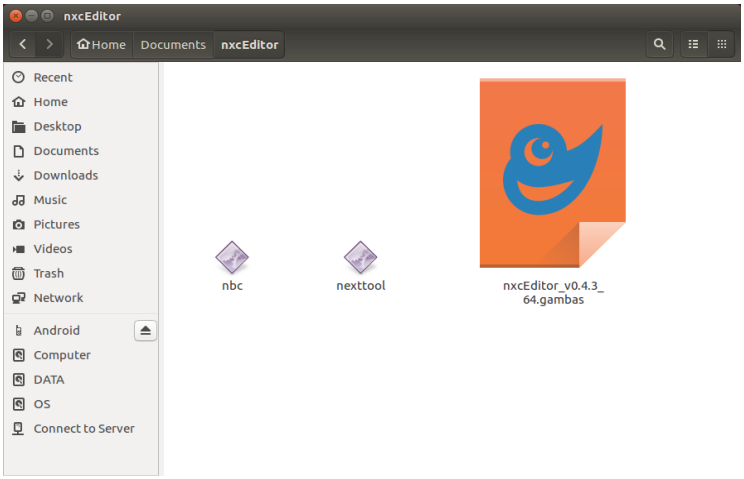
```
ALL ALL = NOPASSWD: /usr/local/bin/nbc
ALL ALL = NOPASSWD: /usr/local/bin/nexttool
```
(Obr. 2.6)



Obr. 2.6: Použitie príkazu „sudo visudo“.

Stlačíme ‘ctrl + x’ pre vystúpenie z okna a ‘y’ pre uloženie a nastavíme sa do priečinka, kde je stiahnutý nxcEditor a v termináli napíšeme : `sudo chmod +x nxcEditor_v0.4.3_64.gambas`, pričom použijeme presne ten istý názov nxcEditora, ktorý sme stiahli.

NxcEditor je po použití predchádzajúcich príkazov nainštalovaný (Obr. 2.7).



Obr. 2.7: Ukážka úspešne nainštalovaného nxcEditora.

Pozn.: Pri používaní inej distribúcie Linuxu ako Ubuntu alebo Mint je potrebné skopírovať „70-nxt.rules“ do priečinka „/etc/udev/rules.d“ : `cp 70-nxt.rules /etc/udev/rules.d`.

K úspešnému nainštalovaniu programu nxcEditor je teda potrebné vykonať sekvenicu príkazov v nasledujúcom poradí:

1) Stiahnutie programov
2) Zabezpečenie aktualizície softvéru: `sudo add-apt-repository ppa:gambas-team/gambas3`
3) Aktualizácia softvéru: `sudo apt-get update`
4) Inštalácia Gambas: `sudo apt-get install gambas3`
5) Nastavenie sa do priečinku, v ktorom sa nachádzajú stiahnuté súbory použitím príkazu „cd“ .
6) Skopírovanie nbc nexttool súborov do „/usr/localbin“ priečinka: `sudo cp nbc nexttool /usr/local/bin`
7) Nastavenie sa do „/usr/localbin“ priečinka: `cd /usr/local/bin`
8) Zmena prístupových práv súborov nbc a nexttool tak, aby boli spustiteľné: `sudo chmod +x nbc nexttool`
   Pre zmenu „sudo súborov“: `sudo visudo` a na spodok strany pridáme:
   ```
   ALL ALL = NOPASSWD: /usr/local/bin/nbc
   ALL ALL = NOPASSWD: /usr/local/bin/nexttool
   ```
9) Stlačíme „ctrl- x“ pre vystúpenie z okna a „y“ pre uložene
10) Nastavíme sa do priečinka, kde sa nachádza stiahnutý nxcEditor použitím „cd“ príkazu.
11) Príkazom `sudo chmod +x nxcEditor_v0.4.3_64.gambas` zabezpečíme spustiteľnosť nxcEditora.
12) Skopírovanie „70-nxt.rules“ do priečinka „/etc/udev/rules.d“ : `cp 70-nxt.rules /etc/udev/rules.d` (tento príkaz si nevyžadujú Ubuntu a Mint).

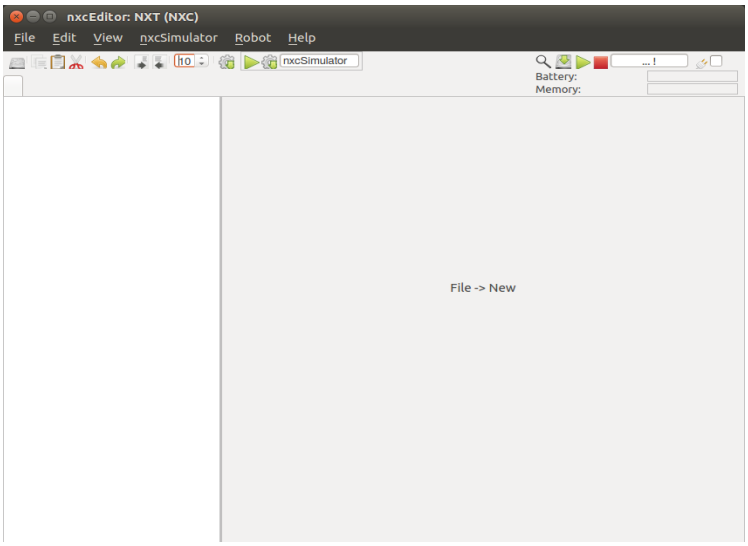
2.2 Rozhranie nxcEditora

Po spustení nxcEditora si všimnete, že používateľské rozhranie (Obr. 2.8), ktorého jednotlivé časti editora sú očíslované na obrázku nižšie (Obr. 2.9), vyzerá podobne ako štandardný textový editor so zvyčajným menu (1) pre otvorenie a uloženie súborov, tlačenie súborov, upravovanie súborov a mnoho iných. Editor však má i niektoré špeciálne položky menu, pre kompiláciu a sťahovanie programov do robota, pre získanie informácií od robota a tiež položku “Help” s možnosťou zmeny jazyka, s vopred naprogramovanými programami, užitočnými najmä pre začiatočníkov a tiež so zoznamom NXC príkazov a informáciami o editore.

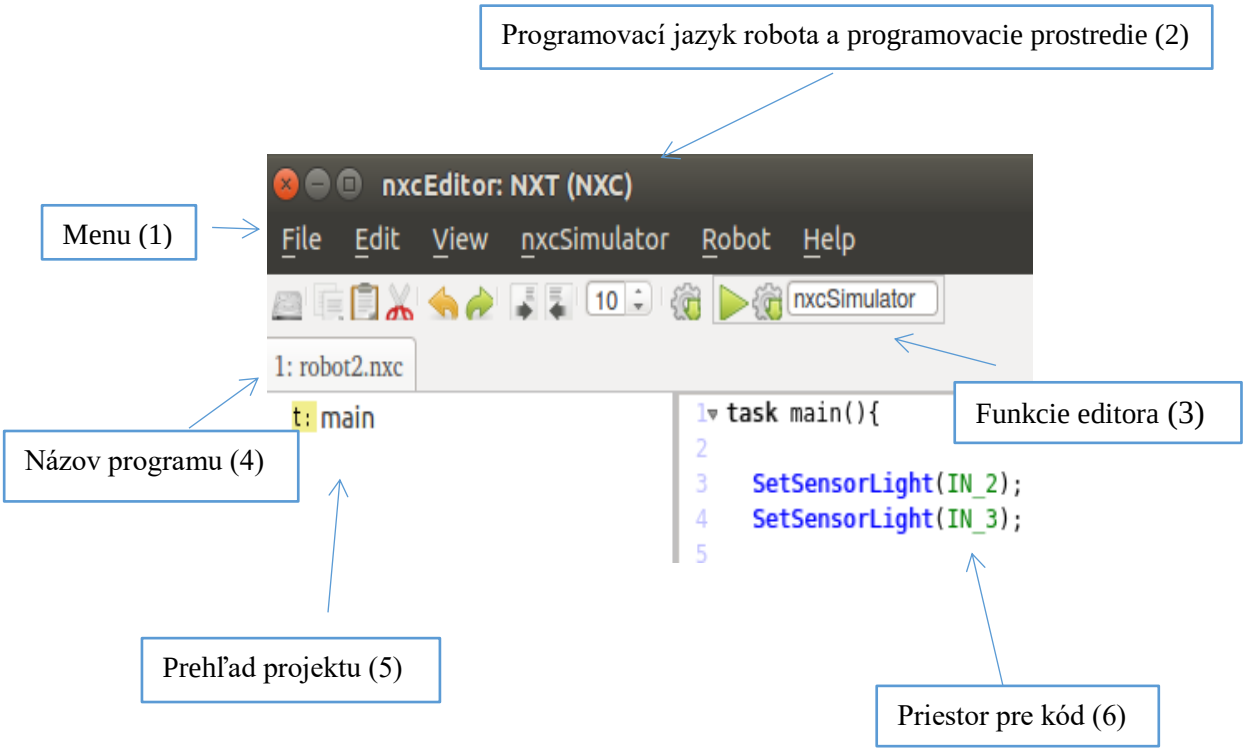
Aby bolo na prvý pohľad jasné, ktoré rozhranie na programovanie robota sa používa a ktorý programovací jazyk bol na programovanie vybraný, zodpovedajúce informácie sú zobrazené v časti

(2). Niektoré funkcie editora (3) môžu byť spustené priamo kliknutím na zodpovedajúcu ikonu, tiež je možné nastaviť veľkosť písma. Prehľad projektu (5) sa vytvára automaticky a samotný program (4) sa píše do textového poľa (6), pričom jednotlivé riadky kódu sa automaticky číslujú, zdrojový kód je farebne zvýraznený a automaticky odsadzovaný.

Po prvom zapnutí nxcEditora je defaultne nastavené NXT ako rozhranie robota. Iný typ robota môže byť vybraný v menu „Robot“ .



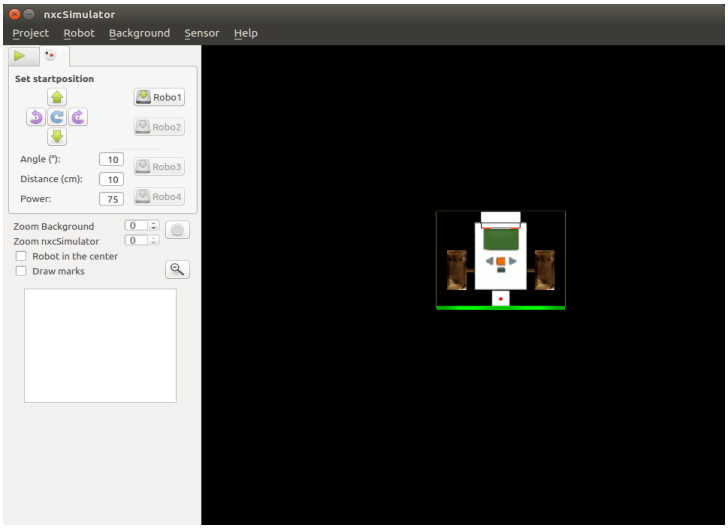
Obr. 2.8: nxcEditor po spustení.



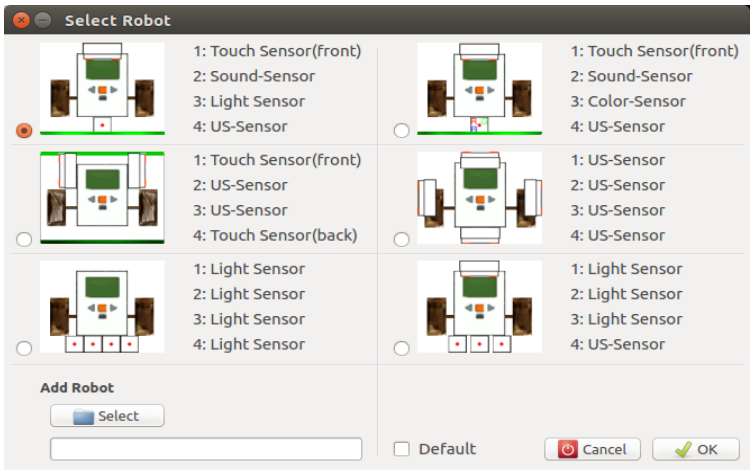
Obr. 2.9: Časti nxcEditora.

3. nxcSimulátor

nxcSimulátor je možné použiť iba v prípade programovania v jazyku NXC. Po spustení programu resp. samotného nxcSimulátora, je vždy defaultne nastavený jeden typ robota a pozadia (Obr. 3.1). V simulátori je na výber množstvo typov robotov s rôznymi rozmermi a senzormi (Obr. 3.2).

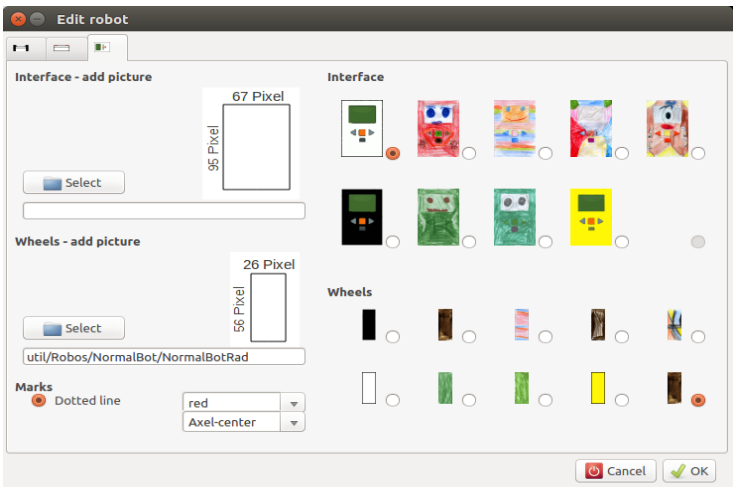


Obr. 3.1: nxcSimulátor po spustení.

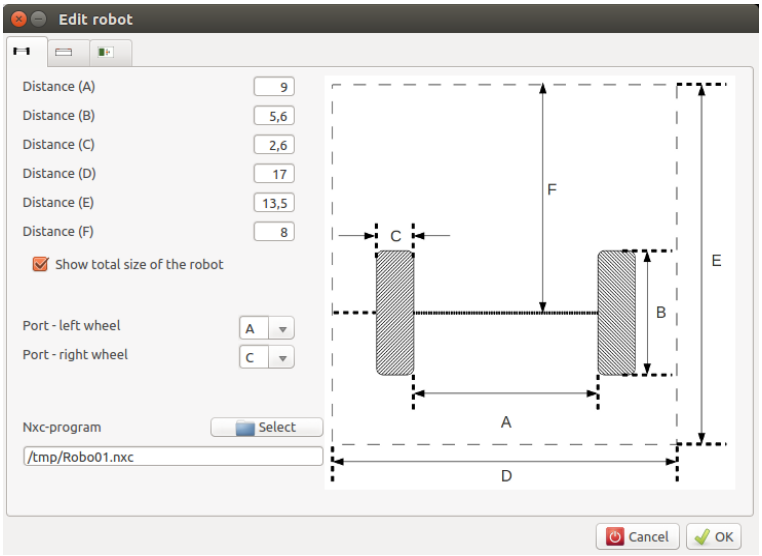


Obr. 3.2: Typy robotov v nxcSimulátori.

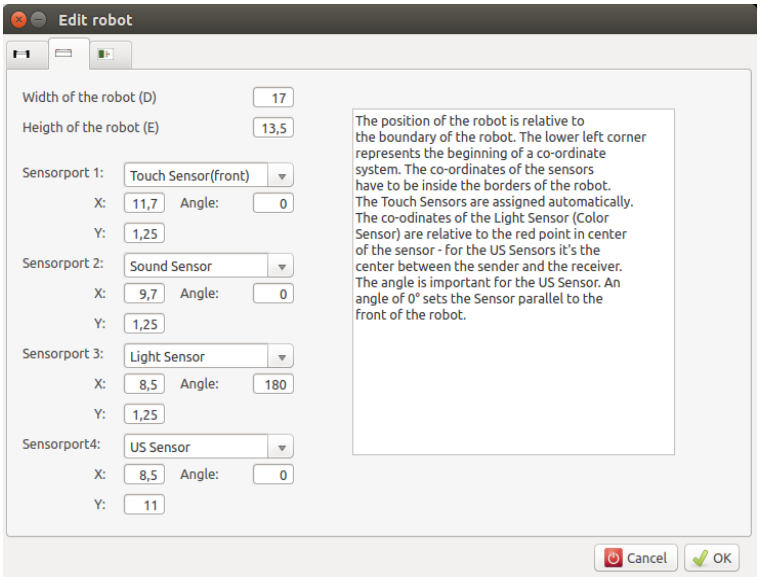
Samotného robota v simulátori je možné aj meniť, jednak jeho vzhľad, rozmery, množstvo a typ senzorov, ale aj pozíciu, tiež je možné nechať vykreslovať čiaru kopírujúcu robotove pohyby („draw marks“).



Obr. 3.3: Možnosti vzhľadu robota v nxcSimulátori.

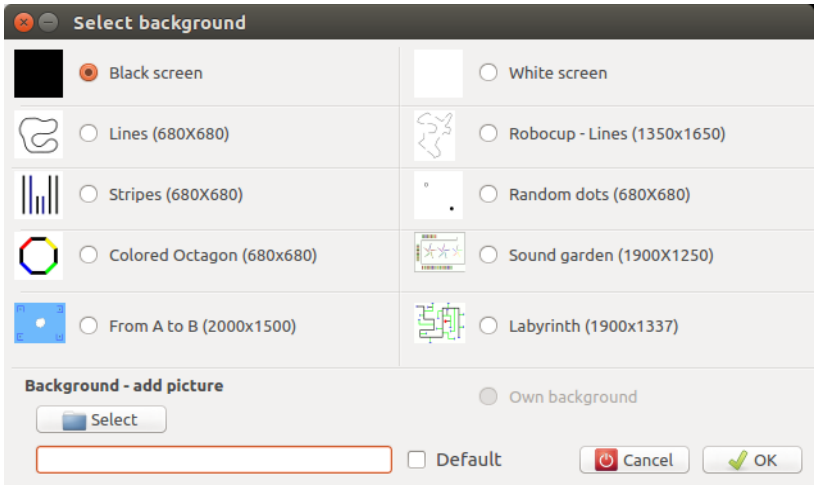


Obr. 3.4: Jednotlivé vzdialenostné parametre a porty robota.



Obr. 3.5: Možnosti menenia senzorov.

Vytvoreného robota je možné uložiť a pracovať s ním ďalej. Taktiež je možné si vybrať, ale aj vytvoriť vlastné pozadie, na ktorom chceme robota simulovať. Na výber sú pozadia na Obr. 3.6.



Obr. 3.6: Pozadia nxcSimulátora.

Program je možné kedykoľvek zastaviť, vykonať zmeny či už v kóde, v parametroch robota alebo na jeho pozícii a následne ho znovu spustiť.

4. NXC programy

V tejto kapitole si vysvetlíme základné pravidlá, funkcie a príkazy (Tab. 4.4) jazyka NXC, pomocou ktorých dokážeme ovládať základné senzory a akčné členy stavebnice Lego Mindstorms, ktoré sme si predstavili v kapitole 1.1 NXT.

4.1 Lexikálne pravidlá

Jedná sa o súhrn pravidiel, ktoré popisujú to, ako má vyzerať štruktúra zdrojového kódu- ako písať komentáre, manipulovať s medzerami alebo aké sú povolené znaky pre identifikátory.

Komentáre

Existujú dve podoby komentárov:

NXC aj jazyk C používajú pre jednoriadkový komentár „//“ (Obr.4.2). V jazyku C môžeme vytvoriť viacriadkový komentár, ktorý začína „/*“ a končí „*/“. Týmto komentárom vytvárame viacriadkové komentáre, ktoré však nemôžu byť do seba vnorené (Obr.4.1) [7].

.

```
/* toto je napriklad komentar */

/* toto je
          dvojriadkovy komentar */
```

Obr.4.1: Viacriadkové komentáre používané v jazyku C.

```
// jednoriadkovy komentar
```

Obr.4.2: Jednoriadkové komentáre používané v jazyku C a v NXC.

Komentáre slúžia ako pomôcka pre autora resp. pre niekoho, kto kód bude čítať, prekladač komentáre ignoruje, nevykonajú sa.

Prázdne miesto

Pod prázdnym miestom rozumieme všetky medzery, tabulátory a nové riadky. Pridanie alebo odobranie medzery nemá žiadny vplyv na význam programu, pokiaľ sú znaky dobre rozlíšiteľné.

Na Obr. 4.3 je uvedená ukážka kódu, ktorý má rovnaký význam vo všetkých troch prípadoch:

```
x=2;
x=   2;
x=     2;
```

Obr. 4.3: Používanie medzier v jazyku NXC.

Existujú však operátory s viacerými znakmi, medzi ktorými sa medzera objaviť nesmie. Na Obr. 4.4 je v prvom riadku použitý operátor „>>“ pre bitový posun vpravo, v druhom riadku je ale vložená medzera za „> “, čo predstavuje chybný zápis [7].

```
//nastavi premennu x na cislo,ktore vznikne posunutim 1 o 4 bity doprava
x = 1 >> 4;
// chyba:
x = 1 > > 4;
```

Obr.4.4: Používanie medzier v jazyku NXC.

Textové reťazce

Konštanty vo forme textových reťazcov sú v NXC ohraničené dvojitými úvodzovkami (Obr.4.5). Tieto reťazce sú na pozadí automaticky konvertované do poľa bytov, kde posledný byte v poli je nulový. Posledná bytová nula sa obvykle označuje „null terminator“ [7].

```
//Na prvom riadku NXT kocky sa vypise: "testovanie"
TextOut(0,LCD_LINE1,"testovanie");
```

Obr.4.5: Ukážka syntaxe textových reťazcov v NXC.

Identifikátory a kľúčové slová

Identifikátory sa používajú ako mená pre premenné, úlohy, funkcie a podprogramy.

Identifikátor musí začínať veľkým alebo malým písmenom alebo podtržníkom „_“ . Nasledujúce znaky môžu byť čísla, podtržníky, písmená.

Pod kľúčovými slovami rozumieme vyhradené skupiny znakov v jazyku NXC , ktoré nemôžu byť použité ako identifikátory. Medzi kľúčové slová v NXC patria:

asm, bool, break, byte, case, char, const, continue, default, do, else, enum, false, float, for, goto, if, inline, int, long, mutex, priority, repeat, return, safecall, short, start, stop, string, struct, sub, switch, task, true, typedef, unsigned, until, void, while [7].

4.2 Štruktúra programu

Program v NXC tvoria kódové bloky a premenné. Existujú dva typy kódových premenných: úlohy a funkcie. Ich maximálny počet môže byť 256.
Každý kódový blok má jedinečné vlastnosti, ale štruktúra je spoločná.

Úlohy – Tasks
Úloha v NXC predstavuje jedno vlákno a je definovaná kľúčovým slovom „task“ . Syntax je uvedená na obrázku nižšie:

```
task (meno)
{
   //tu sa nachadza kod ulohy
   ...
   }
```

Obr.4.6: Syntax „úloh“.

Názvom úlohy môže byť akýkoľvek povolený identifikátor. Každý program musí mať úlohu nazvanú “main”. Main je úloha, ktorá je vykonávaná robotom. Úloha sa skladá z viacerých príkazov a všetky príkazy úlohy sú uzatvorené do zložených zátvoriek tak, aby bolo jasne vidno, že všetky patria do tejto úlohy. Každý príkaz končí bodkočiarkou, čím je evidentné, kde jeden príkaz končí a kde druhý začína [7].
Príkazy používané v súvislosti s úlohami sú uvedené v Tab. 4.4.

Funkcie
Niekedy je potrebné zlúčiť skupinu príkazov do jednej funkcie, ktorú bude kód podľa potreby volať. V NXC sú podporované funkcie so vstupnými argumentami a ich návratovými hodnotami, takže do funkcie môžeme zakaždým poslať iné vstupné argumenty, a tým sa funkcia bude zakaždým chovať inak. Pokiaľ funkcia nevracia žiadnu hodnotu, používa sa „void“ alebo „sub“ [7].

Syntax funkcie je nasledovná:

```
navratovy_typ menoFunkcie (zoznam argumentov)
{
   //telo funkcie
   ...
   }
   ...
```

Obr.4.7: Syntax funkcie.

Zoznam argumentov môže byť prázdny alebo môže obsahovať jeden, či viac argumentov, ktoré od seba oddeľujeme čiarkou“,“. Na nasledujúcich obrázkoch si ukážeme niektoré príklady zapisovania funkcií s rôznymi vstupmi a výstupmi.

```
int cislo () //navratova hodnota je typu int
{
   int x = 1;
   return x; // prikazom return urcujeme, ktoru premennu funkcia vrati
   ...
   }
   ...
// ak chceme funkciu zavolat, pouzijeme nasledujuci kod:
int a = cislo(); // do premennej a sa ulozi vystup funkcie
```

Obr.4.8: Príklad funkcie, ktorá nemá žiadny vstup a výstupom je premenná int.

```
//funkcia pre vypocet sucinu dvoch cisel:
int sucin (int x, int y) // dva vstupne argumenty typu int
return x * y; // funkcia vrati sucin dvoch cisel
```

Obr.4.9: Príklad funkcie ktorá má dva vstupné argumenty a výstupom je premenná typu int.

Premenné:

Premenné sú deklarované pomocou kľúčového slova premennej a jej mena. Deklarácia sa ukončuje bodkočiarkou „ ; “.

V Tab. 4.1 sa nachádzajú typy premenných a ich veľkosti.

Tab. 4.1: Typy a veľkosti jednotlivých premenných používaných v jazyku NXC.

Premenná	Veľkosť
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	špeciálna premenná
string	textové reťazce
struct	užívateľom definovaná štruktúra

Pozn.: unsigned = bez znamienka

Tvrdenia

V nasledujúcich tabuľkách sú operátory, ktoré modifikujú alebo porovnávajú hodnoty premenných [7].

Tab. 4.2: Operátory pre porovnávanie premenných.

Operátor	Funkcia
= =	je rovné
<	je menšie
>	je väčšie
< =	je menšie alebo rovné
>=	je väčšie alebo rovné
!=	nie je rovné
true	pravda
false	nepravda
&&	a zároveň
| |	alebo

Tab. 4.3: Operátory používané v NXC.

Operátor	Funkcia
=	priradenie výrazu do premennej
+ =	pričítanie výrazu k premennej
- =	odčítanie výrazu od premennej
* =	vynásobenie premennej výrazom
/ =	vydelenie premennej výrazom
% =	vráti zvyšok po delení premennej výrazom
& =	bitové A výrazu a premennej
| =	bitové ALEBO výrazu a premennej
| | =	absolútna hodnota výrazu
+ - =	do premennej nastaví +/- výrazu
>> =	bitový posun doprava o počet bitov určených výrazom
<< =	bitový posun doľava o počet bitov určených výrazom
+ +	zvýšenie hodnoty premennej o 1
- -	zníženie hodnoty premennej o 1

Pre zopakovanie si uvedenie zoznam základných pravidiel NXC, ktoré je potrebné dodržať pri písaní programu:

- Komentáre sa píšu v riadkoch za kód použitím //. Počítač tieto komentáre nečíta ani nekompiluje.
- Každý NXC program musí začínať "task main()" . Tento príkaz je to, čo robot na začiatku hľadá a vykoná ako prvé.
- Zložené zátvorky {} obklopujú jednotlivé tasky (úlohy).
- Každá task končí bodkočiarkou.
- Na počte medzier nezáleží (narozdiel od programovacieho jazyka Python), ale zložené zátvorky, bodkočiarky a jednoduché zátvorky sú v NXC veľmi dôležité.
- NXC je case-sensitive, to znamená, že záleží na veľkosti písmen [7].

Ako prehľad funkcií a príkazov používaných v nxcEditore slúži Tab. 4.4 [8].
Pozn.: Pred každým parametrom v syntaxi príkazov v Tab. 4.4 je uvedený aj typ premennej. Typy premenných a ich veľkosti sú spracované v Tab. 4.1.

Tab.4.4: Funkcie nxcSimulátora.

Funkcia	Opis funkcie
Array	
ArrayInit `void ArrayInit (variant & aout[], variant value, unsigned int count )`	inicializuje pole **Parametre:** aout- výstup poľa na inicializáciu value- hodnotu, ktorú chceme priradiť každému prvku count- počet prvkov vytvorených vo výstupe poľa
Motor	
RotateMotor `void RotateMotor ( byte outputs, char pwr, long angle )`	otáča motor **Parametre:** outputs- výstupné porty pwr- rýchlosť na výstupe s hodnotami od 0 do 100. Môže byť záporná pre opačný smer. angle- uhol limitovaný v stupňoch. Môže byť záporný pri opačnom smere.
MotorRotationCount `long MotorRotationCount(byte output)`	vráti stupne (resp. inžinierske jednotky), o koľko sa otočí motor vo formáte long int **Parametre:** output –výstupný port (OUT_A, OUT_B, OUT_C).
ResetRotationCount `void ResetRotationCount(byte output)`	resetuje hodnotu, ktorú vráti `MotorRotationCount` na 0 **Parametre:** output-výstupný port
OnFwd `void OnFwd( byte outputs, char pwr )`	motory sa pohybujú dopredu **Parametre:** outputs- výstupné porty pwr- rýchlosť na výstupe s hodnotami od 0 do 100. Môže byť záporná pre opačný smer. angle- uhol v stupňoch. Môže byť záporný pri opačnom smere.
OnRev `void OnRev (byte outputs, char pwr )`	motory sa pohybujú dozadu **Parametre:** outputs- výstupné porty pwr- rýchlosť na výstupe angle- uhol v stupňoch.

Funkcia	Opis funkcie
Off `void Off (byte outputs)`	vypne motor **Parametre:** outputs- výstupné porty
Coast `void Coast(byte outputs)`	motor nie je zabrzdený, ale odpojený od napájania, postupne sa zastaví zotrvačnosťou **Parametre:** outputs- výstupné porty
Float `void Float(byte outputs)`	alias pre Coast
Senzory	
Sensor(...); SENSOR_... `unsigned int Sensor (const byte & port)`	hodnota senzora na danom porte **Parametre:** port- port senzora (IN_1, IN_2, IN_3 ..)
MotorRotationCount `long MotorRotationCount (byte output)`	vráti stupne (resp. inžinierske jednotky), o koľko sa otočí motor vo formáte long int **Parametre:** output – výstupný port (OUT_A, OUT_B, OUT_C...)
ResetRotationCount `void ResetRotationCount(byte output)`	resetuje hodnotu, ktorú vráti `MotorRotationCount` na 0 **Parametre:** output-výstupný port
SensorUS `byte SensorUS (const byte port )`	hodnota ultrazvukového senzora **Parametre:** port- port UV senzora

SetSensorLight
`void SetSensorLight(const byte & port,`
`bool bActive = true`
`)`
	nastaví svetelný senzor
	Parametre:
	port- port senzora
	bActive- pomocou true/false indikuje to, či je port senzora nastavený ako aktívny alebo neaktívny. Prednastavená hodnota tohto parametra je true.

SetSensorColorFull `void SetSensorColorFull (const byte & port)`	nastaví farebný senzor v plnom farebnom režime na určitom porte **Parametre:** port- port senzora

Funkcia	Opis funkcie
SetSensorTouch `void SetSensorTouch(const byte & port)`	nastaví dotykový senzor **Parametre:** port- port senzora

ButtonPressed
`bool ButtonPressed (const byte btn,`
`bool resetCount = false`
`)`
	skontroluje, či je tlačidlo stlačené
	Parametre:
	btn- tlačidlo, ktoré sa má kontrolovať
	resetCount- možnosť resetovania tlačidla

SetSensorMode
`void SetSensorMode(const byte & port,`
`byte mode`
`)`
	nastaví režim tlačidla
	Parametre:
	port- port senzora
	mode- režim senzora

Timer	
Wait `void Wait (unsigned long ms)`	zastaví úlohu na určitý čas (v milisekundách) **Parametre:** ms- počet milisekúnd
FirstTick `unsigned long FirstTick ()`	vráti 32-bitovú hodnotu, čo je hodnota časovania systému (“tick”) v milisekundách v čase, kedy program začal bežať
CurrentTick `unsigned long CurrentTick()`	prečíta aktuálny systémový “tick”
Displej	

TextOut
`char TextOut (int x,`
`int y,`
`string str,`
`unsigned long options =`
`DRAW_OPT_NORMAL`
`)`
	vypíše text
	Parametre:
	x, y- súradnice LCD displeja
	y- číslo riadku
	str- text, ktorý má výstup na LCD displeji
	options- možnosti vykresľovania

NumOut
`char NumOut (int x,`
`int y,`
`variant value,`
`unsigned long options =`
`DRAW_OPT_NORMAL`
`)`
	vypíše číslo
	Parametre:
	x, y- súradnice LCD displeja
	y- číslo riadku
	variant value- číselná hodnota na LCD displeji, akýkoľvek číselný typ je podporovaný
	options-možnosti vykresľovania

Funkcia	Opis funkcie

GraphicOut
`char GraphicOut ( int x,`
`int y,`
`string filename,`
`unsigned long options = DRAW_OPT_NORMAL`
`)`
	vykreslí grafický obrázok
	Parametre:
	x, y- súradnice LCD displeja
	filename- názov RIC obrázka
	options- možnosti vykresľovania

ClearScreen `void ClearScreen ()`	vymaže všetko na LCD obrazovke
Zvuk	

PlayTone (f < 4200 Hz)
`char PlayTone (unsigned int frequency,`
`unsigned int duration`
`)`
	zahrá jeden tón o určitej frekvencii a dĺžke trvania (v milisekundách)
	Parametre:
	frequency- frekvencia tónu v Hz
	duration- dĺžku tónu v ms

PlayToneEx
`char PlayToneEx (unsigned int frequency,`
`unsigned int duration,`
`byte volume,`
`bool loop`
`)`
	zahrá jeden tón o určitej frekvencii, dĺžke trvania a hlasitosti
	Parametre:
	frequency- frekvencia tónu v Hz
	duration- dĺžku tónu v ms
	volume- hlasitosť tónu
	loop- boolean hodnota indikujúca, či treba opakovane prehrávať tón

PlayFile `char PlayFile (string filename)`	prehrá určitý .rso súbor **Paramtere:** filename- názov zvukového súboru alebo melódie
If and switch	
Switch ... case `switch (expression) body`	vykonáva jednu z niekoľkých častí kódu v závislosti od hodnoty výrazu
If `if(condition)consequence`	`if` vyhodnocuje podmienku ak je podmienka splnená (true), vykoná sa príkaz – dôsledok (consequence). Hodnota podmienky sa považuje za false, keď sa vyhodnocuje nula. V prípade vyhodnocovania nenulovej hodnoty vráti true.

Funkcia	Opis funkcie
If ... else `if(condition)consequence else alternative`	`if …else` vyhodnocuje podmienku ak je podmienka splnená (true), vykoná sa príkaz – dôsledok (consequence). Druhý príkaz (alternative), ktorému predchádza príkaz `else` sa vykoná v prípade, ak podmienka nie je splnená (false). V prípade nuly vráti false, pri nenulových hodnotách vracia true.
Slučky	
Repeat `repeat (expression) body`	vykoná slučku určitý počet opakovaní výraz (expression) určuje koľkokrát sa vykonajú príkazy v tele (body)
For `for (statement1 ; condition ; statement2) body`	umožňuje automatickú inicializáciu a inkrementáciu premenných; slučka v príklade vždy vykoná príkaz1(statement1), následne opakovane kontroluje stav podmienky. Pokiaľ zostáva podmienka pravdivá (true), vykoná príkaz2 (statement2)
While `while (condition) body`	používa sa na vytvorenie podmienenej slučky. Najprv sa vyhodnotí podmienka, ak je splnená (true), telo (body) slučky sa vykoná a následne sa podmienka opäť kontroluje/vyhodnocuje. Tento proces sa opakuje dovtedy, kým je podmienka nesplnená (false), alebo pokiaľ sa nevykoná príkaz `break`
Do ... while `do body while (condition)`	varianta while slučky rozdiel medzi `while` a `do…while` je v tom, že `do..while` slučka vždy vykoná príkazy v tele (body), zatiaľ čo `while` tieto príkazy nemusí spustiť vôbec

Funkcia	Opis funkcie
Until `until (EVENT_OCCURS);`	príkaz `until` vykonáva slučku dovtedy, dokým sa podmienka nesplní.
Úlohy - Tasks	

Precedes
`void Precedes ( task task1,`
`task task2,`
`...,`
`task taskN`
`)`
	spustí všetky úlohy súčasne
	Parametre:
	task1 – prvá úloha, ktorá sa začne vykonávať po tom, čo súčasná úloha skončila.
	task2- druhá úloha, ktorá sa začne vykonávať po tom, čo súčasná úloha skončila
	taskN- posledná úloha, ktorá sa začne vykonávať po tom, čo súčasná úloha skončila

Start `void StartTask (task t)`	spustí úlohy **Parametre:** task t- úloha t, ktorú chceme začať
Acquire `void Acquire (mutex m)`	získa zadanú "mutex" premennú; mutex = objekt, ktorý umožňuje viacerým programovým vláknam zdieľať rovnaký zdroj, ako napr. prístup k súborom, avšak nie súčasne. **Parametre:** m- mutex na získanie
Release `void Release (mutex m)`	= Acquire
StopAllTasks `void StopAllTasks ()`	ukončí všetky aktuálne spustené úlohy
Stop `void Stop (bool bvalue)`	zastaví program **Parametre:** bvalue- ak je táto hodnota true,program ukončí vykonávanie úlohy
Súbory	

WriteString
`unsigned int WriteString (byte handle,`
`const string & str,`
`unsigned int & cnt )`
	zapíše reťazec do súboru
	Parametre:
	handle=file handle
	str- reťazec, ktorý sa do súboru má zapísať
	cnt- počet bytov aktuálne vpísaných v súbore

Funkcia	Opis funkcie

WriteLnString
`unsigned int WriteLnString (byte handle,`
`const string & str,`
`unsigned int & cnt`
`)`
	zapíše reťazec a nový riadok do súboru
	Parametre:
	handle=file handle
	str- reťazec, ktorý sa do súboru má zapísať
	cnt- počet bytov aktuálne vpísaných v súbore

CreateFile
`unsigned int CreateFile (string fname,`
`unsigned int fsize,`
`byte & handle`
`)`
	vytvorí súbor
	Parametre:
	fname- názov súboru, ktorý chceme vymazať
	fsize – veľkosť súboru
	handle= file handle- číslo, ktoré operačný systém dočasne priradí súboru, keď sa súbor otvorí a operačný systém používa "file handle" pri pristupovaní k súborom. Pre "file handle" sa vymedzuje špeciálna časť pamäte a veľkosť tejto časti určuje, koľko súborov sa môže súčasne otvoriť.

CloseFile `unsigned int CloseFile (byte handle)`	zatvorí súbor **Parametre:** handle = file handle
DeleteFile `unsigned int DeleteFile (string fname)`	vymaže súbor **Parametre:** fname- názov súboru, ktorý chceme vymazať
Znaky:	

StrCat
`string StrCat (string str1,`
`string str2,`
`string strN`
`)`
	skopíruje/pripojí reťazec za iný
	Parametre:
	str1-prvý reťazec
	str2- druhý reťazec
	strN- posledný reťazec

NumToStr `string NumToStr (variant num)`	prevedie číslo na reťazec **Parametre:** num- číslo
Ďalšie príkazy	
Random `int Random (unsigned int n = 0)`	generuje náhodného čísla **Parametre:** n- binárna hodnota

Funkcia	Opis funkcie

Sin, Cos
`float sin (float x)`
`float cos float x)`
	vypočíta sinus, cosinus
	Parametre:
	x- sinus/kosínus x

V nxcEditore je možné priamo použiť ".ric "obrázky a ".rso" zvukové súbory, čo sú formáty obrázkov a zvukových súborov používaných pri programovaní NXT.

5. Implementácia algoritmov sledovania čiernej čiary

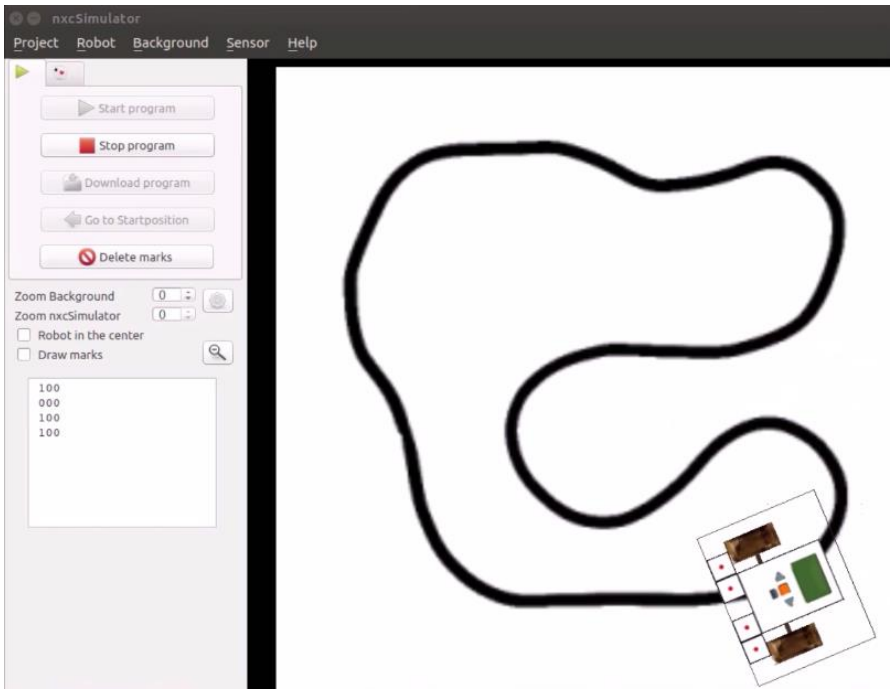
V tejto kapitole sa budeme venovať štyrom algoritmom:

1. Sledovanie čiary prostredníctvom štyroch senzorov.
2. Sledovanie čiary dvoma senzormi.
3. Sledovanie čiary jedným svetelným senzorom.
4. Sledovanie čiary jedným svetelným senzorom a obídenie prekážky.

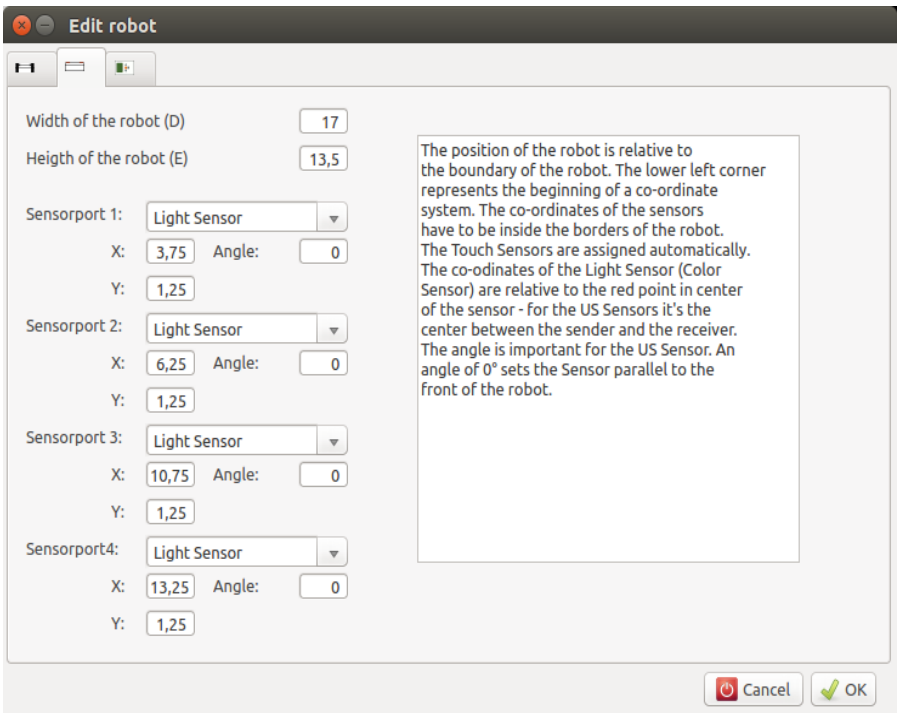
Vrámci sledovania čiary jedným svetelným senzorom bude teoreticky spracovaný uzavretý regulačný obvod a PID regulátor, pomocou ktorého sme vykonávali riadenie robota.

5.1 Robot so štyrmi svetelnými senzormi

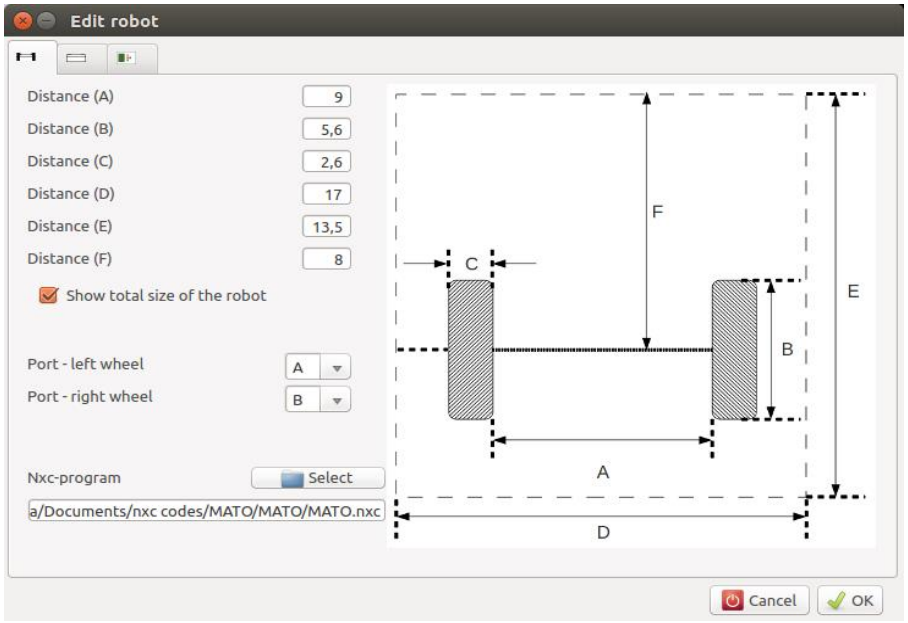
Robot so štyrmi svetelnými senzormi (Obr 5.10) má upravené rozmery tak, aby mal medzi vnútornými senzormi dostatok priestoru na to, aby nimi detekoval čiernu čiaru (Obr. 5.11).



Obr. 5.10: Robot so štyrmi svetelnými senzormi.

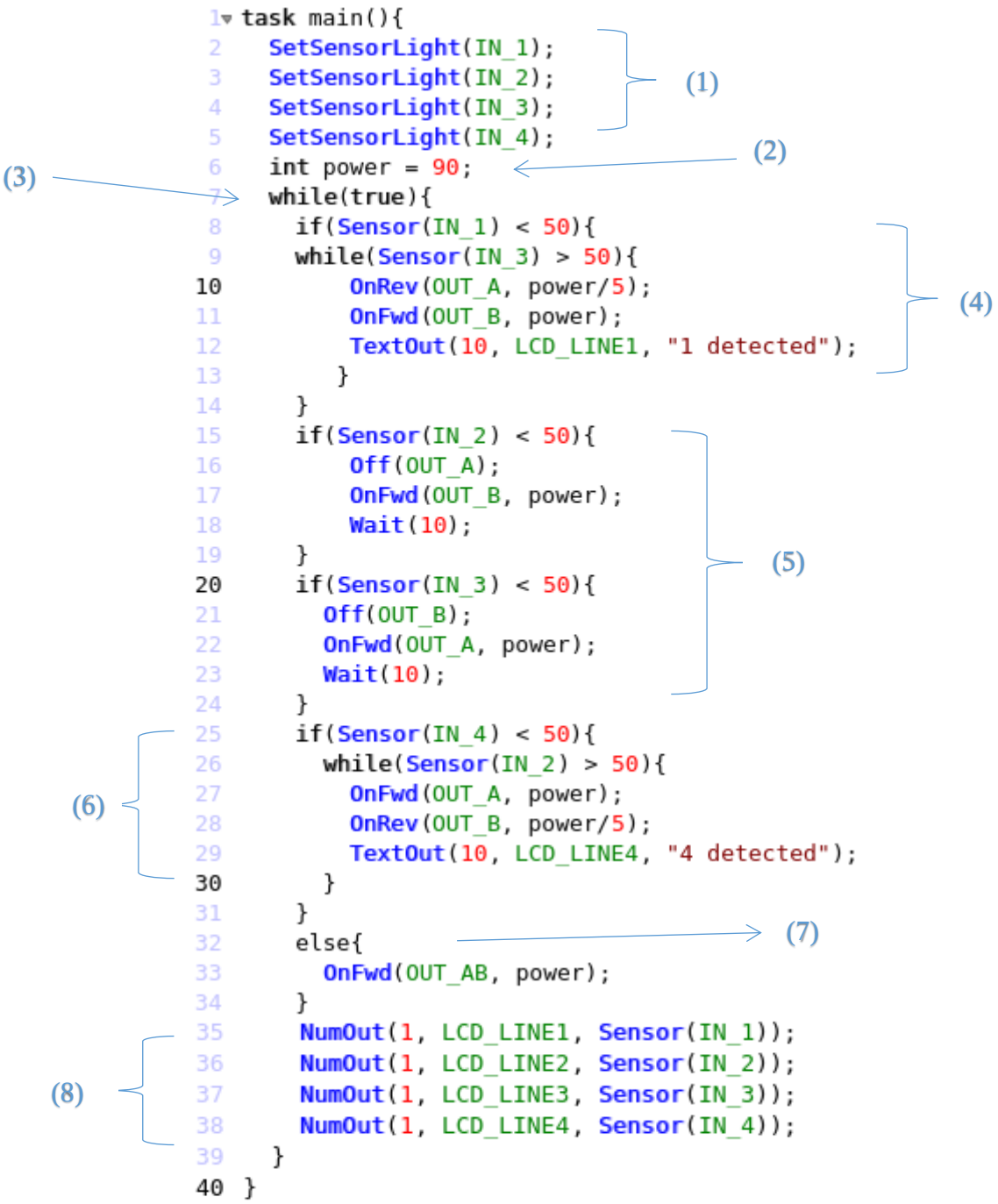


Obr. 5.11: Senzory a ich usporiadanie v prípade robota so štyrmi svetelnými senzormi.



Obr 5.12: Vzdialenostné parametre robota so štyrmi svetelnými senzormi.

Robot je naprogramovaný Algoritmom 5.1. Jednotlivé časti algoritmu sú očíslované a bližšie vysvetlené na nasledujúcej strane a syntax príkazov použitých v algoritme je uvedená v Tab. 4.4.




```
1  task main(){
2    SetSensorLight(IN_1);
3    SetSensorLight(IN_2);
4    SetSensorLight(IN_3);
5    SetSensorLight(IN_4);
6    int power = 90;
7    while(true){
8      if(Sensor(IN_1) < 50){
9      while(Sensor(IN_3) > 50){
10         OnRev(OUT_A, power/5);
11         OnFwd(OUT_B, power);
12         TextOut(10, LCD_LINE1, "1 detected");
13        }
14     }
15     if(Sensor(IN_2) < 50){
16         Off(OUT_A);
17         OnFwd(OUT_B, power);
18         Wait(10);
19     }
20     if(Sensor(IN_3) < 50){
21       Off(OUT_B);
22       OnFwd(OUT_A, power);
23       Wait(10);
24     }
25     if(Sensor(IN_4) < 50){
26       while(Sensor(IN_2) > 50){
27         OnFwd(OUT_A, power);
28         OnRev(OUT_B, power/5);
29         TextOut(10, LCD_LINE4, "4 detected");
30       }
31     }
32     else{
33       OnFwd(OUT_AB, power);
34     }
35     NumOut(1, LCD_LINE1, Sensor(IN_1));
36     NumOut(1, LCD_LINE2, Sensor(IN_2));
37     NumOut(1, LCD_LINE3, Sensor(IN_3));
38     NumOut(1, LCD_LINE4, Sensor(IN_4));
39   }
40 }
```

Algoritmus 5.1: Algoritmus robota sledujúceho čiernu čiaru pomocou štyroch svetelných senzorov.

Podrobné vysvetlenie Algoritmu 5.1:

(1) Pri písaní programu je potrebné si na začiatku v časti „task main" zadefinovať senzory, s ktorými bude robot pracovať. Z kapitoly 1.1 vieme, aké typy senzorov môžeme použiť a že senzory sa zapájajú do portov 1 až 4. V prvej časti kódu sme si teda príkazom

`SetSensorLight` zadefinovali, že chceme použiť štyri svetelné senzory a priradili sme im jednotlivé porty; port 1,2,3 a 4; `SetSensorLight(IN_1)` ... až `SetSensorLight(IN_4)`.

(2) Ďalej si musíme zadefinovať rýchlosť. Keďže rýchlosť v NXC môže nadobúdať celočíselné hodnoty od 0 po 100 (Tab. 4.4), je typu integer („int“).

(3) Cyklus „while true” predstavuje nekonečný cyklus, v ktorom sa vykonávajú podmienky písané ďalej v kóde. Slovo „true“ nám udáva pravdu, čiže splnenie príkazov vo while cykle. Spojenie „while true“ teda znamená „pokiaľ sa nasledujúce podmienky splnia, vykonávaj ich“.

(4) Keďže svetelné senzory detekujú čiernu čiaru na základe jasu (0- čierna, 100- biela), hodnota „< 50“ v if podmienke znamená detekciu čiernej čiary.

Na Obr. 5.12 môžeme vidieť, že ľavý motor je v porte A a pravý motor v porte B. Prvú if podmienku môžeme teda slovne prepísať nasledovne: ak prvý senzor vpravo (senzor v porte 1) zachytí intenzitu svetla menšiu ako 50 a pokiaľ vnútorný senzor v treťom porte čiaru nezachytí, bude robot ľavým motorom cúvať (`OnRev(OUT_A,power/5)`) a pravým pôjde dopredu plnou rýchlosťou (`OnFwd(OUT_B,power)`) . V príkaze `OnRev` je rýchlosť vydelená piatimi (`power/5`), pretože chceme, aby cúval pomalšie, a tým zabránil neplynulému pohybu dopredu. Príkazom `OnRev` zároveň zabezpečíme narovnanie robota na čiaru.

Príkazom `TextOut` si môžeme nechať vypisovať text.

(5) Pri detekovaní čiary vnútornými senzormi v portoch 2 a 3 sme použili príkaz `Off` na zastavenie motora a príkaz `OnFwd` na pokračovanie robota dopredu. Senzor dva sa u nás nachádza vpravo. Motor v porte A zabezpečí pootočenie vľavo a motor v porte B vpravo, v smere hodinových ručičiek. Ak teda druhý senzor zachytí čiaru, motor sa bude otáčať doprava (`OnFwd(OUT_B,power)`, pokiaľ sa senzor nachádza nad čiarou. Analogicky funguje aj otáčanie tretieho senzora. Je dôležité si uvedomiť, že stále sme vo while cykle, ktorý sa zatiaľ stále vykonáva, preto sme na vykonanie jednej otočky v tejto časti kódu vyhradili 10 milisekúnd (`Wait(10)`). Ak by sme napríklad vyhradili viac milisekúnd, tak sa robot môže príliš otočiť (v zmysle veľkého uhla), až prejde senzorom cez čiaru. Naopak, v prípade menšieho počtu milisekúnd síce tiež prejde cez čiaru, ale kvôli tomu, že za každým pootočením je pohyb dopredu.

(6) Podmienka pre štvrtý a druhý senzor je rovnaká ako v časti (4).

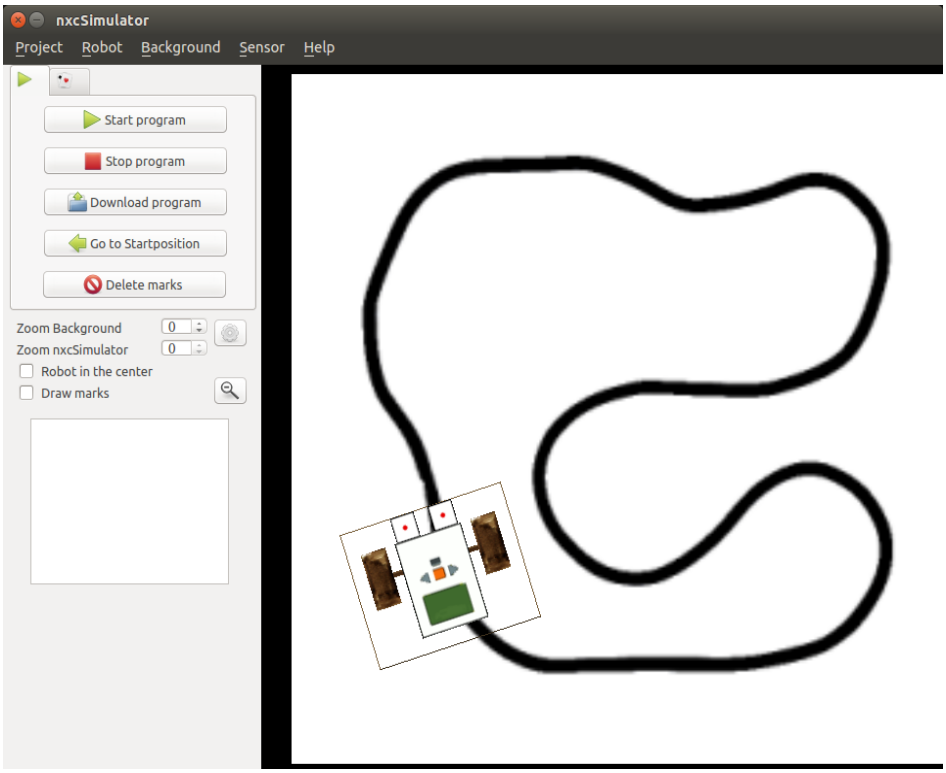
(7) Ak sa žiadna z predchádzajúcich podmienok while cyklu nevykoná, spustí sa príkaz `else`, ktorý nám o našom robotovi v podstate hovorí, že sa mu nepodarilo zachytiť čiernu čiaru. Robot v takomto prípade pokračuje oboma motormi dopredu (`OnFwd(OUT_AB)`), až kým nenarazí na čiernu čiaru, kedy sa budú opäť vykonávať časti algoritmu opísané v bodoch (3) až (7).

(8) Pre vypisovanie hodnôt intenzity osvetlenia (od 0 po 100) sme použili príkaz `NumOut`.

Takto naprogramovaný robot dokáže nielen sledovať, ale aj nájsť čiernu čiaru. V ľavej časti nxcSimulátora sa vypisujú hodnoty od 0 do 100, vďaka čomu pri simulácii dokážeme rozoznať, ktorým senzorom aktuálne robot detekuje čiaru. Na Obr. 5.10 je možné vidieť, že robot rozpoznal čiernu čiaru druhým svetelným senzorom, o čom nám hovorí aj vypísaná hodnota 000 na druhej pozícii.

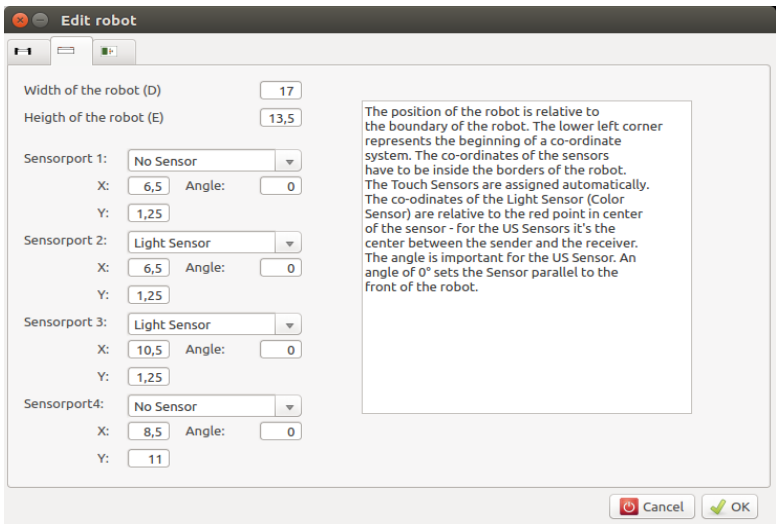
5.2 Robot s dvoma svetelnými senzormi

Na sledovanie čiernej čiary je možné použiť aj robota s dvoma svetelnými senzormi (Obr.5.13).



Obr. 5.13: Robot a pozadie použité na sledovanie čiernej čiary v prípade robota s dvoma svetelnými senzormi.

Usporiadanie senzorov v prípade robota s dvoma svetelnými senzormi je ukázané na Obr. 5.14. Senzory sme umiestnili do portov číslo dva a tri.



Obr. 5.14: Senzory a ich usporiadanie v prípade robota s dvoma svetelnými senzormi.

Algoritmus 5.2 použitý pri robotovi s dvoma svetelnými senzormi je logikou analogický s Algoritmom 5.1.

```
task main(){

  SetSensorLight(IN_2);
  SetSensorLight(IN_3);

  int power = 50;
  while(true){

    if(Sensor(IN_2) < 50){
        Off(OUT_A);
        OnFwd(OUT_B, power);
        Wait(70);
    }
    if(Sensor(IN_3) < 50){
      Off(OUT_B);
      OnFwd(OUT_A, power);
      Wait(70);
    }

    else{
      OnFwd(OUT_AB, power);
    }

    NumOut(1, LCD_LINE2, Sensor(IN_2));
    NumOut(1, LCD_LINE3, Sensor(IN_3));

  }
}
```

Algoritmus 5.2: Algoritmus robota sledujúceho čiernu čiaru dvoma svetelnými senzormi.

Premietnime si na chvíľu naše algoritmy do reálneho riadenia a predstavme si, že prostredníctvom nich riadime auto. Ak by sme k autu pripojili štyri senzory, auto by malo byť schopné hľadať a sledovať čiernu stopu pri akejkoľvek rýchlosti. Povedzme, že tou našou stopou je kľukatá cesta. Ak by sme spomalili, zabrzdili a následne prudko zrýchlili, auto by stále pokračovalo v jazde po ceste. Auto s dvoma senzormi je tak isto schopné sledovať cestu. K problému však dôjde vtedy, ak sa rozhodneme ísť príliš rýchlo alebo by sme mali prejsť príliš prudkou zákrutou. V takom prípade auto vybočí a pokračuje ďalej. Môžu nastať dva prípady; buď po čase auto narazí na prekážku a nebude môcť ďalej pokračovať alebo dorazí k inej ceste, po ktorej bude opäť jazdiť dovtedy, kým mu to rýchlosť a trasa dovolia.

Na príklade s autom sme si ukázali, že pri robotovi so štyrmi svetelnými senzormi môžeme v Algoritme 5.1 použiť aj maximálnu možnú rýchlosť – 100, pretože robot má 4 svetelné senzory. To znamená, že aj keby vnútorné senzory nestihli kvôli vysokej rýchlosti zachytiť čiernu čiaru, stále sú tu ďalšie dva senzory (vonkajšie senzory), ktoré ju zachytia a naopak. Robot so štyrmi senzormi je schopný detekovať aj zložitejšie trasy s ostrejšími uhlami na rozdiel od robota s dvoma senzormi, ktorý je navyše obmedzený aj nižšou rýchlosťou. Nižšia rýchlosť zabezpečí to, že robot nevyjde z čiary a bude ju schopný sledovať, aj keď nebude priama, ale pri ostrom uhle čiary by z trasy opäť vyšiel. Ak by priama bola, robot s dvoma senzormi by sa, samozrejme, mohol pohybovať aj maximálnou rýchlosťou.

Algoritmus 5.2 bol vyskúšaný aj v riadení lego robota (Obr. 5.15, Obr.5.16). Čiara, ktorú robot sledoval je na Obr. 5.17. Čiara je prerušená, pretože algoritmus bol skúšaný aj pre prerušovanú čiaru, ktorú robot s dvoma senzormi zvládol prejsť. V praxi je totiž možné sa stretnúť napr. s opotrebovanou cestou. Videoukážka sledovania súvislej čiary je dostupná na webe [B]. Robot pri reálnom riadení dokázal prejsť trasu maximálnou rýchlosťou 50. Pri rýchlosti 60 už z čiary vyšiel. Tento výsledok sa zhoduje s výsledkom pozorovaným v nxcSimulátori.

Obr. 5.15 a Obr. 5.16: NXT robot s dvoma svetelnými senzorom.

Popis jednotlivých častí robota z Obr. 5.15 a 5.16:

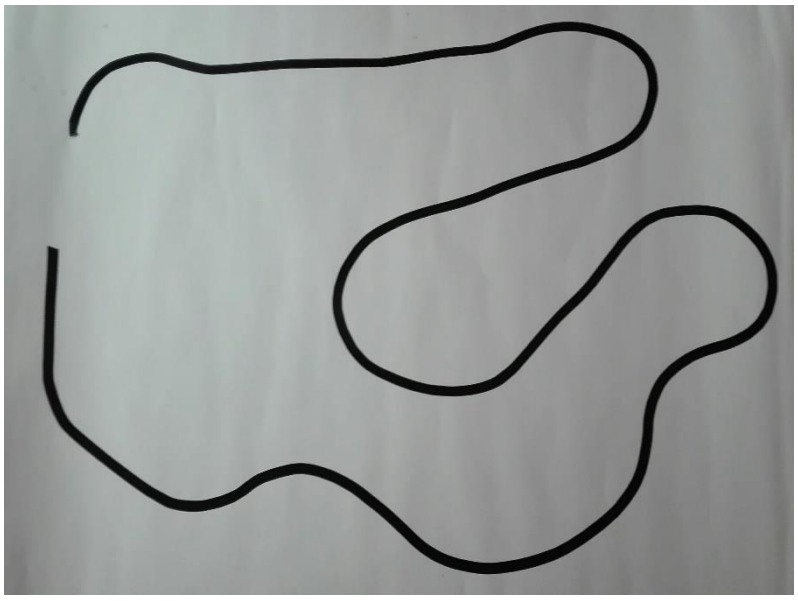
(1): svetelný senzor

(2): kocka

(3): motor

(4): vstupné porty, port 1, 2, 3 a 4 pre napájanie senzorov

(5): výstupné porty A,B,C pre motory



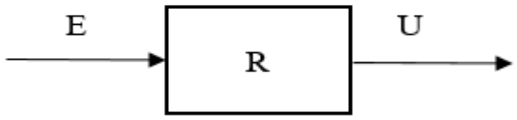
Obr.5.17: Trasa použitá pri reálnom riadení robota.

5.3 Robot s jedným svetelným senzorom

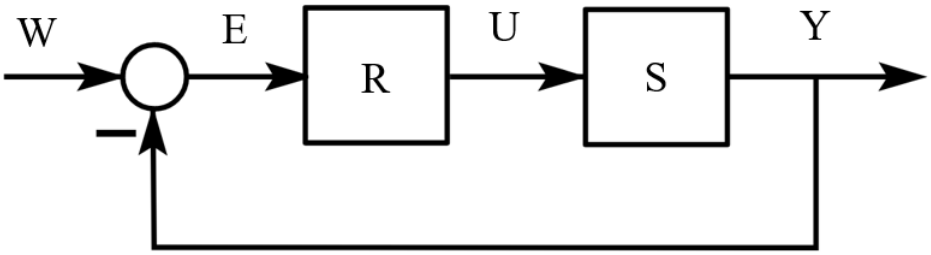
O sledovanie čiary jedným senzorom sa snažíme hlavne preto, že je finančne menej náročné ako použitie viacerých senzorov. To, aby robot sledoval čiaru jediným senzorom, dokáže zabezpečiť PID regulátor. PID regulátor predstavuje regulačný mechanizmus, ktorým sa snažíme dosiahnuť požadovanú hodnotu, v našom prípade hodnotu intenzity osvetlenia predstavujúcu čiernu čiaru.
PID regulátor nachádza využitie aj v mnohých iných sférach priemyslu: pri regulovaní teploty, tlaku, množstva a prietoku vody, hladiny vody v zásobníkoch a pod.
Na to, aby sme si mohli vysvetliť riadenie nášho systému (robota resp. auta sledujúceho čiernu čiaru), musíme si najprv zadefinovať pojmy z riadenia procesov a keďže je táto práca písaná aj pre študentov, ktorí s programovaním NXT robotov začínajú už na základných, či stredných školách, z riadenia procesov si vysvetlíme len tie najpodstatnejšie pojmy. Zainteresovaní čitatelia nájdu ohľadom riadenia procesov viac informácií v knihe Riadenie procesov [9].

5.3.1 URO

Pojmy z riadenia procesov, bez ktorých sa určite nezaobídeme, sú regulátor a uzavretý regulačný obvod (URO). Regulátor je zariadenie, ktoré sa stará, aby daný systém automaticky pracoval a fungoval v požadovanom rozsahu hodnôt.
Uzavretý regulačný obvod je vo všeobecnosti súbor technických prostriedkov, ktoré slúžia na to, aby vybrané zariadenie (proces) uspokojivo pracovalo v automatickom režime. URO predstavuje spätnoväzbové riadenie. Spätnoväzbové riadenie je vo svojej podstate najjednoduchším typom riadenia, ktoré dokáže eliminovať vplyv zmien žiadaných veličín [10]. Berie do úvahy výstup systému, ktorý umožňuje systému nastaviť výkon tak, aby vyhovoval požadovanej výstupnej odozve. Pre prípad robota sledujúceho čiernu čiaru môžeme túto definíciu prepísať takto: Spätnoväzbové riadenie berie do úvahy aktuálnu hodnotu nameranú senzorom a táto hodnota umožňuje robotovi nastaviť takú rýchlosť motorov, aby plynule sledoval čiernu čiaru. Hlavnou nevýhodou spätnoväzbového riadenia je, že vstup do riadeného procesu je zmenený až vtedy, keď sa výstup z procesu začne meniť [10]. Spätnoväzbové riadenie je na nasledujúcich schémach značené šípkou smerujúcou od výstupu smerom k žiadanej veličine. Abstrakciou URO je jeho všeobecná bloková schéma, ktorej zjednodušenie je na Obr. 5.19 [9]. Jednotlivé časti sú popísané pod obrázkom.



Obr. 5.18: Bloková schéma regulátora.



Obr. 5.19: Zjednodušená schéma URO.

R: regulátor

S: riadený systém

W: žiadaná veličina

E: regulačná odchýlka

U: riadiaca (akčná) veličina

Y: riadená (výstupná) veličina

Platí: $E = W - Y$

Riadený systém je naša predstava o riadenom procese zo systémového hľadiska. Záleží nám na riadenej veličine *Y*, ktorá je výstupom z riadeného procesu a ktorú dokážeme ovplyvňovať riadiacou (alebo akčnou) veličinou *U*. Regulátor je zariadenie, ktoré vyhodnocuje regulačnú odchýlku *E* a generuje takú akčnú veličinu *U*, aby sme na výstupe dosiahli referenciu *W* (Obr.5.18) [9].

5.3.2 PID regulátor

PID regulátor je súčasťou systému so spätnoväzbovým riadením. PID regulátor budeme v texte vysvetľovať pomocou názvov premenných, ktoré sú zaužívané pri zavádzaní PID algoritmu v programovaní NXT robotov. Veličiny použité v obrázku 5.19 by sme mohli prepísať nasledovne:

R: algoritmus

S: robot resp. auto

W: `setpoint`

E: `error`

U: rýchlosť motorov; `output`

Y: hodnota nameraná senzorom; `actual_value`

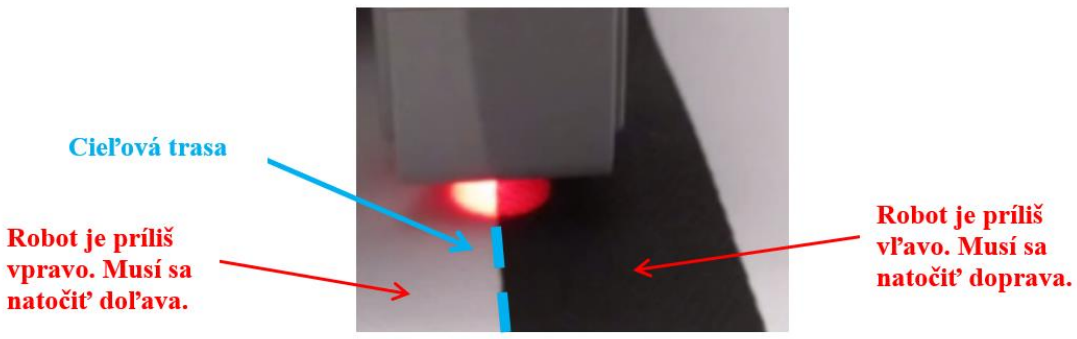
Žiadanú hodnotu budeme označovať ako `setpoint`. `Setpoint` získame kalibráciou senzora. Ide o spriemerovanú najnižšiu a najvyššiu nameranú hodnotu, ktorú robot získa počas krátkeho

prechádzania trasou. Kalibrácia je potrebná najmä kvôli meniacim sa svetelným podmienkam. Kalibrácia bude bližšie vysvetlená v Algoritme 5.4.
Premenná `actual_value` predstavuje hodnotu, ktorú senzor zachytí v aktuálnom okamihu. Regulačná odchýlka (`error`) je rozdiel týchto dvoch hodnôt:

```
error = setpoint - actual_value
```

Inými slovami predstavuje to, ako ďaleko mimo čiary sa robot posunul. Táto odchýlka sa použije na výpočet toho, o koľko treba zmeniť výstup (`output`), aby sme našu aktuálnu hodnotu priblížili čo najbližšie tej požadovanej. Naším cieľom je teda odchýlku minimalizovať. V skratke, PID algoritmus v reálnom čase vypočítava ako prudko má robot zatáčať, aby čiaru sledoval plynule.

Dôležitým faktorom pri použití PID regulátora na sledovanie čiary je to, že svetelný senzor v podstate nesleduje čiernu čiaru, ale časť medzi bielou a čiernou čiarou, teda sa nachádza v strede medzi bielou a čiernou (Obr. 5.20). Našou cieľovou trasou je teda rozhranie medzi týmito dvoma farbami. Dôvod, prečo chceme, aby sa senzor nachádzal medzi bielou a čiernou je ten, aby robot vedel, kedy sa od cieľovej trasy odchyľuje. Keby sa senzor nachádzal iba na čiernej, nevedel by ktorým smerom sa odchyľuje, pretože z oboch strán by bola biela farba. Avšak keď sa nachádza v strede medzi bielou a čiernou a keď už nameria vysokú hodnotu intenzity osvetlenia, vtedy už vie, že sa odchyľuje od čiary (smerom k bielej) a pomocou PID výpočtu sa vráti späť jedným smerom. Naopak, keď senzor nameria príliš nízku hodnotu intenzity osvetlenia (odchyľuje sa do druhej strany, smerom k čiernej), vráti sa späť druhým smerom. Tento princíp je znázornený na obrázku 5.20.
Vďaka `setpointu` senzor vie, že celý čas by mal namerať práve hodnotu `setpointu`. Keď je aktuálne nameraná hodnota príliš vysoká (biela), otočí sa do jednej strany, ak je príliš nízka (čierna), otočí sa do druhej strany. Týmto spôsobom bol navrhnutý Algoritmus 5.4, ktorým riadime robota s jedným svetelným senzorom.

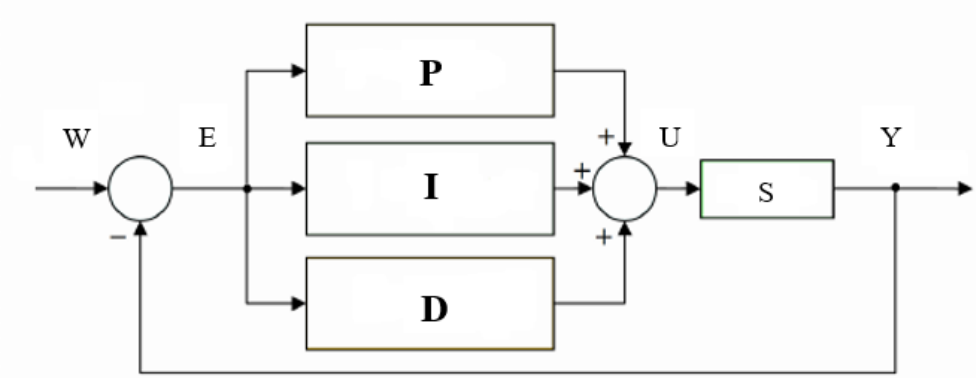


Obr. 5.20: Poloha senzora pri PID algoritme.

Zovšeobecnený PID algoritmus znázorňuje Algoritmus 5.3. Aby sme ho mohli pochopiť, poďme sa teraz pozrieť na jednotlivé zložky PID regulátora. Už skratka PID napovedá, že regulátor obsahuje tri zložky: proporcionálnu (P), integračnú (I) a derivačnú (D). Pre lepšiu predstavu je na Obr. 5.21 uvedená bloková schéma regulátora.

```
lastError = 0
integral = 0
start:
error = setpoint - actual_value
integral = integral + (error*dt)
derivative = (error - lastError)/dt
output = (Kp*error) + (Ki*integral) + (Kd*derivative)
lastError = error
wait(dt)
goto start
```

Algoritmus 5.3: Pseudoalgoritmus PID regulátora. Hodnota `dt` určuje citlivosť PID regulátora, musí byť väčšia ako 0 a hovorí nám o reakčnom čase robota. Je to tzv. čas vzorkovania.



Obr. 5.21: Bloková schéma PID regulátora.

P + I + D = PID

S: robot

W: `setpoint`

E: `error`

U: `output`

Y: `actual_value`

Proporcionálna zložka

Táto zložka meria odchýlku od požadovanej hodnoty, a to tak, že odčíta aktuálnu hodnotu (`actual_value`) od požadovanej (`setpoint`). Rozdiel (odchýlka, `error`) vynásobí konštantou *Kp*:

```
P = Kp*(setpoint - actual_value)
```

Robot využívajúci iba P zložku síce čiarou prechádza plynule, ale v miestach s ostrou zákrutou z čiary vyjde, preto treba zaviesť I a D zložky. Tie zabezpečia rýchlu odozvu robota.

Integračná regulátor

Keď sa I zložka pridá k P zložke, zrýchli sa pohyb robota smerom k `setpointu` a eliminujú sa zvyškové chyby (`errory`), ku ktorým dochádza, keď P regulátor pracuje sám. I zložka nám teda garantuje konvergenciu k `setpointu`.

Integračná zložka zaznamenáva históriu pohybov robota. Keďže integrál je zovšeobecnením sumy, jedná sa o súčet všetkých hodnôt odchýliek, ktoré boli zaznamenané od spustenia robota. Integračnú zložku teda dostaneme „akumulovaním " regulačných odchýliek. K nejakej dostatočne veľkej premennej, ktorej iniciálna hodnota je 0, pripočítavame aktuálnu hodnotu regulačnej odchýlky vynásobenú časom vzorkovania.

```
integral = integral + (error*dt)
```

Ak sledujeme rovnú čiaru, všetky naakumulované `errory` by mali byť 0. Keď budeme musieť robota narovnať doprava, bude náš `error` kladný. Ak robota musíme narovnať na ľavú stranu čiary, `error` bude záporný. Po sčítaní všetkých `errorov`, by sme mali získať nulu.

Derivačná zložka

Derivačná zložka zložka poskytuje rýchlosť zmeny odchýlky. Najjednoduchšie ju môžeme dostať tak, že urobíme rozdiel aktuálnej a predposlednej regulačnej odchýlky (`lastError`) a tento rozdiel vydelíme časom vzorkovania. Tým D zložka predpovedá nasledujúcu odchýlku.

```
derivative = (error - lastError)/dt
```

Dostaneme tak smernicu, ktorá vyjadruje prudkosť zákruty. Znamienko určuje smer zatáčania. Čím rýchlejšie sa zmení odchýlka v čase, tým bude derivačná hodnota výraznejšie ovplyvňovať výsledok [11].

Všimnime si, že P zložka závisí od aktuálnej hodnoty, I zložka od hodnôt z minulosti a D zložka od budúcnosti (D zložka predikuje kam robot pôjde).

Ako prehľad jednotlivých parametrov používaných v PID algoritme slúži Tab. 5.1.

Tab. 5.1: Jednotlivé hodnoty a parametre PID algoritmu.

Názov	**Opis**	**Hodnota, výpočet**
`setpoint`	Požadovaná hodnota - získaná kalibráciou senzora.	0-100
`actual_value`	Hodnota zachytená senzorom v aktuálnom okamihu.	0-100
`error`	Regulačná odchýlka. Vzdialenosť robota od čiary.	`error = setpoint – actual_value`
`dt`	Čas vzorkovania.	Väčší ako 0
Kp, Ki, Kd	Konštanty. Násobia P, I, D zložku. Znižujú alebo zvyšujú dopad P, I a D.	Ich hodnoty sú veľmi individuálne. Spravidla, hodnota *Kd* býva najvyššia, *Ki* najnižšia.
`P`	Proporcionálna zložka. Zabezpečí plynulé sledovanie čiary. Sledovanie je obmedzené na trasu bez ostrých zákrut.	`P = Kp*error`
`I`	Integrálne zložka. Umožní sledovanie zložitejšej trasy, avšak s agresívnym riadením.	`I =Ki*(I + error*dt)`
`D`	Derivačná zložka. Zabezpečí plynulé prejdenie čiary, obmedzí agresívne riadenie.	`D = Kd*(error - lastError)/dt`
`Output`	PID; súčet P, I a D zložky.	`PID = P + I + D`

5.3.2 Nastavenie PID regulátora

Keď máme všetky tri zložky vypočítané, potrebujeme ich skombinovať do výsledku. Ten dosiahneme jednoduchým spočítaním troch zložiek: PID = P + I + D. Musíme im ale určiť vhodnú váhu – vynásobiť ich určitou konštantou *Kp, Ki, Kd*. Akú hodnotu konštanty zvoliť je to najpodstatnejšie a zároveň najťažšie pre dosiahnutie dobrého riadenia. Nedá sa určiť, aké hodnoty sú optimálne [9].

Fyzické prostredie, v ktorom sa robot pohybuje sa značne líši a nemôže byť matematicky namodelované. Zahŕňa pozemné trenie, indukčnosť motora, ťažisko a veľa ďalších vlastností, preto sú konštanty len odhadnuté čísla získané metódou pokus-omyl. Ich najvhodnejšia hodnota sa líši od robota k robotovi, od okolností, v ktorých je robot spustený, tiež od použitej platformy. Existujú však základné pravidlá, ktoré pomôžu s ich ladením [12]:

- Konštanty *Ki* a *Kd* sa na začiatku nastavia na nulovú hodnotu. Hodnotu *Kp* skúsime nastaviť na hodnotu 1 a sledujeme robota. Cieľom je, aby robot sledoval čiaru, aj keď sa pohybuje s osciláciou. Ak robot prejde a stratí čiaru, hodnotu *Kp* je potrebné znížiť. Ak robot nie je schopný sledovať zákrutu, hodnotu *Kp* treba zvýšiť.
- Keď už robot ako tak dokáže sledovať čiaru, priradíme konštante *Kd* hodnotu 1 (*Ki* na chvíľu vynechajme). Hodnotu *Kd* budeme zvyšovať, kým uvidíme, že robot sa už pohybuje plynulejšie. Vieme, že derivácia odchýlky je menšia ako odchýlka samotná. Aby sme teda dosiahli zmysluplné hodnoty konštánt, D zložku musíme vynásobiť väčšou konštantou. *Kd* bude teda oveľa väčšia ako *Kp*. Príliš vysoká hodnota *Kd* však zapríčiní agresívne riadenie robota.
- Akonáhle je robot pomerne stabilný pri sledovaní čiary, priradíme konštante *Ki* hodnotu menšiu ako 1. Čím je táto hodnota vyššia, tým rýchlejšie sa robot dostane na čiaru. Ak je však *Ki* príliš vysoká, robota sa bude trhavo pohybovať doprava a doľava.
- Ak už robot dokáže sledovať čiaru s dobrou presnosťou, môžeme zvýšiť rýchlosť a uvidíme, či je stále schopný sledovať čiaru. Rýchlosť ovplyvňuje PID regulátor. So zmenou rýchlosti sa vyžaduje aj preladenie konštánt [12,13].

Kód, ktorý bol použitý pri sledovaní čiary jedným svetelným senzorom predstavuje nižšie uvedený Algoritmus 5.4. Tento algoritmus bol overovaný na reálnom robotovi (Obr. 5.22, Obr.5.23). V nxcSimulátori ho testovať nebolo možné, pretože tento simulátor predstavuje ideálne prostredie, v ktorom svetelný senzor rozlišuje dve hodnoty; 100 – biela farba a 0 – čierna farba. To znamená, že `error` by vychádzal stále záporný a robot by sa točil do jednej strany. Na sledovanie čiernej čiary v nxcSimulátori môžeme teda použiť Algoritmy 5.1 a 5.2.

Algoritmus 5.4 je rozdelený na dve časti. Pod každou časťou je jej vysvetlenie. Prvá časť algoritmu predstavuje kalibráciu senzora. Kalibráciou rozumieme prejdenie robota čiernou čiarou na malom úseku, počas ktorého robot zaznamená najvyššiu (biela farba) a najmenšiu (čierna farba) hodnotu intenzity osvetlenia. Tieto dve hodnoty potom spriemeruje a získa `setpoint`, ktorého hodnota sa počas sledovania čiary nemení, pretože kalibráciu sme vykonávali vždy iba na začiatku pred tým, ako

robot začal sledovať čiaru. Kalibrácia je vhodná z dôvodu meniacich sa podmienok, v ktorých je robot spustený. To znamená, že inú hodnotu `setpointu` dostaneme v rozsvietenej miestnosti ako v miestnosti s denným svetlom.

Druhou časťou Algoritmu 5.4 je samotný PID algoritmus. Hodnoty konštánt *Kp, Ki* a *Kd* uvedené v kóde v našom prípade zabezpečili plynulé sledovanie čiary. Treba podotknúť, že tie isté hodnoty konštánt nemusia fungovať v prípadoch iných robotov a trás, pretože iný systém (iná konštrukcia robota) spravidla potrebuje aj iný regulátor. So správne odhadnutými konštantami by však tento kód mal zabezpečiť pomerne plynulé sledovanie čiernej čiary v prípade NXT robota s jedným svetelným senzorom.

Obr. 5.22. a Obr. 5.23: NXT robot s jedným svetelným senzorom

```
int setpoint = 50;
    void calibrate(){
    int max = 0, min = 100;
    TextOut(0, LCD_LINE1, "Place the robot ");
    TextOut(0, LCD_LINE2, "in front of the");
    TextOut(0, LCD_LINE3, "line.");
    TextOut(0, LCD_LINE5, "Press the");
    TextOut(0, LCD_LINE6, "ORANGE button");
    until(ButtonPressed(BTNCENTER, false));
    ClearScreen();
    TextOut(20, LCD_LINE4, "CALIBRATING");
    for(int i = 0; i < 800; i++){
        OnFwd(OUT_AB, 20);
        if(max < Sensor(S3)) max = Sensor(S3);
        if(min > Sensor(S3)) min = Sensor(S3);
        Wait(1);
    }
    Off(OUT_AB);
    ClearScreen();
    setpoint = (max + min) / 2;
    until(ButtonPressed(BTNCENTER, true)){
        TextOut(0, LCD_LINE2, "White = "); NumOut(50, LCD_LINE2, max);
        TextOut(0, LCD_LINE4, "Black = "); NumOut(50, LCD_LINE4, min);
        TextOut(0, LCD_LINE6, "offset = "); NumOut(50, LCD_LINE6, setpoint);
    }
    ClearScreen();
}
```

(1) (2) (3) (4) (5) (6) (7)

Algoritmus 5.4, 1.časť: Kalibrácia senzora

Podrobné vysvetlenie prvej časti Algoritmu 5.4:

(1) Po zadefinovaní premennej `setpoint` sme vytvorili metódu `calibrate`. V nej sme definovali premenné `max` a `min`.

(2) Pomocou príkazu `TextOut` sme na LCD obrazovku robota nechali vypisovať text, podľa ktorého máme robota postaviť pred čiaru a stlačiť oranžové tlačidlo robota.

(3) Po stlačení oranžového tlačidla sa text na obrazovke zmaže a začína kalibrácia.

(4) Vo for-cykle sa maximálna hodnota nameraná senzorom počas kalibrácie zapíše do premennej `max` a minimálna do premennej `min`. Robot sa pohybuje dopredu pomocou `OnFwd` príkazu rýchlosťou 20.

(5) Príkaz `Off` robota zastaví a `ClearScreen` zmaže text na obrazovke.

(6) `Setpoint` získame spriemerovaním maximálnej a minimálnej hodnoty.

(7) Po stlačení oranžového tlačidla sa na displeji vypíšu hodnoty intenzity osvetlenia, ktoré senzor považuje za bielu, čiernu a `offset=setpoint` a pomocou `ClearScreen` zmaže text na obrazovke.

Kalibráciu by sme mohli vykonávať aj počas sledovania čiary, a to napríklad tak, že by sme nechali kontrolovať čas (napr. 1 hod). Po uplynutí tohto času by sa robot na chvíľu zastavil na čiare, vykonala by sa kalibrácia a robot by pokračoval ďalej v sledovaní čiary.

```
30 task main(){
31     SetSensorLight(S3);
32     calibrate();
33
34     float Kp = 2.1;
35     float Ki = 0.001;
36     float Kd = 50;
37
38     int error;
39     int integral = 0;
40     int derivative = 0;
41     int lastError = 0;
42     int actual_value;
43
44     int powerA;
45     int powerB;
46
47     int power = 40;
48     int output;
49     while(true){
50         actual_value = Sensor(S3);
51         error = setpoint - actual_value;
52         integral = integral + error;
53         derivative = error - lastError;
54         output = Kp*error + Ki*integral + Kd*derivative; // dt=1
55         lastError = error;
56         powerA = power - output;
57         powerB = power + output;
58         if(powerA > 100) {powerA = 100;}
59         if(powerA < 0) {powerA = 0;}
60         if(powerB > 100) {powerB = 100;}
61         if(powerB < 0) {powerB = 0;}
62         OnFwd(OUT_A, powerA);
63         OnFwd(OUT_B, powerB);
64         NumOut(50, LCD_LINE2, actual_value);
65     }
66
67 }
```

(1) (2) (3) (4) (5) (6)

Algoritmus 5.4, 2.časť: PID algoritmus pre sledovanie čiernej čiary jedným svetelným senzorom.

Podrobné vysvetlenie druhej časti Algoritmu 5.4:

(1) V `task main` definujeme senzor a zavoláme metódu `calibrate`, ktorou zistíme `setpoint`.

(2) Postupne si zadefinujeme všetky premenné, ktoré budeme potrebovať. Konštanty sme odhadli podľa pravidiel v kapitole 5.3.2 Nastavenie PID regulátora. Sú typu *float*, pretože používame desatinné čisla. Ostatné premenné sú typu *int*, keďže ich hodnoty sú celé čisla.

(3) Vo while cykle sa vykonávajú príkazy, ktorými získame `actual_value, error`, integračnú, derivačnú zložku, `output` a `lastError`. Hodnota `dt` je 1 ms, čo znamená, že robot reaguje každú milisekundu. Tým že `dt` je rovné 1, vo vzorcoch pre I a D zložku, ako je možné vidieť v ukážkovom Algoritme 5.3, nevystupuje.

(4) `PowerA` a `powerB` predstavujú rýchlosť ľavého a pravého motora. `PowerA` predstavuje pohyb robota doľava, `powerB` doprava Oba motory pracujú súčasne. Robot sa teda pohybuje do jednej strany vľavo alebo vpravo podľa toho, v ktorom motore je vyššia rýchlosť. Otáčanie robota do strany rýchlejším motorom je kompenzované pomalším motorom.
Hodnotu `output` v prípade `powerA` odčítavame, pretože keď je senzor mimo čiernej čiary, `output` je záporný. Aby sme teda nezískali zápornú rýchlosť, `output` musíme odčítať. Ak je output záporný, rýchlosť motora `powerA` je vyššia ako `powerB`. Ak sa teda robot vychýli od čiary smerom doprava (na bielu farbu), bude sa točiť doľava rýchlosťou `powerA`, čím odchýlenie vykompenzuje. Takým istým princípom pracuje aj motor na opačnej strane, pohybujúci sa rýchlosťou `powerB`.

(5) Pomocou príkazu `if` je ošetrená rýchlosť, pretože náš robot dokáže ísť maximálnou rýchlosťou 100 a minimálne nulovou rýchlosťou. Je to vec hardvéru. NXT motory sú obmedzené rýchlosťou v intervale <0,100>.
Ak by teda vypočítané rýchlosti `powerA` a `powerB` mali hodnotu vyššiu ako 100, robot sa bude pohybovať rýchlosťou 100. Ak bude rýchlosť menšia od nuly, bude sa pohybovať nulovou rýchlosťou do danej strany.

(6) Príkaz `OnFwd` zapne motory, ktoré sa pohybujú rýchlosťou regulovanou PID regulátorom súčasne dopredu. V tomto prípade príkaz `OnFwd` nemôžeme zapísať ako `OnFwd(OUT_AB, rýchlosť)`, ako to bolo pri kalibrácii, pretože tu má každý motor robota inú rýchlosť.
Aktuálnu hodnotu intenzity osvetlenia sme nechali vypísať príkazom `NumOut`.

Pozorovania zistené počas testovania algoritmu spracúva Tab. 5.2 a video publikované na youtube [C]. V Tabuľke sú uvedené aj možné riešenia našich konkrétnych situácií.

Tab. 5.2: Správanie sa robota (Obr.5.15, Obr.5.16, Obr. 5.17) v závislosti od konštánt *Kp, Ki* a *Kd* a od rýchlosti. Návrh zlepšenia jednotlivých situácií.

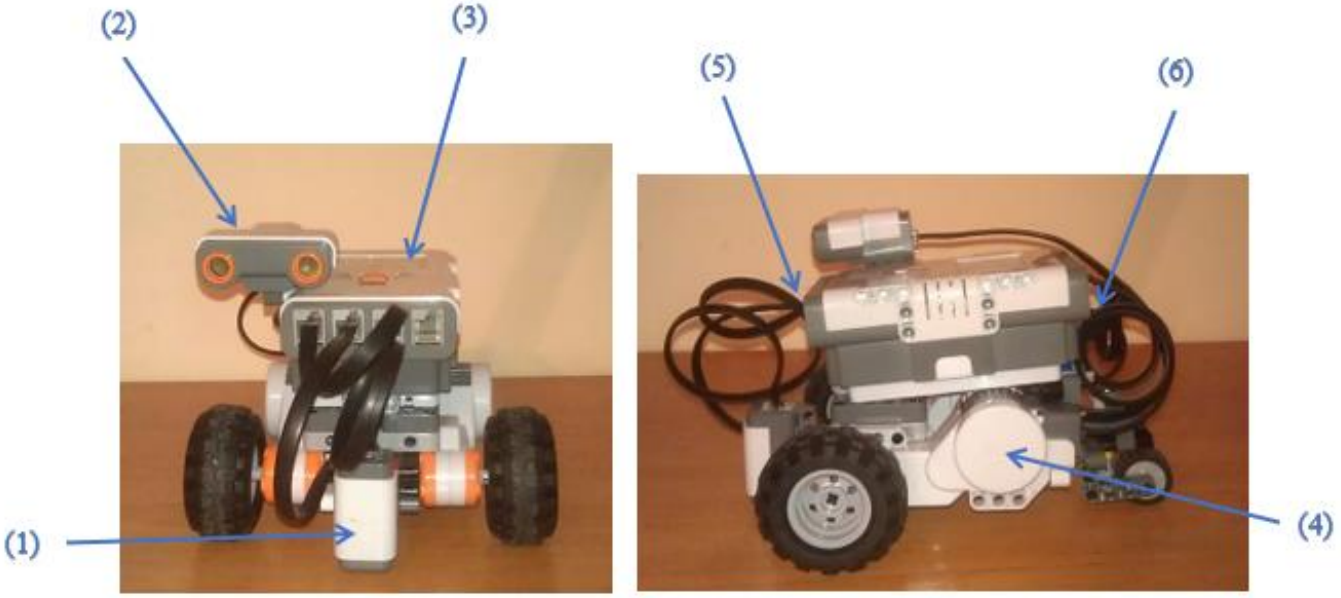
Kp	***Ki***	***Kd***	**Rýchlosť**	**Pozorovanie**	**Riešenie**
1.5	0	0	40	Robot prejde rovnú čiaru plynule a hladko. Nedokáže prejsť minimálne zatočenie.	Postupné zvyšovanie *Kp*. Čím vyššie *Kp*, tým ostrejšie pohyby.
5	0	0	40	Podobné s predchádzajúcou situáciou, ale robot vybočí v zákrute.	Postupné znižovanie *Kp*. Čím nižšie *Kp*, tým hladšie pohyby.
1.7	0	0	30	Robot prešiel plynule celú trasu.	
2.1	0.001	10	40	Robot dokáže prejsť celú trasu. Osciluje (pôsobenie I zložky).	Vykompenzovanie oscilácie zvýšením *Kd*.
2.1	0.001	50	45	Robot prejde celú trasu aj po zvýšení rýchlosti. V zákrutách stále pozorujeme oscilovanie. Na rovnej čiare ide plynulejšie.	
2.1	0.001	60	45	Robot nezvládne zatáčku.	Zníženie *Kd*.
2.1	0.0008	50	40	Robot nezvládne zatáčku.	Zvýšenie *Ki* alebo zníženie *Kp* a *Kd*.

Z tabuľky vyplýva, že najhladšie čiaru prešiel robot s P regulátorom rýchlosťou 30. Keďže táto rýchlosť sa nám zdala príliš nízka, bolo potrebné zaviesť I a D zložku. Bez nich by robot nevytočil zákruty s rýchlosťou vyššou ako 35. Pri experimentovaní s konštantami boli použité pravidlá opísané v kapitole 5.3.2 Nastavenie PID regulátora. Pri robotovi, ktorý sledoval čiaru najvhodnejšie a zároveň šiel rýchlosťou vyššou ako 30, boli použité tieto konštanty: *Kp*=2.1, *Ki*=0.001, *Kd*=50. Rýchlosť robota bola 45.

5.3.3 Vyhýbanie sa prekážkam

Robot (Obr. 5.23, Obr. 5.24) sledujúci čiernu čiaru jedným senzorom, ktorý je schopný vyhnúť sa prekážke je naprogramovaný Algoritmom 5.5. Tento algoritmus je tiež rozdelený do dvoch častí a bol skúšaný reálnym riadením, ktorého video je uverejnené na webe [D]. Prvá časť algoritmu sa týka kalibrácie senzora, je teda totožná s prvou časťou Algoritmu 5.4. V druhej časti je použitý PID

algoritmus z Algoritmu 5.4, doplnený o metódu `obstacle`, ktorá robotovi zabezpečí, aby sa vyhol prekážke. K druhej časti je vysvetlená iba tá časť kódu, ktorá sa nevyskytuje v Algoritme 5.4.



Obr. 5.24 a Obr. 5.25: NXT robot so svetelným a ultrazvukovým senzorom.

Popis jednotlivých častí robota z Obr. 5.24 a 5.25:

(1): svetelný senzor

(2): ultrazvukový senzor

(3): kocka

(4): motor

(5): vstupné porty, port 1, 2, 3 a 4 pre napájanie senzorov

(6): výstupné porty A,B,C pre motory

```
30 void obstacle(){
31       OnFwd(OUT_A, 40); OnRev(OUT_B, 40); Wait(800);
32       OnFwd(OUT_AB, 40); Wait(1500);
33       OnFwd(OUT_B, 40); OnRev(OUT_A, 40); Wait(800);
34       OnFwd(OUT_AB, 40); Wait(3000);
35       OnFwd(OUT_B, 40); OnRev(OUT_A, 40); Wait(800);
36       while(Sensor(S1) > setpoint){
37             OnFwd(OUT_AB, 40);
38       }
39       OnFwd(OUT_A, 30); Off(OUT_B); Wait(950);
40       while(Sensor(S1) > setpoint){
41             OnFwd(OUT_A, 30); OnRev(OUT_B, 30);
42       }
43       Off(OUT_AB);
44 }
45
46 task main(){
47     SetSensorLight(S1);
48     SetSensorLowspeed(S2);
49     calibrate();
50
51     float Kp = 1.7;
52     float Ki = 0.001;
53     float Kd = 2.0;
54
55     int error;
56     int integral = 0;
57     int derivative = 0;
58     int lastError = 0;
59     int actual_value;
60
61     int powerA;
62     int powerB;
63
64     int power = 45;
65     int output;
66     while(true){
67         if(SensorUS(S2) < 20){
68            obstacle();
69         }
70         actual_value = Sensor(S1);
71         error = setpoint - actual_value;
72         integral = integral + error;
73         derivative = error - lastError;
74         output = Kp*error + Ki*integral + Kd*derivative; //dt=1
75         lastError = error;
76         powerA = power - output;
77         powerB = power + output;
78         if(powerA > 100) {powerA = 100;}
79         if(powerA < -100) {powerA = -100;}
80         if(powerB > 100) {powerB = 100;}
81         if(powerB < -100) {powerB = -100;}
82         OnFwd(OUT_A, powerA);
83         OnFwd(OUT_B, powerB);
84     }
85
86 }
```

(1) (2) (3) (4) (5) (6) (7)

Algoritmus 5.5, 2.časť: PID algoritmus a metóda `obstacle`.

Podrobné vysvetlenie druhej časti Algoritmu 5.5:

(1) Ľavý motor je v porte A a pravý motor v porte B. Na základe vysvetlených príkazov v Tab. 4.4 vieme, že robot prekážku obíde zľava [D].
(2) Pokiaľ je senzor mimo čiary, robot sa pohybuje oboma motormi dopredu.
(3) Keď už robot narazí na čiaru, pootočí sa trocha doľava, aby vyšiel z čiary. Tento príkaz používame preto, lebo robot je po vykonaní predchádzajúcich príkazov kolmo na čiaru. My ho však chceme na čiaru vyrovnať tak, aby bol pozdĺž nej.
(4) Pokiaľ je senzor mimo čiary, tak sa na ňu natočí, aby bol pozdĺž čiary.
(5) Príkazom `Off` sme vypli oba motory.
(6) Nastavili sme si svetelný a ultrazvukový senzor. Ultrazvukový senzor sa nastavuje príkazom `SetSensorLowSpeed`. Tento senzor sa ďalej v kóde používa ako `SensorUS`. Ultrazvukový senzor meria vzdialenosť v centimetroch.
(7) Keď ultrazvukový senzor nameria vzdialenosť prekážky menšiu ako 20 cm, vykonajú sa príkazy v metóde `obstacle`.

PID algoritmus nezaručuje efektívne výsledky iba samotnou implementáciou kódu. Vyžaduje neustále vylepšovanie v závislosti od okolností. Keď už je algoritmus správne navrhnutý, prináša výnimočné výsledky [11].
Na záver kapitoly treba dodať, že nie je vždy nutné implementovať všetky tri zložky PID regulátora. Ak implementácia iba P alebo PI zložiek prináša dobré výsledky, zvyšné časti môžeme preskočiť. Ako sme mohli vidieť, v našom prípade robot dokázal sledovať čiaru iba s P zložkou. Po práci s PID algoritmom môžeme zhrnúť jeho výhody a obmedzenia pri sledovaní čiary:

Výhody PID

- Sledovanie čiary iba jedným senzorom.
- Plynulý priebeh sledovania čiary.

Obmedzenia PID

- Prerušená čiara

  Prerušovanú čiaru na Obr. 5.17 nie je možné použitím PID algoritmu prejsť, pretože keď sa robot nachádza mimo čiary (biela farba), robot sa točí smerom k čiare (k čiernej farbe). To znamená, že na prerušovanej čiare by sa točil dookola.

- Pravouhlé zákruty

  Pravouhlú zákrutu by konkrétne náš robot s jedným svetelným senzorom nezvládol. Z čiary by vyšiel a točil by sa na mieste.

- Pomalšie reakcie [11].

  V porovnaní s robotmi využívajúcimi dva alebo štyri senzory, je robot riadený PID algoritmom pomalší. Porovnajme si maximálne rýchlosti robotov, ktoré sme využívali v tejto práci:

  Robot so štyrmi svetelnými senzormi: 100

  Robot s dvoma svetelnými senzormi: 50

  Robot s jedným svetelným senzorom: 45.

6. Záver

Táto práca bola písaná z pozície študenta pre študentov. Účelom práce bolo v prvej kapitole uviesť čitateľa do problematiky NXT robotov, zoznámiť ho so základnými súčiastkami stavebnice Lego Mindstorms a stručne opísať programovací jazyk NXC (Not eXactly C), ktorým je možné tieto roboty programovať. Ďalej je v práci predstavený nxcEditor, prostredie pre vytváranie a simuláciu NXC algoritmov prostredníctvom nxcSimulátora. Pre študentov, ktorí ešte nemajú skúsenosti s operačným systémom Linux je vo forme tutoriálu v tretej kapitole spísaná inštalácia tohto prostredia, ktorá je dostupná aj na webe [E].

Druhá a tretia kapitola predstavujú úvod do nxcEditora a nxcSimulátora, kde sa študenti zoznámia s ich jednotlivými časťami a funkcionalitami. V práci je predstavený simulátor kvôli tomu, že z finančného a časového hľadiska je oveľa efektívnejšie využívať program, v ktorom zároveň môžeme simulovať riadený proces. Je jednoduchšie si program napísať, kedykoľvek podľa potreby upraviť a vyskúšať v simulátori na rozdiel od častokrát pracného konštruovania lego robotov. nxcSimulátor je mimoriadne vhodný hlavne na študijné účely.

Študentom, ktorí si chcú osvojiť programovacie zručnosti pri tvorbe algoritmov pomocou NXC sú venované štvrtá a piata kapitola. Na začiatku sú podrobne opísané pravidlá písania algoritmov a ich štruktúry. Pomocou prehľadu príkazov dostupných pre nxcSimulátor sme si mohli vysvetliť algoritmus sledovania čiernej čiary pomocou svetelných senzorov. Posledná časť piatej kapitoly je vhodná pre tých, ktorí majú záujem doplniť svoje znalosti o PID regulátory, pomocou ktorých dokáže robot sledovať čiaru jediným senzorom. Keďže PID algoritmy nebolo možné simulovať v simulátori, študenti majú v tejto časti možnosť vidieť demonštráciu reálneho riadenia. Simulácie algoritmov a videoukážky reálneho riadenia sú k dispozícii na internete [A, B, C, D].

.

Zoznam použitej literatúry

[1] History of LEGO Robotics [online]. Dostupné na https://www.lego.com/en-us/mindstorms/history

[2] BURNETT, Wayne. Robotics Competitions [online]. 2015-09-30. Dostupné na http://www.legoengineering.com/robotics-competitions/

[3] BÁTORA, Vladimír. Lego Mindstorms [online]. 2010-09-10. Dostupné na http://www.posterus.sk/?p=8553&output=pdf

[4] BENEDETTELI, Daniele. Programming LEGO NXT Robots using NXC, version 2.2, 2007.

[5] Robotik-AG [online]. Dostupné na http://www.gymnasium-rahden.de/robotik-ag.html

[6] nxcEditor [online]. Dostupné na www.nxceditor.sourceforge.net

[7] HANSEN, John. NXC Programmer's Guide [online]. 2010-05-30
The NXC Guide. Dostupné na www.bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf

[8] HANSEN, John. NXC Programmer's Guide [online]. 2013-02-18. Dostupné na http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html

[9] BAKOŠOVÁ, Monika. FIKAR, Miroslav. Riadenie procesov. STU 2012, ISBN 978-80-227-3763-0.

[10] Rozvetvené regulačné obvody [online]. Dostupné na http://www.kirp.chtf.stuba.sk/moodle/pluginfile.php/2671/mod_resource/content/0/predn6.pdf

[11] LENČUCHA, Andrej. Fungovanie PID [online]. 2012-07-30. Dostupné na http://rgt.sk/2012/fungovanie-pid/

[12] PID CONTROL [online]. Dostupné na https://www.robotix.in/tutorial/avr/pid/

[13] SITOULA, Ashim. PID Tutorials for Line Following [online]. 2014-01-06. Dostupné na http://www.robotshop.com/letsmakerobots/pid-tutorials-line-following

Prílohy

[A] Simulácie nxc algoritmov v nxcSimulátori pre sledovanie čiernej čiary:
DUDÁKOVÁ, Zuzana. nxcSimulator line follower robot [online]. 2016-12-08. Dostupné na https://www.youtube.com/watch?v=7LLGx-pPWK4&feature=youtu.be
[B] Reálne riadenie robota sledujúceho čiernu čiaru dvoma svetelnými senzormi:
DUDÁKOVÁ, Zuzana. Line follower with 2 light sensors [online]. 2017-04-11. Dostupné na https://www.youtube.com/watch?v=obgKHVvCX4E
[C] Reálne riadenie robota sledujúceho čiernu čiaru jedným svetelným senzorom:
DUDÁKOVÁ, Zuzana. PID line follower [online]. 2016-12-08. Dostupné na https://www.youtube.com/watch?v=k66aVoxfi04
[D] Reálne riadenie robota sledujúceho čiernu čiaru s prekážkou:
DUDÁKOVÁ, Zuzana. pid line follower and obstacle avoidance [online]. 2017-04-27. Dostupné na https://www.youtube.com/watch?v=mBEqSQcp6HE
[E] nxcEditor a jeho inštalácia:
DUDÁKOVÁ, Zuzana. nxcEditor and its step by step installation [online]. 2016-11-12. Dostupné na http://www.kirp.chtf.stuba.sk/~kvasnica/blog/nxceditor/