ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra řídící techniky

Uživatelský manuál (soubor řešených úloh) pro předmět ROBOTI

User manual (set of solved tasks) for the subject ROBOTS

Bakalářská práce

Studijní program: Kybernetika a Robotika

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

Jan Daněk

Praha 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Daněk** Jméno: **Jan** Osobní číslo: **491847**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Uživatelský manuál (soubor řešených úloh) pro předmět ROBOTI

Název bakalářské práce anglicky:

User manual (set of solved tasks) for the subject ROBOTS

Pokyny pro vypracování:

1. Seznamte se s možnostmi robota LEGO Mindstorms NXT (současný stav, HW a SW vybavení).
2. Vytvořte výukové materiály (soubor řešených úloh) v různých programovacího prostředích (např. NXT-G, NXC a jiné) pro předmět ROBOTI.
3. Případně prozkoumejte možnosti programování LEGO Mindstorms NXT v programovacím jazyce Python.

Seznam doporučené literatury:

[1] http://bricxcc.sourceforge.net/
[2] https://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html
[3] https://lego.zcu.cz/web/moduly-systemu-lego-nxt
[4] https://en.wikipedia.org/wiki/Lego_Mindstorms_NXT

Jméno a pracoviště vedoucí(ho) bakalářské práce:

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

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

Datum zadání bakalářské práce: **10.01.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Martin Hlinovský, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Michael Šebek, DrSc. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

Datum převzetí zadání	Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Prohlášení

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

V Praze dne…… ……..

Jan Daněk

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu bakalářské práce Ing. Martinu Hlinovskému, PH.D. za jeho podnětné rady, připomínky a materiály poskytnuté mi při zpracování této práce. Zároveň vedoucímu práce děkuji za čas, který si na mě našel během společných konzultací, a za čas potřebný k vytvoření připomínek, které vedly ke zvýšení kvality této práce.

Abstrakt

Cílem této práce je formulovat praktická cvičení pro předmět ROBOTI, který je vyučován na Fakultě elektrotechnické ČVUT v Praze v prvním ročníku bakalářského studia programu Kybernetika a Robotika.

Základ této práce tvoří programovatelná robotická sada Lego Mindstorms® NXT (Lego Group, 2006), konkrétně sety 9648 a 9797, které jsem detailně představil se zaměřením na jejich důležité komponenty. Dále jsem prozkoumal možnost programování hlavního mikrokontroleru kostky NXT populárním a lehce pochopitelným programovacím jazykem Python 2, jako alternativu ke známějšímu na C založenému programovacímu jazyku NXC. Pro konverzi kódu mezi jazykem Python a NXC jsem použil PyNXC converter, ke kterému jsem dopsal vlastní grafické prostředí. Během tvorby této práce jsem narazil na několik chyb v PyNXC a tyto chyby jsem zdokumentoval a částečně opravil.

Nakonec jsem vytvořil pět cvičení pro předmět ROBOTI. Kladl jsem důraz na prohledávání stavového prostoru a ukázal jsem několik příkladů zpětnovazební regulace s použitým regulátorů typu PID. Tyto úkoly jsou podrobně popsány a byl kladen důraz na opakovatelnost řešení jednotlivých úkolů studenty již zmíněného předmětu ROBOTI. Těchto pět úkolů je naprogramovaných ve třech programovacích jazycích NXT-G, NXC a Python 2. Zdrojové kódy pro řešení těchto úkolů jsou rovněž detailně popsány v této práci.

Klíčová slova

Lego Mindstorms NXT, PD regulátor, PID regulátor, Bludiště, Deep search, Not Exactly C, NXT-G, PyNXC, regulační houpačka, sledování čáry, objíždění překážky, Python 2.7, gyroskopický senzor.

Abstract

The aim of this bachelor thesis is the formulation of practical exercises for the course ROBOTI, which is being held regularly at the Czech Technical University in Prague for the first-year students of the bachelor study program Kybernetika a Robotika.

As the basis I choose the programmable robotics kit Lego Mindstorms® NXT (Lego Group, 2006), namely the sets 9648 and 9797, which I introduce in a greater detail and pay a special attention to its crucial components. Further, I investigate the possibility of programming the main microcontroller NXT brick with the popular and intuitive programming language Python 2 as an alternative to the broadly used C-based programming language NXC for the Lego Mindstorms NXT. For the conversion of the Python source code, I use the PyNXC converter, which I extend with my own graphical user interface. Several error in the original PyNXC converter's code are discovered during this process and are corrected.

Finally, I create five textbook exercises for the students with the intention to teach them about the search of the state-space and several cases of the back-loop control with various application of the PID regulator. The exercises are formulated a detailed and easy to follow manner and prepared in three various programming languages Python 2, NXC and NXT-G, which makes them along with the attached source codes easily reproducible by students of various programming background.

Key Words

Lego Mindstorms NXT, PID controller, PD controller, Maze, Deep search, Not Exactly C, NXT-G, PyNXC, angle level holding, line tracking, obstacle avoidance, Python 2.7, gyroscopic sensor.

Obsah

1. Úvodní kapitola ... 17
1.1. Cíle ... 17
1.2. Úvod do bakalářské práce ... 17
1.3. Historie ... 18
1.4. Využití stavebnice z řady LEGO Mindstorms ve světe ... 18
1.5. Definice pojmů použitých v této práci ... 19
2. Představení programovatelné sady Lego Mindstorms NXT ... 19
2.1. Řídící jednotka (kostka) NXT ... 20
2.2. Programovací prostředí ... 21
2.2.1. NXT-G ... 21
2.2.2. NXC (Not Exactly C) ... 21
2.2.3. PyNXC ... 22
2.3. Nahrávání programů do NXT kostky z prostředí PyNXC ... 22
2.3.1. První způsob převodu PyNXC na NXC ... 23
2.3.2. Druhý způsob převodu PyNXC na NXC ... 23
2.3.3. Třetí způsob převodu PyNXC na NXC ... 23
2.4. Aktuátory ... 24
2.4.1. Servomotory ... 25
2.4.1.1. Příkazy pro práci se servomotory v PyNXC ... 25
2.4.1.2. Ukázkový kód v programovacím prostředí PyNXC ... 26
2.4.2. Displej ... 27
2.4.2.1. Programovací prostředí PyNXC a LCD ... 27
2.4.2.2. Ukázkový kód v jazyce PyNXC ... 28
2.4.3. Reproduktor ... 28
2.4.3.1. Programovací prostředí PyNXC a reproduktor ... 28
2.4.3.2. Ukázka použití reproduktoru v programovacím jazyku PyNXC ... 29
2.5. Senzory ... 29
2.5.1. Senzor zvuku ... 29
2.5.1.1. Programovací prostředí PyNXC a senzor zvuku ... 30
2.5.1.2. Ukázka odčítání hodnoty ze senzoru v jazyce PyNXC ... 30
2.5.2. Dotykový senzor ... 31
2.5.2.1. Programovací prostředí PyNXC a dotykový senzor ... 31
2.5.2.2. Ukázkový program funkčnosti dotykového senzoru v jazyce PyNXC ... 31
2.5.3. Senzor světla ... 32
2.5.3.1. Programovací prostředí PyNXC a senzor světla: ... 32

2.5.3.2. Ukázka kódu pro měření odraženého světla v jazyce PyNXC. ... 33
2.5.4. Ultrazvukový senzor ... 33
2.5.4.1. Programovací prostředí PyNXC a ultrazvukový senzor ... 34
2.5.4.2. Ukázka snímání hodnoty vzdálenosti pomocí ultrazvukového senzoru v jazyce PyNXC ... 35
2.5.5. Gyroskopický senzor ... 35
2.5.5.1. Programovací prostředí a gyroskopický senzor ... 37
2.5.5.1.1. NXT-G ... 37
2.5.5.1.2. NXC a PyNXC ... 39
2.5.5.1.3. Ukázkový kód na určení hodnoty offset (řádek 13-18) a určení natočení senzoru (řádek 22-32) v jazyce NXC ... 39
2.5.5.1.4. Ukázkový kód na určení hodnoty offset (řádek 9-12) a určení natočení senzoru (řádek 18-27) v jazyce PyNXC ... 40
2.5.5.2. Užitečná rada o nulování úhlu ... 41
2.5.5.3. Užitečná rada o stanovení doby běhu smyčky ... 41
2.6. Závěr této kapitoly ... 42
3. Sborník úkolů ... 42
3.1. Konstrukce robota ... 42
3.1.1. Obecné požadavky na roboty ... 42
3.1.2. Programování robota ... 43
3.1.3. Napájení robota ... 43
3.2. Úkol 1: Sumo ... 43
3.2.1. Zadání ... 43
3.2.2. Rozbor problému ... 45
3.2.3. Konstrukce ... 46
3.2.4. Příprava konstant ... 46
3.2.5. Programování ... 47
3.2.5.1. Pseudo-kód/diagram ... 47
3.2.5.2. NXT-G ... 48
3.2.5.3. NXC ... 49
3.2.5.4. PyNXC ... 52
3.2.6. Použití gyroskopického senzoru ... 54
3.2.7. Závěr ... 54
3.3. Úkol 2: Bludiště ... 54
3.3.1. Zadání ... 54
3.3.2. Rozbor problému ... 56
3.3.3. Konstrukce ... 56

3.3.4. Sledování stěny ... 57
3.3.4.1. Programovací část ... 57
3.3.4.1.1. Nxt-G Pseudokód ... 58
3.3.4.1.2. NXT-G ... 58
3.3.4.1.3. Pseudokód pro NXC a PyNXC ... 59
3.3.4.1.4. NXC ... 60
3.3.4.1.5. PyNXC ... 63
3.3.5. Deep First Search (DFS) ... 66
3.3.5.1. Pseudokód, jak DFS algoritmus funguje ... 68
3.3.5.2. Pseudokód pro programovaní robota ... 69
3.3.5.2.1. Prohledávací část pro prozkoumání bludiště ... 69
3.3.5.2.2. Program jízdy bludištěm ... 70
3.3.5.3. Programování v NXC ... 70
3.3.5.3.1. Prohledávací část pro prozkoumání bludiště ... 70
3.3.5.3.2. Program jízdy bludištěm ... 72
3.3.5.3.3. Prohledávání bludiště DFS v programu PyNXC ... 76
3.3.6. Závěr ... 77
3.4. Úkol 3: Sledování čáry pomocí regulátorů ... 77
3.4.1. Zadání ... 77
3.4.1.1. Popis soutěžní plochy ... 77
3.4.1.2. Finální hodnocení ... 78
3.4.2. Rozbor problému ... 78
3.4.3. Konstrukce ... 79
3.4.4. Zig-Zag ... 79
3.4.5. Programování Zig-Zag regulátoru ... 81
3.4.5.1. Pseudokód ... 81
3.4.5.2. NXT-G ... 82
3.4.5.3. NXC ... 82
3.4.5.4. PyNXC ... 83
3.4.6. Regulátory PID/PD ... 83
3.4.6.1. Proporcionální část ... 85
3.4.6.2. Derivační složka. ... 86
3.4.6.3. Integrální složka ... 86
3.4.6.4. Anti-windup ... 87
3.4.6.5. PID regulátor ... 87
3.4.6.6. Nastavení konstant $KP, KD\ a\ KI$ ... 87

3.4.7. Programování PD regulátoru ... 87
3.4.7.1. Pseudokód ... 88
3.4.7.2. NXT-G ... 88
3.4.7.3. NXC ... 89
3.4.7.4. PyNXC ... 89
3.4.8. Programování PID regulátoru ... 90
3.4.8.1. Pseudokód ... 90
3.4.8.2. NXT-G ... 91
3.4.8.3. NXC ... 92
3.4.8.4. PyNXC ... 93
3.4.9. Nastavení konstant ... 94
3.4.9.1. Zkušební a chybová metoda ... 95
3.4.9.2. Metoda Zeigler-Nichols ... 95
3.4.9.3. Naměřené hodnoty ... 96
3.4.10. Teoretické porovnání jednotlivých typů regulátorů z hlediska jejich výhod a nevýhod 97
3.4.11. Závěr ... 98
3.5. Úkol 4: Objíždění překážky na dráze ... 98
3.5.1. Zadání ... 99
3.5.1.1. Popis soutěžní plochy ... 99
3.5.1.2. Popis překážky ... 99
3.5.2. Rozbor problému ... 99
3.5.3. Konstrukce ... 100
3.5.4. Programování ... 101
3.5.4.1. Pseudokód ... 101
3.5.4.2. NXT-G ... 101
3.5.4.3. NXC ... 102
3.5.4.4. PyNXC ... 104
3.5.5. Obecná doporučení k naprogramování PID regulátoru ... 104
3.5.6. Závěr ... 104
3.6. Úkol 5: Robot číšník ... 105
3.6.1. Zadání ... 106
3.6.2. Rozbor problému ... 106
3.6.3. Konstrukce ... 107
3.6.4. Programování PD regulátoru ... 107
3.6.4.1. Pseudokód ... 108
3.6.4.2. NXC ... 108

3.6.4.3. PyNXC ... 111
3.6.5. Nastavení konstant PD regulátoru ... 112
3.6.6. Závěr PD regulátoru ... 113
3.6.7. Vícestavová regulace ... 114
3.6.8. Závěr ... 115
4. Problémy, se kterými se autor práce setkal při řešení úloh ze sborníku úkolů ... 115
4.1. Nemožnost napsání všech kódů v NXT-G ... 115
4.2. Problémy spojené s nedokonalostí ultrazvukového senzoru ... 116
4.3. Nastavení konstant u PID regulátoru ... 116
4.4. Složitost kódu u PID regulátoru při objíždění překážky ... 117
4.4.1. Kód napsaný v jazyce NXC určený pro úkol objíždění překážky společně s PID regulátorem ... 117
4.5. Nemožnost vytváření 2D polí v NXC ... 118
4.6. Závěr této kapitoly ... 119
5. Změření času nejlepších průjezdů jednotlivých drah ... 119
6. Úprava a popis jazyka PyNXC ... 120
6.1. Jiné možnosti programování NXT kostky v jazyce Python ... 121
6.2. Oprava některých chyb ... 121
6.2.1. Oprava vkládání chybných definic na začátek překládaného souboru ... 121
6.2.2. Oprava špatné inicializace proměnných a funkcí ... 121
6.2.3. Oprava chování smyčky for ... 121
6.3. Dopsání uživatelského rozhraní Sekretar pro práci s prostředím PyNXC ... 122
6.4. Závěr této kapitoly ... 123
Závěr bakalářské práce ... 123
Citace ... 125
Odkazy ... 127
Seznam obrázků ... 127
Seznam Tabulek ... 129
Digitální přílohy ... 129

1. Úvodní kapitola

1.1. Cíle

Cílem této práce je prozkoumat stav a možnosti počítačem programovatelné stavebnice Lego Mindstorms NXT 2.0 za účelem tvorby výukových materiálů k předmětu ROBOTI, který je povinným předmětem v prvním semestru bakalářského programu Kybernetika a robotika. Stěžejní výstup této práce tvoří Sborník řešených úloh, který má být použitý jako podpůrný materiál pro výuku již zmíněného předmětu. Tento sborník bude obsahovat několik úkolů, které budou vyřešeny v několika programovacích jazycích. Bude zde představený i teoretický rozbor úkolu, včetně nastínění jeho technického řešení.

K podpoře, nejen distanční výuky vzniknou videa, které budou ukazovat vzorové řešení jednotlivých úloh.

Jako posledním úkolem mé práce je prozkoumat možnost programování Lego Mindstorms NXT pomocí programovacího jazyka Python.

1.2. Úvod do bakalářské práce

Stavebnice Lego Mindstorms NXT patří do velké skupiny Lego Mindstorms. Původ těchto stavebnic je ve spolupráci Lega a MIT, kdy vznikla první počítačově programovatelná lego kostka [1]. Jelikož se těšila velké popularitě, nejenom v univerzitních kruzích, vznikaly postupně předchůdkyně této populární stavebnice. Vývoj se ale u NXT nezastavil a pokračuje neméně úspěšnou řadou Lego Mindstorms EV3.

Během této práce si ve stručnosti projdeme historii této řady stavebnic. Podíváme se rovněž na využití této stavebnice ve světě.

Během této práce vzniká sborník řešených úloh, který má za cíl sloužit jako podpůrný materiál k výuce předmětu ROBOTI. V tomto sborníku budou předvedeny a popsány jednotlivé části stavebnice Lego Mindstorms NXT včetně popisu jejich programování v jazyce PyNXC. Zároveň bych rád upozornil, že tato práce si nedává za úkol vyřešit jednotlivé úkoly ve sborníku co možná nejlépe nebo nejefektivněji. Tato řešení by neodpovídali zaměření tohoto sborníku, jelikož studenti prvního ročníku nemají potřebné znalosti, k pochopení těchto složitějších řešení. Společně s řešením této práce vznikají videa, která slouží k lepšímu pochopení jednotlivých úkolů ze sborníku řešených úkolů.

Během tvorby této práce jsem vyšel především z programovacích jazyků NXT-G, NXC a PyNXC. Samotné úlohy, vznikají jako variace na již známé a otestované úlohy v Lego komunitě.

Další kapitola zahrnuje tvorbu podkladů k navržení bodového hodnocení tohoto předmětu tak, aby předmět odpovídal svému účelu jakožto motivačního předmětu.

V poslední části této práce se pak pouštím do posledního úkolu ze zadání, a to prozkoumání možnosti programování NXT pomocí programovacího jazyka Python. Zde jsem objevil jedinou možnost, jak pomocí tohoto jazyka kostku programovat. Tento programovací jazyk však nefunguje dokonale, a proto bych rád opravil jeho největší chyby tak, aby byl využitelný k programování jednotlivých úkolů ze sborníku úloh. Zároveň bych zde rád dopsal uživatelské prostředí, které usnadní nahrávání souborů .py přímo do kostky NXT.

1.3. Historie

V této kapitole se stručně seznámíme s historií vývojové řady Lego Mindstorms. Tuto podkapitolu zde zařazuji, protože si myslím, že by každý student měl ve stručnosti vědět historií věcí, o kterých se učí, a měl by znát jejich předchůdce a popřípadě vědět o jejich novějších verzích.

Časová osa vývoje Lego Mindstorms:

1932 – v Dánsku založena společnost Lego [2]

1980 – společnost Lego zakládá oddělení pro vzdělávací produkty [1]

1984 – počátek spolupráce firmy Lego a americké univerzity MIT [1]

1986 – uvedení první kostky programovatelné počítačem [3]

1998 – představen předchůdce Lega Mindstorms NXT, a to Lega Mindstorms RCX [3]

2006 – první verze stavebnice Lega Mindstorms NXT [3]

2009 – uvedena na trh stavebnice Lega Mindstorms NXT 2.0 [3]

2013 – představena stavebnice Lega Mindstorms EV3 a společnost LEGO ukončuje výrobu stavebnice Lega Mindstorms NXT 2.0 [3]

2020 – představena sada Robot Invertor [4]

2020 – na trh přichází stavebnice LEGO Education Spike Prime [5]

2021 – ukončena výroba Lega Mindstorms EV3

1.4. Využití stavebnice z řady LEGO Mindstorms ve světe

Stavebnice Lego Mindstorms se ve světě těší velké popularitě. Využívá se především k edukačním účelům na základních, středních i vysokých školách. Tato stavebnice podporuje kreativitu studentů a rozvíjí jejich vlastnosti, jako jsou například abstraktní a logické myšlení [6]. Řada institucí, jako například Temple University [7], nabízí výukové kempy zaměřené na pochopení základů práce s touto stavebnicí, čímž ve studentech rozvíjí kreativitu a zájem o technické obory [6].

Řada univerzit po celém světě vyučuje pomocí těchto stavebnic programování, nebo například automatické řízení, ale i jiné předměty zaměřené na kybernetiku či robotiku. Příklady takových univerzit:

- Univerzita Palackého v Olomouci [8]
- Technická univerzita v Liberci [9]
- České vysoké učení technické v Praze (ČVUT)
- Massachusetts Institute of Technology [10]

Ve stavbě robotů se pořádá po celém světě řada soutěží. Během těchto soutěží se studenti zdokonalují v logickém myšlení, spolupráci s kolegy, nebo si často během nich osvojí základy programování. Příkladem takových soutěží je například:

- Robosoutěž pořádaná Českým vysoký učením technickým
- FIRST Lego League Challenge [11]
- FIRST Tech Challenge [12]

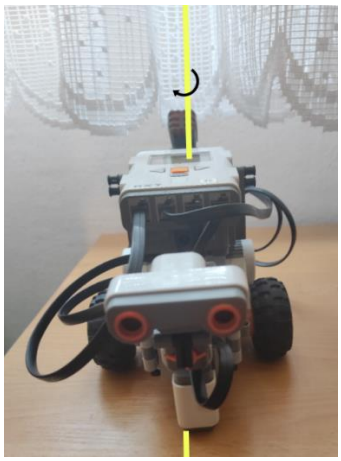
Samozřejmě množství univerzit a soutěží pořádané s touto stavebnicí je více, ale i tento stručný výčet ukazuje, jak je tato stavebnice ve světě populární.

V současné době řeší vysoké školy problém s nedostatkem stavebnic EV3 z důvodu ukončení jejich výroby. Nástupce EV3 LEGO Education Spike Prime se ukázal jako nevhodný pro práci na vysokých školách, nebo k pořádání soutěží typu Robosoutěž. Tato nová stavebnice cílí především na žáky na základních školách.

1.5. Definice pojmů použitých v této práci

Robot: Za robota se obecně považuje spojení motorů a senzorů, které do určité míry samostatnosti řídí chod robota. V této práci si dovolím slovem robot nazývat takové zařízení, které je uzpůsobené k řešení jednotlivých úkolů, tak jak budou představeny v kapitole Sborník úkolů.

Otočení robota o předem daný úhel: V tomto textu předpokládám, že otočením robota o 90° doprava, se myslí situace, kdy se robot natočí tak, aby při dalším pohybu směrem dopředu jel doprava oproti původní poloze před začátkem otáčení. Ilustrace otočení a následného pohybu je na Obr. 1 a Obr. 2. Na Obr. 1 je vidět, podle které osy se robot otáčí.



Obr. 1 Otáčení robota o určitý úhel doprava

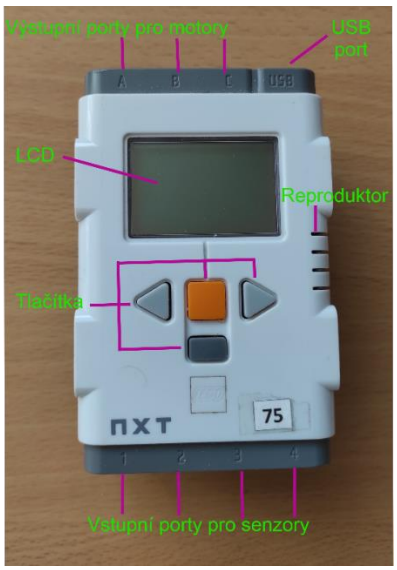
Obr. 2 ilustrace trajketorie robota, při otočení o 90° doprava

2. Představení programovatelné sady Lego Mindstorms NXT

V této kapitole si dávám za cíl seznámit studenty s Lego Mindstorms NXT, včetně možnosti jeho programování v programovacím prostředí PyNXC. Popíšu zde obecně stěžejní senzory a aktuátory, které se nachází v již zmíněné sadě. Podrobněji zde bude rozebrán gyroskopický senzor, který není přímo součástí setů Lego Mindstorms NXT, ale musí se dokoupit zvlášť od výrobců třetích stran.

2.1. Řídící jednotka (kostka) NXT

Pomyslným mozkem robota je řídící jednotka NXT, které budeme říkat kostka NXT. Na Obr. 3 je vidět, jak kostka vypadá včetně popisu jejích jednotlivých částí.



Obr. 3 Řídící jednotka (kostka) NXT

Středobodem kostky je displej s rozlišením 100 x 64 pixelů, který umí zobrazovat pouze černou a bílou barvu. Dále se na kostce nachází 4 tlačítka. Šipky slouží pro orientaci v menu. Oranžové tlačítko funguje na potvrzování a tmavě šedé tlačítko slouží jako zpětné tlačítko.

V horní části kostky se nachází 3 porty pro připojení servomotorů a USB-B pro připojení kostky NXT k počítači. V dolní části kostky se pak nachází 4 porty pro připojení senzorů. K propojení senzorů nebo servomotorů s kostkou slouží 6 žilové ploché kabely zakončené konektory RJ12. Detail těchto kabelů je na Obr. 4 a Obr. 5. Za zmínku stojí, že kostka mimo jiné obsahuje i reproduktor, takže je na ní možné vytvářet nejrůznější zvuky. [13]

Obr. 4 Pohled na propojovací kabely

Obr. 5 Detailní pohled na konektor

Na opačné straně kostky, než je displej, se nachází slot pro baterie. My v našem případě budeme využívat akumulátorovou baterii o maximálním napětí 7,4 V.

Asi největší nevýhoda, se kterou se částečně budeme setkávat, je malý prostor pro ukládání souborů v NXT kostce. NXT kostka obsahuje paměť typu flash, která je podobná SD kartě, ale její nevýhodou je, malá kapacita paměti a to pouze 256 KB. V této paměti se uchovává vše od firmwaru až po samotné

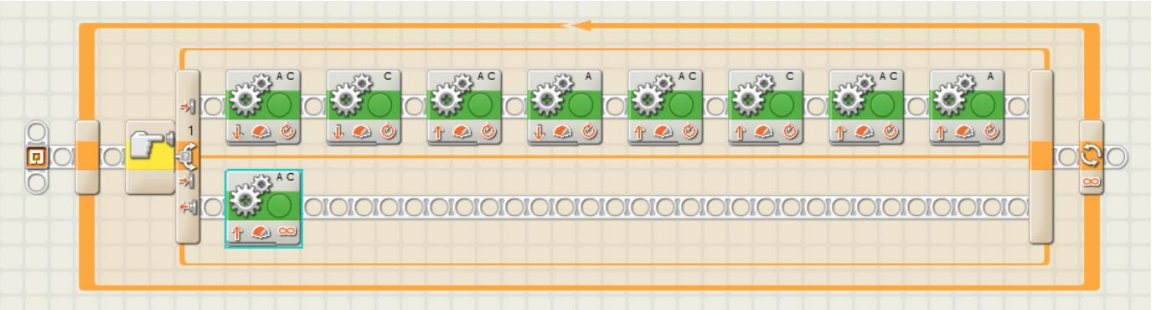
programy. Velice často dochází k jejímu zaplnění. Tento problém lze řešit vymazáním již nepoužívaných programů.

2.2. Programovací prostředí

S programovacími jazyky u NXT přichází jeden problém, a to ten, že na kostku lze uložit omezené množství programovacích jazyků. Odborně řečeno: „na kostce je potřeba přehrát firmware“, což je vlastně operační systém kostky. Takže některé programy jsou psané na jeden operační systém, a naopak jiné programy jsou psány zase pro jiný operační systém. Pro nás nejdůležitější firmware se jmenuje LEGO Mindstorms NXT firmware v aktuální verzi 1.31. Tento firmware je do kostky NXT nahrán již v továrně během její výroby. Jako příklad jiného firmware použitelného u této kostky uvedu LeJOS 0.9.1. Tento firmware slouží k programování NXT kostky například pomocí Javy nebo grafického jazyka Enchanting. Všechny programovací jazyky použité v tomto sborníku jsou kompatibilní s oficiálním firmwarem od společnosti Lego.

2.2.1. NXT-G

Grafické programovací prostředí vyvíjené přímo firmou Lego, často se udává pod zkráceným názvem NXT nebo NXG. Toto prostředí se hodí především pro jednodušší projekty, případně testování jednotlivých součástí robota. Složitější programy se v něm stávají značně nepřehledné. Programování v tomto prostředí probíhá pomocí přetahování a spojování bloků. Každý blok plní svůj úkol. Kupříkladu jsou zde bloky pro pohyb servomotorů, odčítání hodnot ze senzorů nebo blok pro základní matematické operace. Existují zde i bloky pro smyčky nebo bloky pro podmínky. [13]



Obr. 6 Ukázka kódu v NXT-G

2.2.2. NXC (Not Exactly C)

Jak již název napovídá, nejedná se přímo o C, ale o jazyk podobný jazyku C. Od jazyka C zdědil syntaxi. Naopak, co ho odlišuje od jazyka C, je reprezentace některých proměnných z důvodu firmwaru v kostce NXT [14]. Jedná se o textový jazyk a po NXT-G je to nejrozšířenější způsob, jak programovat roboty obsahující klasický firmware NXT. Je vhodný i pro velké a složité projekty. Jedna z jeho největších nevýhod je nedostatečná dokumentace. Rovněž tento jazyk nepodporuje veškeré funkcionality robota NXT, se kterými je možné se setkat v NXT-G. [13]

```
1  int light;
2
3  task main(){
4      SetSensorLight(IN_3);
5      while(true){
6              light = Sensor(IN_3);
7              if(light < 50){
8                      OnFwd(OUT_A, 80);
9                      OnFwd(OUT_C, 0);
10             }else{
11                     OnFwd(OUT_C, 80);
12                     OnFwd(OUT_A, 0);
13             }
14     }
15 }
```

Obr. 7 Ukázka kódu v jazyce NXC

2.2.3. PyNXC

Jedná se o jazyk NXC, ale v syntaxi Pythonu. Před samotným nahráním programu do kostky se musí přeložit do formátu NXC a teprve pak se může nahrát do NXT kostky. Na druhou stranu je tento jazyk mnohem čistější a přehlednější pro řadu studentů. Jedním z důvodu je, že s Pythonem pracují studenti již od začátku studia a jeho základy jsou jim přednášeny v paralelně běžícím předmětu Algoritmy a programování (ALP). [13], [15]

```
1  def main():
2      SetSensorLight(IN_3)
3      while 1:
4          light = Sensor(IN_3)
5          if light < 50:
6              OnFwd(OUT_C, 50)
7              OnFwd(OUT_A, 0)
8          elif light > 48:
9              OnFwd(OUT_A, 50)
0              OnFwd(OUT_C, 0)
1          else:
2              OnFwd(OUT_A, 50)
3              OnFwd(OUT_C, 50)
4
```

Obr. 8 Ukázka kódu v jazyce PyNXC

2.3. Nahrávání programů do NXT kostky z prostředí PyNXC

Nahrávání programu do kostky NXT se uskutečňuje pomocí prostředí BricxCC. Pro samotnou kompilaci mezi kódem PyNXC a NXC je potřeba nejprve stáhnout kompilátor PyNXC, který bude dostupný ke stažení ze stránek předmětu v Moodlu. Kompilátor je rovněž možné stáhnout volně na internetu, zde ovšem dochází k problému, kdy takto stažený kompilátor obsahuje řadu chyb a nedokonalostí v předkladu. Některé z těchto nedokonalostí autor tohoto textu opravil (dostupné v Moodlu).

Pro spuštění kódu pro kompilaci je třeba nainstalovat Python 2.7.x. V současné chvíli máme tři možnosti, jak daný kód v jazyce PyNXC převést do jazyka NXC. Kódy napsané v jazyce PyNXC se dají psát v libovolném textovém editoru, či jakémkoliv pythonovském IDLE. Tyto programy však musí obsahovat příponu .py v názvu souboru.

2.3.1. První způsob převodu PyNXC na NXC

V tomto případě nejprve umístíme požadovaný text pro kompilaci do složky *..\pynxc-master\pynxc\tests\in* a následně ve složce *..\pynxc-master\pynxc\tests* spustíme script tests.py. Pokud program neobsahuje chyby do stejné složky, do které jsme umístili program .py, se nám přeloží kód do jazyka NXC. Takto zkompilovaný kód nahrajeme do kostky NXT způsobem používaným pro nahrávání souboru NXC do NXT kostky pomocí prostředí BricxCC.

Tento způsob kompilace má tu nevýhodu, že nám program neukáže případné chyby v jazyce PyNXC. Pokud se v kódu vyskytne syntaktická chyba, program není přeložen, ale kompilátor nám nenahlásí chybu. Pokud je ve složce více programů .py a v některém je syntaktická chyba, dochází k tomu, program s chybou a všechny následující zůstanou nepřeloženy.

2.3.2. Druhý způsob převodu PyNXC na NXC

Tento způsob nám nedává možnost kompilovat několik kódů z jazyka PyNXC najednou. Na druhou stranu nám program kompilátoru ukáže případné syntaktické chyby v jazyce PyNXC, které brání přeložení souboru do jazyka NXC.

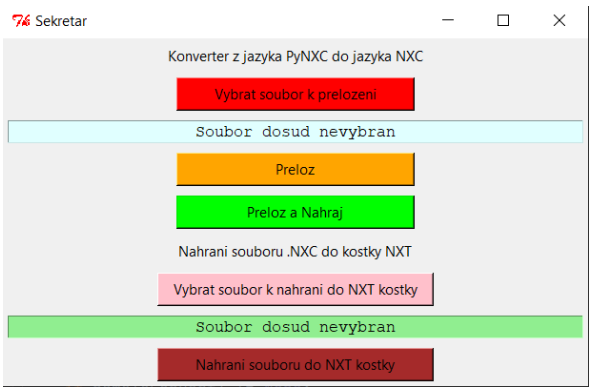
Program v jazyce .py umístíme do složky *..\pynxc-master\pynxc*. Následně spustíme, například z prostředí PyCharm program pynxc.py s argumentem -c „jmeno prekldaneho souboru.py“. Po spuštění tohoto programu se nám kód .py přeloží na kód .nxc. Pokud se v kódu .py vyskytují syntaktické chyby program nás na ně upozorní. Po odstranění chyb můžeme kód .py zkusit znovu přeložit. Přeložený soubor .nxc se nám vygeneruje přímo ve složce *..\pynxc-master\pynxc*. Nahrávání přeloženého kódu do kostky NXT probíhá již standartním způsobem pomocí programu BricxCC.

Mimo jiné tento způsob překladu slibuje, že dokáže nahrát přeložený soubor přímo do kostky NXC bez použití prostředníka v podobě BricxCC. Bohužel se tento způsob nahrávání autorovi tohoto textu nepodařilo zprovoznit.

2.3.3. Třetí způsob převodu PyNXC na NXC

Tato možnost je dopsaná autorem této práce, proto se nenachází ve verzi stažené na stránkách nebo Gituhubu autora PyNXC. Ke správnému fungování tohoto kódu je nutné nainstalovat knihovnu pyautogui a na počítači mít nainstalovaný program BricxCC.

Bylo vytvořeno uživatelské rozhraní Sekretar, které má za úkol zjednodušit překládání PyNXC do NXC a jeho následné nahrávání do kostky NXC. Toto uživatelské rozhraní nalezneme na Obr. 9.



Obr. 9 Uživatelské rozhraní

Toto prostředí obsahuje několik tlačítek, které slouží k ulehčení práce s prostředím PyNXC. Pomocí prvního červeného tlačítka **Vybrat soubor k prelozeni** se vybírá soubor PyNXC s příponou .py v kterékoliv složce v počítači. Ve světle modrém rámečku se vypíše cesta k námi vybranému souboru. Následně pomocí oranžového tlačítka **Preloz** se soubor přeloží do formátu .nxc. Přeložený soubor nalezneme ve stejné složce jako byl původní soubor .py. Zelené tlačítko **Preloz a nahraj** pak soubor .py přeloží do jazyka .nxc a následně jej nahraje do NXT kostky. V obou případech, když dojde k chybě v překladu vypíše se nám chybová hláška s příslušným řádkem v souboru .py, který brání správnému překladu.

Pomocí růžového tlačítka **Vybrat soubor k nahrani do NXT kostky** se vybírá soubor .nxc k nahrání do NXT kostky. Cesta k vybranému souboru se následně vypíše ve světlezeleném rámečku. Pomocí hnědého tlačítka **Nahrani souboru do NXT kostky** dochází k nahrání vybraného souboru .nxc do NXT kostky.

Nahrávání souborů do NXT kostky program Sekretar probíhá přes USB kabel připojený do zapnuté kostky NXT.

2.4. Aktuátory

Mezi aktuátory patří vše, s čím může kostka NXT pohybovat nebo na tom něco ukazovat, případně přehrávat zvuk. Budeme se tedy zabývat servomotory, displejem a reproduktorem.

V této kapitole bude popsané, jak se jednotlivé aktuátory programují v programovacím prostředí PyNXC a je zde ukázán i příklad jejich použití.

Robot, na kterém se ukazují jednotlivé aktuátory a v další kapitole senzory, je zachycený na Obr. 10 a Obr. 11..

Obr. 10 Foto ukázkového robota z hora

Obr. 11 Foto ukázkového robota zepředu

2.4.1. Servomotory

Napřed si vysvětleme, jaký je rozdíl mezi pojmy servomotory a motory. Pro nás je nejpodstatnější rozdíl, že u servomotorů lze nastavit přesnou polohu natočení hřídele. [16] U obyčejných motorů tohoto ovšem nelze dosáhnout. V textu budu rozlišovat dva případy podle toho, jestli uvažuji přesné ovládání natočení hřídele, tj. servomotory, u kterých počítám s nastavováním přesné polohy hřídele, a motory, u kterých nás polohování hřídele nebude zajímat.

Ke každé kostce NXT lze připojit až 3 servomotory k portům označeným písmeny A, B, C v horní části NXT kostky. Rychlost, s jakou se hřídele u servomotorů otáčí, je závislá na napětí baterie, která kostku napájí. Čím vyšší je napětí na baterii, tím se servomotory otáčí rychleji. Když se baterie vybíjí, klesá na ní napětí, a tím se servomotory otáčejí pomaleji, proto doporučuji při práci s robotem baterii pravidelně nabíjet, i když není zcela vybitá.

Nevýhoda motorů je, že se kvůli výrobní nedokonalosti každý otáčí jinou rychlostí. To vede k tomu, že robot, pohybující se pomocí dvou motorků, není schopný jet zcela rovně bez použití dalšího senzoru, který by rychlost otáčení motorů korigoval.

Obr. 12 Servomotor

2.4.1.1. Příkazy pro práci se servomotory v PyNXC

V této podkapitole si vysvětlíme základní práci se servomotory v prostředí PyNXC.

Porty, na kterých se nachází jednotlivé servomotory, se v tomto jazyce označují následovně:

Port A – OUT_A, Port B – OUT_B, Port C – OUT_C

Porty se případně dají spojovat mezi sebou následovně:

Porty A a B – OUT_AB, Porty B a C – OUT_BC, Porty A a C – OUT_AC

Příkazy, pomocí kterých programujeme chování motorů:

- OnFwd(Port, Rychlost) – roztáčí servomotor „dopředu“
  - **Port** – označuje port, na kterém se nachází servomotor
  - **Rychlost** - (0-100) označuje jakou rychlostí se servomotor roztáčí
- OnRev(Port, Rychlost) – roztáčí servomotor „dozadu“
  - **Port** – označuje port, na kterém se nachází servomotor
  - **Rychlost** - (0-100) označuje jakou rychlostí se servomotor roztáčí
- Off(Port) – zastavuje běh motorku
  - **Port** – označuje port na kterém se nachází servomotor

- RotateMotor(Port, Rychlost, Úhel) – Otáčí servomotorem o určitý úhel
    - **Port** – označuje port, na kterém se nachází servomotor
    - **Rychlost** - (0-100) označuje jakou rychlostí se servomotor roztáčí
    - **Úhel** – Natavuje úhel ve stupních, o který se má servomotor pootočit
- OnFwdSync(Porty, Rychlost, Otáčení) – Synchronizovaný pohyb 2 servomotorů
    - **Port** – označuje porty na kterých se nachází servomotory
    - **Rychlost** - (0-100) označuje jakou rychlostí se servomotory roztáčí
    - **Otáčení** – (-100-100) označuje, jak se dané dva motorky budou otáčet rychle případně, kam bude robot zatáčet
    - **Příklad použití:** OnFwdSync(OUT_AC, 40, -20)

Doporučuje se, při synchronizaci pohybu dvou servomotorů, připojit servomotory na porty B a C. Tuto skutečnost se bohužel autor tohoto textu dozvěděl, až po naprogramování úkolů z této bakalářské práce. Nicméně i při připojení servomotorů na porty A a C a jejich následné synchronizaci, dosáhnul autor uspokojivých výsledků.

Rovněž se doporučuje při změně směru pohybu servomotorů tyto nejprve zastavit, například příkazem Off(), chvíli počkat pomocí příkazu Wait() a teprve poté změnit směr. Je to z důvodu, že při prudké změně směru pohybu motorů by se informace o této změně nemusela správně interpretovat a motor by se zastavil.

2.4.1.2. Ukázkový kód v programovacím prostředí PyNXC

```
def main():
    OnFwd(OUT_AC, 75) #robot jede dopredu motory AC rychlosti 75
    Wait(2000) #robot spi po 2 sekundy
    OnRev(OUT_C, 50) #robot se bude otacet proti smeru hodinovych rucicek
    OnFwd(OUT_A, 50)
    Wait(1000) #robot spi po 1 sekundu
    Off(OUT_AC) #robot se zastavi
    Wait(3000) #robot spi po 3 sekundy
    RotateMotor(OUT_AC, 25, 360) #robot pootoci kola o 360 stupnu
```

Příkaz Wait(**Čas**) uspí program na čas zvolený v argumentu **Čas**, který se zadává v milisekundách. Neznamená to ovšem, že se servomotory přestanou točit. Právě naopak motory se po tento čas budou otáčet v původním směru, příkazy OnFwd() a OnRev() nastaví nový směr otáčení a po uplynutí příkazu Wait(**Čas**) se začnou otáčet v novém směru.

Kód vykoná následující akce. Robot nejprve pojede rovně rychlostí 75 % jeho maximální rychlosti po dobu 2 sekund, protože roztáčí dopředu servomotory připojené na portech A a C příkazem OnFwd(OUT_AC, 75). Následně se bude robot na místě otáčet proti směru hodinových ručiček po dobu 1 sekundy. Roztáčí totiž servomotory na portech A a C proti sobě příkazy OnFwd(OUT_A, 50) a OnRev(OUT_C, 50). Následně bude robot stát 3 sekundy na místě, jelikož se vypnul chod motorů na portech A a C příkazem Off(OUT_AC). Jako poslední robot otočí svými koly o 360° ve směru hodinových ručiček. Robot se posune dopředu.

O kolik robot s koly o průměru 55 mm popojede dopředu při otočení hřídele o 360° je jednouchý výraz

$$d = 2 \cdot \pi \cdot \frac{55}{2} = 173\ mm. \qquad (1)$$

Na rozdíl od klasického Pythonu, musíme v PyNXC klasifikovat, která funkce je hlavní a má se spouštět na začátku běhu programu. Funkci, kterou chceme spustit na začátku programu, pojmenujeme jako main(). Hlavička hlavní funkce, která odpovídá task main() z prostředí NXC, je def main().

2.4.2. Displej

NXT je osazeno LCD bez jakéhokoliv podsvícení. Zkratka LCD označuje Liquid Crystal Display česky pak displej z tekutých krystalů. Proto by se správně nemělo říkat LCD displej. LCD má rozlišení 100x64 pixelů neboli 20x8 znaků. V základní konfiguraci, kterou budeme používat, umí zobrazovat pouze černou a bílou barvu.

Obr. 13 LCD

2.4.2.1. Programovací prostředí PyNXC a LCD

PyNXC zapisuje text na jednotlivé řádky LCD. Řádky jsou reprezentovány LCD_LINEX, kde X označuje číslo řádku. Tedy příkaz LCD_LINE3 bude zapisovat na řádek 3.

Seznam příkazů k práci s displejem:

- TextOut(PoziceX, PoziceY, „TEXT“, clear) – zapisuje text na displej
  - **PoziceX** – (1-20) určuje sloupec, od kterého se začíná text zapisovat
  - **PoziceY** – (LCD_LINE1 - LCD_LINE8) určuje řádek, na který se text zapisuje
  - **TEXT** – určuje text, který se bude na kostku zapisovat. Text se zapisuje do uvozovek.
  - **Clear** – (true/false) volitelný parametr (bez zadání nastaveno na false), který určuje, zda se má displej přemazat.
- NumOut(PoziceX, PoziceY, číslo, clear) – zapisuje číslo na displej
  - **PoziceX** – (1-20) určuje sloupec, od kterého se začíná číslo zapisovat
  - **PoziceY** – (LCD_LINE1 - LCD_LINE8) určuje řádek, na který se číslo zapisuje
  - **číslo** – určuje číslo v libovolném formátu, které se bude na displej zapisovat
  - **clear** – (true/false) volitelný parametr (bez zadání nastaveno na false), který určuje, zda se má displej přemazat
- GraphicOut(PoziceX, PoziceY, JménoSouboru, clear) – kreslí obrázek ze souboru .ric na displej
  - **PoziceX** – (1-20) určuje sloupec, od kterého se začíná text zapisovat
  - **PoziceY** – (LCD_LINE1 - LCD_LINE8) určuje řádek, na který se text zapisuje
  - **JménoSouboru**– určuje jméno souboru, ze kterého se má obrázek načíst, tento soubor musí být nahrán v paměti NXT kostky

- **Clear** – (true/false) volitelný parametr (bez zadání nastaveno na false), který určuje, zda se má displej přemazat.

- ClearScreen() – smaže vše co se nachází na displeji

2.4.2.2. Ukázkový kód v jazyce PyNXC

```
def main():
    Hodnota_1 = 10 #hodnota, ktera se ukaze na displeji
    Hodnota_2 = 20 #hodnota, ktera se ukaze na displeji
    TextOut(10, LCD_LINE2, "Hodnota 1:", true) #vypise se: Hodnota 1 resetuje se obrazovka
    NumOut(70, LCD_LINE2, Hodnota_1, false) #vypise se ciselo odpovidajici Hodnota_1
    TextOut(10, LCD_LINE4, "Hodnota 2:", false) #vypise se: Hodnota 2 neresetuje se obrazovka
    NumOut(70, LCD_LINE4, Hodnota_2, false) #vypise se ciselo odpovidajici Hodnota_2
    Wait(10000) #program 10 sekund nebude nic delat
    ClearScreen() #smaze se co je na obrazovce
    GraphicOut(10, 10, "Swearing.ric") #nactu a zobrazim obrazek ze souboru Swearing.ric
    Wait(10000) #program 10 sekund nebude nic delat
```

Do proměnných Hodnota_1 a Hodnota_2 uložíme hodnoty 10 a 20. Následně vypíšeme na druhý řádek displeje „Hodnota 1: 10“ a na čtvrtý řádek: „Hodnota 2: 20“. Počkáme 10 sekund, abychom si výsledek prohlédli. Následně smažeme vše, co je na displeji a zobrazíme na něm obrázek Swearing.ric, který je uložený v paměti NXT kostky. Po vykreslení obrázku počkáme 10 sekund a program se sám ukončí.

2.4.3. Reproduktor

Jde o klasický elektrodynamický reproduktor, jaký známe například z mobilů nebo počítačů. V závislosti na programovacím prostředí dokáže přehrávat, jak tóny o určité frekvenci, tak i celé zvukové soubory ve vhodném formátu.

V této práci bude tento aktuátor využíván, na upozornění, že se s robotem něco děje, nebo ukončil provádění určité části úkolu.

2.4.3.1. Programovací prostředí PyNXC a reproduktor

Zde dochází k výjimce, kdy jazyk PyNXC zcela nepodporuje všechny příkazy z jazyka NXC. Problém nastává u příkazu SysSoundPlayFile(spfArgs), který se z jazyka PyNXC nesprávně překládá do jazyka NXC. Problém je zde s definováním struktury spfArgs.

- PlayTone(Frekvence, Čas) – přehraje tón o určité frekvenci
    - **Frekvence** – frekvence tónu
    - **Čas** – doba trvání tónu v milisekundách
- PlaySound(Makro) – Přehraje zvuk podle zadaného Makra
    - **Tabulka Maker**

aCode	***Resulting Sound***
SOUND_CLICK	Key click sound
SOUND_DOUBLE_BEEP	Double beep
SOUND_DOWN	Sweep down
SOUND_UP	Sweep up
SOUND_LOW_BEEP	Error sound
SOUND_FAST_UP	Fast sweep up

Tabulka 1: Makra pro jednotlivé zvuky

Následující příkaz není implementovaný v PyNXC, autor ho sem dává pro upozornění, aby nedocházelo k jeho nesprávnému použití v tomto programovacím prostředí.

- SysSoundPlayFile(spfArgs) – přehraje zvukový soubor ve formátu .rso uložený v NXT kostce, spfArgs se musí nastavit následovně:
    - U NXC je potřeba deklarovat strukturu SoundPlayFileType spfArgs;
    - spfArgs.Filename = „hello.rso“ určuje jméno souboru, k přehrání
    - spfArgs.Loop = false – určuje, zda se má soubor přehrávat pořád dokola
    - spfArgs.SoundLevel = 3 – určuje hlasitost souboru
    - V případě, že chcete tento příkaz použít u PyNXC musíte ručně nakopírovat na začátek přeloženého kódu následující deklaraci SoundPlayFileType spfArgs

2.4.3.2. Ukázka použití reproduktoru v programovacím jazyku PyNXC

```
def main():
    PlaySound(SOUND_UP) #zacene prehravat zvuk SOUND_UP
    Wait(500) #pocka 0.5 sekundy
    PlaySound(SOUND_CLICK) #zacene prehravat zvuk SOUND_CLICK
    Wait(500) #pocka 0.5 sekundy
    PlayTone(440, 2000) #zacne prehravat zvuk o frekveci 440 Hz po dobu 2 sekund
    Wait(2000) #pocka 2 sekundy
```

Program začne přehrávat zvuk, který se nachází pod makrem **SOUND_UP**, pomocí příkazu PlaySound(). Dávejte si pozor, program nečeká na dokončení přehrávání, proto se zde nachází příkaz Wait(). Dále se přehrává zvuk pod makrem **SOUND_CLICK**. Jako poslední se přehrává tón o frekvenci 440 Hz po dobu 2 sekund. Příkaz opět nezastaví běh programu, proto se zde nachází příkaz Wait(), díky kterému program skončí, až po přehrání daného tónu.

2.5. Senzory

Pomocí senzorů se robot dozvídá o prostředí okolo něho, jsou to vlastně jeho smysly. Obecně senzory převádí fyzikální veličiny na elektrický proud. Aby se tento elektrický proud mohl přenést do kostky NXT, musí být kostka a senzor propojený příslušným kabelem. My budeme pracovat především s pěti typy senzorů. Pro lepší představu jejich funkce se pokusím najít jejich analogii v říši zvířat. Rovněž se podíváme na to, jak robotovi říct, aby s tímto senzorem komunikoval a mohl z něho odečítat data v prostředí PyNXC.

Senzory vyráběné přímo firmou Lego se nachází ve standardizovaném krytu. To je z důvodu, aby bylo možné jednotlivé senzory na robotovi mezi sebou libovolně zaměňovat. Drobnou výjimku tvoří ultrasonický senzor, který má odlišný tvar. Způsob jeho připojení k samotnému robotovi je ovšem stejný, jako v případě ostatních senzorů.

2.5.1. Senzor zvuku

Senzor zvuku, funguje podobně jako uši. Každý zvuk, který tento senzor uslyší, převede na elektrický impulz a vyšle ho do NXT kostky. Měření tohoto impulzu probíhá ve dvou rozdílných jednotkách a to dB a dBA. Jednotkou dB jsou decibely, tedy logaritmicky vyjádřená úroveň hluku. Jednotkou dBA jsou opět decibely, ale od předchozích se liší váhováním podle lidského ucha. Výstup tohoto senzoru se udává v procentech, kdy 100 % přibližně odpovídá 90 dB nebo dBA. Výsledek 1 % odpovídá zhruba hodnotě 35 dB nebo dBA. [17]

Obr. 14 Senzor zvuku

2.5.1.1. Programovací prostředí PyNXC a senzor zvuku

Senzory se musí v tomto programovacím prostředí nainicializovat, neboli říct kostce NXT, na kterém portu se nachází jaký senzor. Senzor zvuku se inicializuje následujícím způsobem:

- SetSenzorSound(Port, dB)
  - **Port** označuje port na kostce NXT, na kterém je připojený senzor zvuku. Zapisuje se ve tvaru IN_X, kde X označuje číslo portu.
  - **dB** nastavuje v jakých jednotkách chceme úroveň zvuku měřit. Hodnota True znamená, že měření v dB, hodnota False pak značí měření v dBA.

Samotné odčítání dat ze senzoru se uskutečňuje následujícím způsobem:

- Sound = Senzor(Port)
  - Port označuje port na kostce NXT, na které je připojený senzor zvuku. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.
  - Příkaz senzor odečte hodnotu ze senzoru na příslušném portu a tu poté zapíše do proměnné Sound. Hodnoty odčítané ze senzoru jsou stejné, jako v případě programovacího jazyka NXT-G.

2.5.1.2. Ukázka odčítání hodnoty ze senzoru v jazyce PyNXC

```
2  def main():
3      SetSensorSound(IN_2, True) #inicializuje se senzor zvuku na portu 2, merici v dB, pro dBA je potreba argument zmenit na false
4      while(True): #nekonecna smycka
5          sound = Sensor(IN_2)  #nacte se hodnota ze senzoru na portu 2
6          TextOut(10, LCD_LINE2, "Hodnota hluku:", True) #vypise se: Hodnota hluku, přičmž se resetuje se obrazovka
7          NumOut(70, LCD_LINE3, sound, False) #vypise se ciselo odpovidajici sound
```

V programu se nejdříve inicializuje senzor zvuku na řádku 3. Tento senzor je připojený na portu 2. Zde je senzor inicializovaný na měření v dB, které ty jsou vyjádřeny v procentech. Tak jak je popsané na začátku kapitoly. Kdybychom chtěli přepnout na měření v dBA musíme na řádku 3 změnit v příkazu SetSensorSound() druhý argument na False. Dále se pak ve smyčce vykonává čtení hodnoty ze senzoru na portu 2 a výpis změřené hodnoty na displej společně s textem: „Hodnota hluku“. Všimněte si, že díky čtvrtému argumentu ve funkci TextOut() se displej před vypsáním nové hodnoty vymaže.

2.5.2. Dotykový senzor

Dotykový senzor by se dal připodobnit hmatu u lidí. Ve své podstatě je to ale tlačítko. Kdykoliv je tento senzor stlačený vysílá do kostky signál o tom, že je stlačený.

Oficiálně se tento senzor nazývá tlakový senzor, ovšem tento název přesně neodpovídá jeho funkci, jelikož vysílá pouze dvě úrovně signálu stisknuto/nestisknuto. Od tlakového senzoru bychom očekávali, že bude měřit hodnotu tlaku, která na něj působí. Což se v případě tohoto senzoru neděje. Proto si dovolím v tomto textu používat přesnější název dotykový senzor.

Obr. 15 Dotykový senzor

2.5.2.1. Programovací prostředí PyNXC a dotykový senzor

Senzor je nejprve potřeba nainicializovat. Tato inicializace se provádí následujícím příkazem:

- SetSenzorTouch(Port)
  - **Port** označuje port na kostce NXT, na kterém je připojený dotykový senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.

Odečítání hodnot ze senzoru se uskutečňuje následujícím příkazem:

- Touch = Senzor(Port)
  - **Port** označuje port na kostce NXT, na kterém je připojený dotykový senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.
  - Příkaz senzor odečte hodnotu ze senzoru na příslušném portu a tu poté zapíše do proměnné Touch. Hodnota je binární 1 nebo 0. 1, když je tlačítko stisknuté a 0, když je tlačítko rozpojené.

2.5.2.2. Ukázkový program funkčnosti dotykového senzoru v jazyce PyNXC

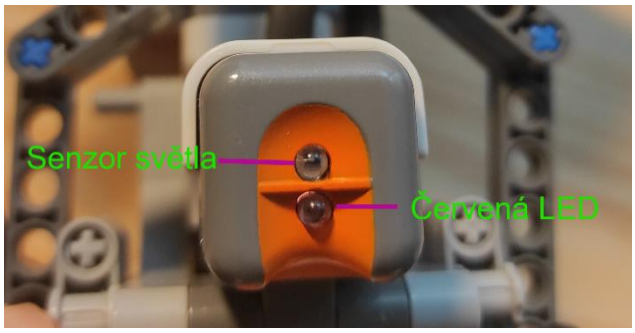
```
2  def main():
3      SetSensorTouch(IN_1) #inicializuje se dotykovy senzor na portu 1
4      while(True): #nekonecna smycka
5          touch = Sensor(IN_1)  #nacte se hodnota ze senzoru na portu 1
6          TextOut(10, LCD_LINE2, "Hodnota tlacitka:", True) #vypise se: Hodnota tlacitka, přičmž se resetuje se obrazovka
7          NumOut(70, LCD_LINE3, touch, False) #vypise se ciselo odpovidajici stavu zmacknuti tlacitka
```

Nejdříve se nainicializuje dotykový senzor, který je připojený na portu 1. V nekonečné smyčce se pak provádí načtení stavu senzoru na řádku 5. Dále pak výpis textu: „Hodnota tlačítka“ následovaná výpisem stavu senzoru.

2.5.3. Senzor světla

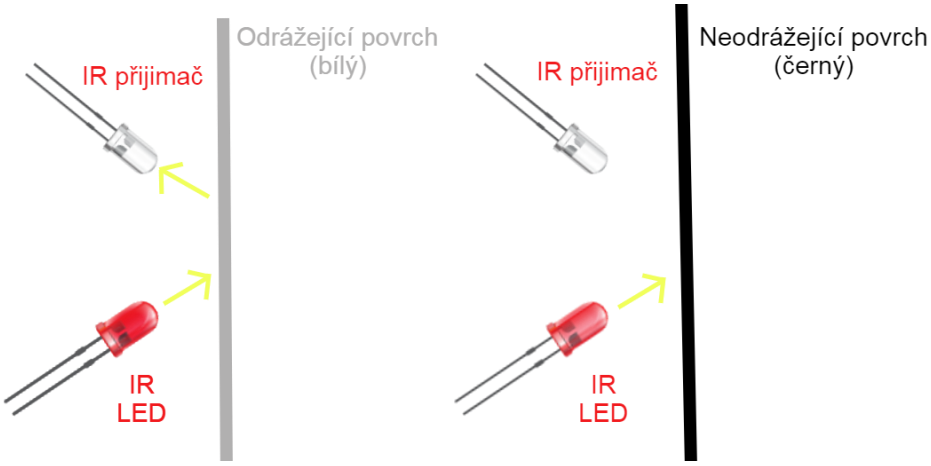
Senzor světla funguje podobně jako oči. Může pracovat ve dvou módech. V prvním módu určuje intenzitu okolního světla. V druhém módu pak určuje intenzitu odraženého světla.

Obr. 16 Senzor světla



Obr. 17 Popis senzoru světla

U druhého módu se na chvíli zastavíme. Při tomto módu se rozsvítí červená ledka, která je směřovaná tak, aby se její světlo odráželo přímo do samotného senzoru světla. Jak víme z fyziky, obecně se záření nejlépe odráží od světlých lesklých povrchů. Naopak se záření nejhůře odráží od tmavých matných povrchů. Tento senzor využijeme při sledování černé čáry na světlém pozadí. Černá čára téměř nebude odrážet světlo, a tudíž naměřená hodnota bude nízká. Naopak světlé pozadí bude světlo vyzařované ledkou odrážet značně a senzorem měřená hodnota bude vysoká. Tento problém je ilustrovaný na Obr. 18.



Obr. 18 Ilustrace odrazu světla

2.5.3.1. Programovací prostředí PyNXC a senzor světla:

Nejprve je potřeba senzor nainicializovat příkazem:

- SetSensorLight(Port)
    - **Port** označuje port na kostce NXT, na kterém je připojený světelný senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.

Dále je potřeba nastavit, v jakém módu bude senzor pracovat. Módem rozumíme, zda senzor pracuje jako senzor odraženého světla, nebo senzor okolního světla. Přepínání módů se provádí následovně:

- SetSensorType(Port, Makro)
  - **Port** označuje port na kostce NXT, na kterém je připojený světelný senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.
  - Makro je dvou typů:
    - **IN_TYPE_LIGHT_ACTIVE** – měření v módu odraženého světla
    - **IN_TYPE_LIGHT_INACTIVE** – měření v módu okolního světla

Odčítání hodnoty ze senzoru se provádí příkazem:

- Light = Sensor(Port)
  - **Port** označuje port na kostce NXT, na kterém je připojený světelný senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.
  - Příkaz senzor odečte hodnotu ze senzoru na příslušném portu a tu poté zapíše do proměnné Light. Tato proměnná nabývá hodnot od 0-100 podle intenzity světla.

2.5.3.2. Ukázka kódu pro měření odraženého světla v jazyce PyNXC.

```
2  def main():
3      SetSensorLight(IN_3) #inicializuje se senzor svetla na portu 3
4      SetSensorType(IN_3, IN_TYPE_LIGHT_ACTIVE) #nastavi se mod senzoru na mereni odrazenoho svetla
5      while(True): #nekonecna smycka
6          light = Sensor(IN_3)  #nacte se hodnota ze senzoru na portu 3
7          TextOut(10, LCD_LINE2, "Hodnota svetla:", true) #vypise se: Hodnota svetla, přičmž se resetuje se obrazovka
8          NumOut(70, LCD_LINE3, light, false) #vypise se ciselo odpovidajici stavu zmacknuti tlacitka
```

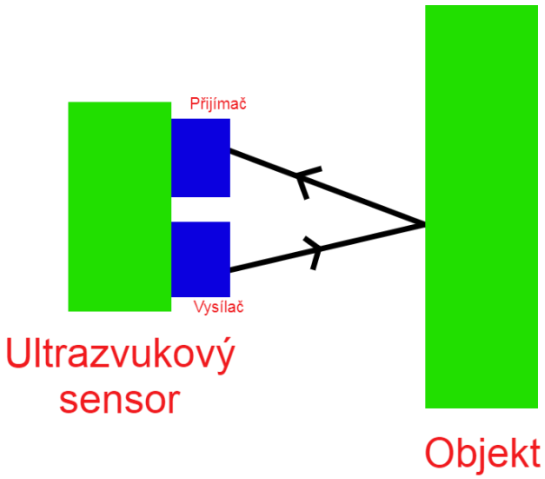
Na řádku 3 nastavíme senzor světla na portu 3. Následně nastavíme příkazem SetSenostType() mód měření, jako měření odraženého světla. Dále v nekonečné smyčce budeme odečítat hodnoty ze světelného senzoru připojeného na port 3 pomocí příkazu Sesnor(). Zároveň uložíme odečtenou hodnotu do proměnné light. Následně tuto změřenou hodnotu zapíšeme na obrazovku NXT kostky.

2.5.4. Ultrazvukový senzor

Ultrazvukový senzor pracuje podobně jako echolokace u netopýrů a slouží k měření vzdálenosti mezi robotem a překážkou, která může být podle výrobce vzdálena až 255 cm. Senzor vyšle krátký pulz ultrazvuku tzv. vysílačem, který se odrazí od překážky a senzor ho opět příjme tzv. přijímačem. Z času mezi vysláním pulzu a jeho příjmem se vypočítá vzdálenost překážky a to následovně

$$vzdálenost = \frac{dobaLetuPaprsku}{2 \cdot rychlostLetuPaprsku}. \qquad (2)$$

Ilustrace vyslání a příjmu pulzu senzorem je na Obr. 19.



Obr. 19 Ilustrace funkce senzoru

Obr. 20 Ultrazvukový senzor

Problém u tohoto senzoru je jeho robustnost a dále, že ultrazvuk se odráží nejlépe od lesklých předmětů, naopak matné předměty ho v řadě případů zcela pohlcují, čímž může docházet k nesprávnému měření vzdálenosti. Příkladem může být látkou potažený předmět, který se stává pro tento senzor téměř nezachytitelný. Jiný problém může nastat u zaoblených předmětů, které můžou ultrazvukový paprsek zcela odrazit mimo dosah přijímače. Další nedokonalost tohoto senzoru je, že se zvuk šíří při různých teplotách různou rychlostí. Tuto skutečnost senzor, respektive implementace čtení jeho naměřené hodnoty v řadě programovacích jazyků nezohledňuje. Přes všechny tyto nedokonalosti bude pro nás senzor nepostradatelný a my se ho naučíme správně používat.

Oproti jiným senzorům tento senzor lze připojit pouze na port 4 NXT kostky. Jedině tento port je totiž schopný ultrazvukovému senzoru dodávat až 9 voltů, které jsou potřebné pro jeho funkčnost. Ostatní porty pro senzory mohou poskytovat pouze napětí 4,3 voltů. [18]

2.5.4.1. Programovací prostředí PyNXC a ultrazvukový senzor

Jako obvykle je potřeba senzor nejprve nainicializovat a to příkazem:

- SetSensorLowspeed(Port)
    - **Port** označuje port na kostce NXT, na které je připojený ultrazvukový senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.

Nebo se zde nachází druhý způsob inicializace senzoru.

- SetSensorUltrasonic(Port)
  - **Port** označuje port na kostce NXT, na kterém je připojený ultrazvukový senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.

Odčítání hodnoty ze senzoru se uskutečňuje následujícím způsobem:

- distance = SensorUS(Port)
  - **Port** označuje port na kostce NXT, na kterém je připojený ultrazvukový senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.
  - Příkaz SensorUS() odečte hodnotu ze senzoru na příslušném portu a tu poté zapíše do proměnné distance. Odečítání hodnot se provádí v centimetrech.

Proč na rozdíl od ostatních senzorů používáme příkaz SensorUS() a ne jen Sensor()? Ostatní senzory jsou připojené přímo na analogově číslicový převodník (A/D) v kostce NXT. Zjednodušeně řečeno hodnota měřené veličiny se převádí na elektrický impuls, až v NXT kostce. Naproti tomu ultrazvukový senzor je připojený na sběrnici I2C. K převodu měřené veličiny na elektrický impuls tedy dochází již v senzoru. Mimo jiné přináší tato sběrnici řadu výhod, ale nám postačí znalost, že se jedná o dvě odlišné technologie, a proto se musí i mírně jinak programovat. Rovněž si dejte pozor na čas načítání dat z ultrazvukového senzoru, který činí 15 ms. [19]

2.5.4.2. Ukázka snímání hodnoty vzdálenosti pomocí ultrazvukového senzoru v jazyce PyNXC

```
1  def main():
2      SetSensorLowspeed(IN_4) #inicializuje se ultrazvukovy senzor na portu 4
3      while(True): #nekonecna smycka
4          distance = SensorUS(IN_4)  #nacte se hodnota ze senzoru na portu 4
5          TextOut(10, LCD_LINE2, "Vzdalenost:", true) #vypise se: Vzdalenost, přičmž se resetuje se obrazovka
6          NumOut(70, LCD_LINE3, distance, false) #vypise se ciselo odpovidajici sound
7          Wait(10) #z duvodu obnoveni displeje
```

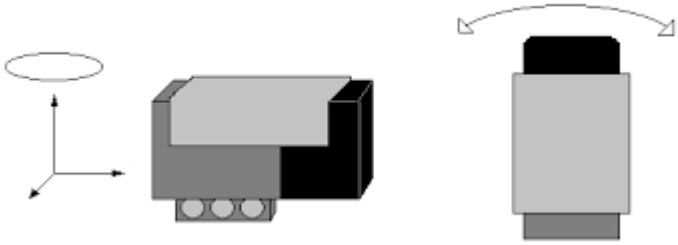
Nejprve si inicializujeme ultrazvukový senzor na portu 4 příkazem SetSensorLowspeed(). Jinou možnost, kam připojit senzor nemáme. Dále se v nekonečné smyčce načítá hodnota ze senzoru příkazem SensorUS(), která se následně vypisuje na obrazovku NXT kostky. Příkaz Wait() je zde z důvodu, abychom lépe viděli naměřenou vzdálenost a nepřepisovala se nám na obrazovce tak rychle.

2.5.5. Gyroskopický senzor

Tento senzor je pro Lego Mindstorsms NXT senzorem třetích stran, nevyrábí ho firma Lego, ale firma HiTechnic. Je nicméně velice populární a pro stavebnice Lego Mindstorsms EV3 si ho již Lego v mírně odlišné verzi vyrábí samo.

Gyroskopický senzor měří úhlovou rychlost v poněkud netradičních jednotkách stupňů za sekundu. Toto měření provádí pouze v jedné ose, která je naznačená na Obr. 21. Asi nejbližší připodobnění tohoto senzoru je k lidskému uchu, konkrétněji k jeho hlemýždi. Ten taky pozná díky pohybu tekutiny v něm úhlové zrychlení. Senzor funguje podobně s tím rozdílem, že je zde tekutina nahrazena gyroskopem.

Obr. 21 Gyroskopický senzor



Obr. 22 Znázorněný měřené osy [20]

Kdybychom chtěli určit pomocí tohoto senzoru úhel natočení oproti počátečnímu stavu, není to veliký problém. Vyžaduje to akorát trochu matematiky. Ze základní znalosti fyziky víme, že integrace úhlové rychlosti podle času je úhel, což není nic složitého. Ve skutečnosti stačí ve smyčce provést následující výpočet.

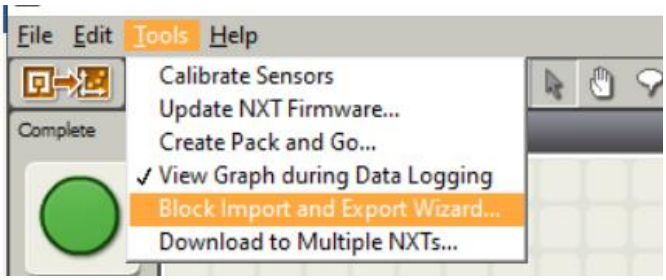
$$uhel = uhel + dobaTrvaniSmycky \cdot uhlovaRychlost \qquad (3)$$

Problém, který přichází s tímto senzorem, je potřeba jeho kalibrace před každým jeho použitím. Senzor měří úhlovou rychlost a my si musíme uvědomit, že i když je senzor v absolutním klidu vzhledem k Zemi, Země se s námi pohybuje. I když se vliv této rychlosti upravuje již v továrně, nedochází k jeho úplnému odstranění. Proto můžeme na senzoru naměřit okolo 2 °/s, i když se senzor vzhledem k povrhu Země nepohybuje. Offsetem v tomto případě nazýváme hodnotu, která znázorňuje rozdíl mezi naměřenou hodnotu na senzoru a skutečnou hodnotu úhlové rychlosti. Tento offset se dá odstranit tím, že provedeme několik měření, když se senzorem nehýbeme. Tyto měření následně zprůměrujeme. Zprůměrovanou hodnotu neboli offset budeme odčítat od hodnoty naměřené senzorem, abychom dostali co možná nejpřesnější údaje o úhlové rychlosti.

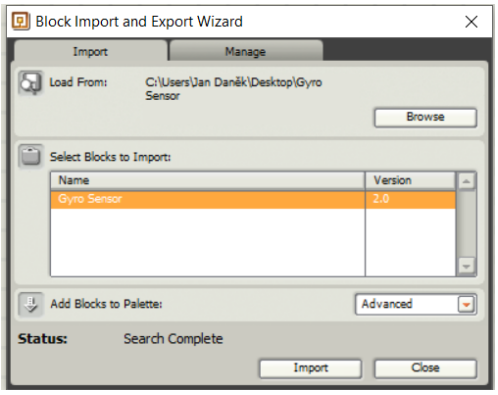
2.5.5.1. Programovací prostředí a gyroskopický senzor

2.5.5.1.1. NXT-G

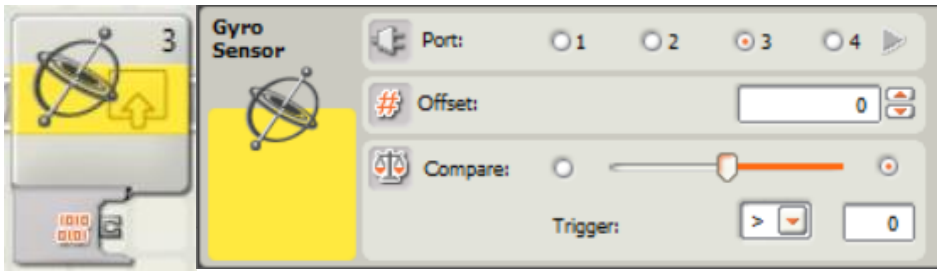
Blok pro tento senzor je potřeba doinstalovat. Stáhněte si ho na stránkách výrobce tohoto senzoru ({1}). Stažený soubor .zip si rozbalte.



Dále je potřeba v prostředí NXT-G vybrat v menu nahoře v záložce **Tool** sekci **Block Import and Export Wizard**. Zobrazí se vám následující dialogové okno.

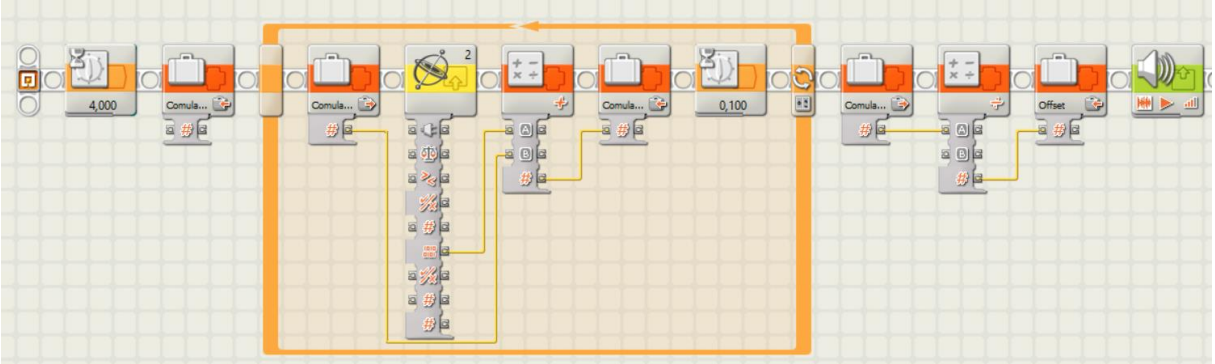


V první sekci **Load From** vyberete rozbalenou složku s gyro senzorem. Následně pak v druhé sekci **Select Blocks to Import** vyberte **Gyro Sensor**. Následně vybere v Add Block to Palette, kde chcete, aby se vám nový blok pro senzor zobrazoval. Když zde vyberete sensors, naleznete ho mezi senzory.



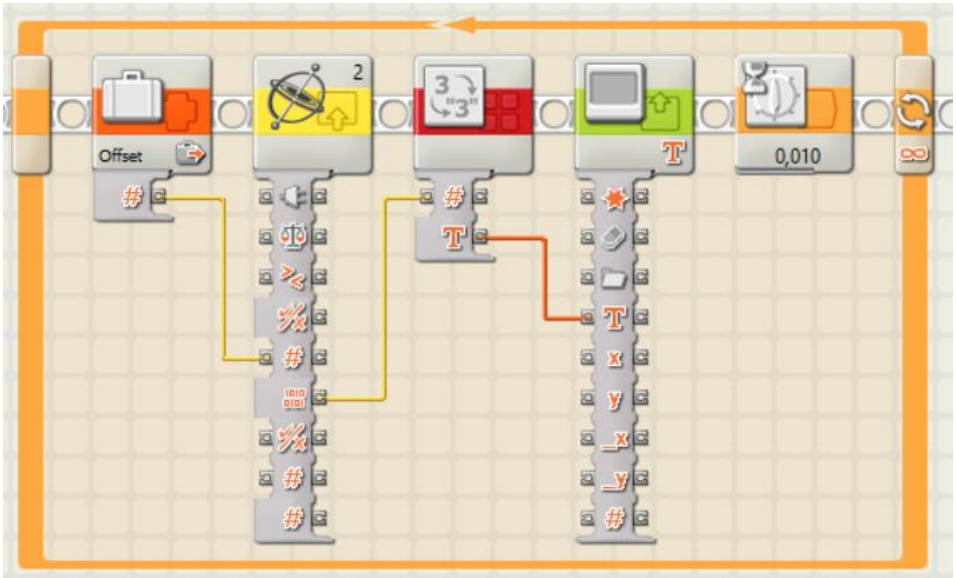
Gyroskopický senzor se programuje pomocí bloku **Gyro Sensor**. Ve vrchní části *Ports* zaškrtneme, na kterém portu je senzor připojený. V sekci *Offset* se pak dá nastavit offset, tedy o kolik se liší hodnota úhlové rychlosti od změřené rychlosti. V sekci *Compare* se pak dá nastavit hodnota, při které bude senzor vysílat signál pravda z logického výstupu.

Nyní si ukážme jak jednoduše spočítat offset tohoto senzoru v prostředí NXT-G.



Po zapnutí programu počkáme 4 sekundy, během kterých dojde k oddálení ruky od NXT kostky a senzor se během tohoto času dostane do klidu. V prvním bloku **Variable** se nastaví proměnná Comulative na 0. Dále se ve smyčce, která se uskuteční celkem 30x, provedou následující úkony. Nejdříve se načte hodnota z proměnné Comulative. Dále se načte honota raw value z gyroskopického senzoru, který je v tomto případě připojený na port 2. Raw value je hodnota senzoru, která není nijak přeškálovaná. V bloku **Math** se sečte hodnota Comulative a hodnota naměřená na senzoru. Takto spočtená hodnota se pak zapíše opět do proměné variable. Následně se počká 0,1 sekundy a celý proces se 29krát zopakuje. Po této smyčce následuje výpočet offsetu. Zde se jednoduše zprůměrují hodnoty naměřené v předchozí smyčce tím, že se hodnota Comulative vydělí 30. Takto spočtená hodnota se zapíše do proměnné Offset. Blok **Sound** nás upozorní na ukončení procesu výpočtu offsetu.

Odečítání hodnoty z tohoto senzoru včetně její regulace z hledistka offsetu a její výpis na obrazovku NXT kosky se provádí následovně.



Proměnná offset se připojí na senzor do jeho vstupu offset. Dávejte si pozor, offset je značený #, stejně tak jako MUX port nebo Datalog output. Dále ze senzoru načteme data ve °/s, která jsou již upravená o odečtení offsetu. Tyto data převedeme na text v bloku **Number to Text** a dále je pak zobrazíme na displeji NXT kostky.

2.5.5.1.2. NXC a PyNXC

Jelikož se jednotlivé příkazy pro práci mezi PyNXC a NXC mezi sebou neliší, rozhodl jsem se, že tyto dvě kapitoly sloučím do jediné.

Tak jako každý senzor musíme i gyroskopický senzor nejprve inicializovat. Tuto inicializaci uskutečňujeme následujícím příkazem.

- SetSensorHTGyro(Port)
  - **Port** označuje port na kostce NXT, na kterém je připojený gyroskopický senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.

Odčítání hodnoty ze senzoru se provádí následujícím příkazem.

- speed = SensorHTGyro(Port)
  - **Port** označuje port na kostce NXT, na kterém je připojený gyroskopický senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.
  - Odečte se hodnota ve °/s ze gyroskopického senzoru, která se následně zapíše do proměnné speed.

Druhý způsob odečtení dat ze senzoru, který ale autor sborníku nemůže doporučit je následující.

- speed = SensorHTGyro(Port, Offset)
  - **Port** označuje port na kostce NXT, na kterém je připojený gyroskopický senzor. Zapisuje se ve tvaru IN_X, kde X značí číslo portu.
  - Odečte se hodnota ve °/s z gyroskopického senzoru upravená o offset, která se následně zapíše do proměnné speed.
  - **Offset** – hodnota označuje offset našeho senzoru
  - Tento příkaz nefunguje tak jak by se dalo očekávat a přesnějších výsledků dosáhneme v okamžiku, kdy si offset od naměřené hodnoty sami odečteme.

2.5.5.1.3. Ukázkový kód na určení hodnoty offset (řádek 13-18) a určení natočení senzoru (řádek 22-32) v jazyce NXC

```
1  task main(){
2      float speed; //inicalizace promenne speed pro zapisovani namerenych dat
3      float angle; //inicalizace promenne angle pro zapisovani namereneho uhlu
4      float comulative; //inicalizace promenne comulative pro vypocet offsetu
5      float offset; //inicalizace promenne offset pro ulzeni vypocitaneho offsetu
6
7      SetSensorHTGyro(IN_2); //inicalizace gyroskopickeho senzoru na portu 2
8      angle = 0; //nastaveni uhlu na nulu
9      comulative = 0; //nastaneni comulative na nulo
10
11     Wait(4000); //cekani z duvodu ustaleni senzoru
12
13     for(int i = 0; i <30; i++ ){ // smycka se 30 opakovanimi
14             comulative = comulative + SensorHTGyro(IN_2); //nacteni dat a jejijich pricteni ke komulative
15             Wait(100); //cekani
16     }
17
18     offset = comulative/30; //vypocet offsetu
19
20     PlaySound(SOUND_LOW_BEEP); //oznameni o ukoncneni mereni offsetu
21
22     while(1){ //nekonecna sycka
23              speed = SensorHTGyro(IN_2); //odecteni hodnoty ze senzoru
24
25              angle = angle + 0.013*(speed-offset); //vypocet uhlu
26
27              TextOut(10, LCD_LINE2, "Speed:", true); //vypis na displej
28              NumOut(50, LCD_LINE2, speed-offset); //vypis na displej
29              TextOut(10, LCD_LINE4, "Angle:", false); //vypis na displej
30              NumOut(50, LCD_LINE4, angle); //vypis na displej
31              Wait(10); //cekani
32     } //konec nekonecne smycky
33 } //konec programu
```

Program začíná inicializací proměnných speed pro zapisování hodnoty ze senzoru, angle pro zapisování úhlu, o který se senzor natočil. Dále zde je proměná comulative jako pomocná pro výpočet offsetu a samotnou proměnou offset, do které se zapíší finální hodnotu offsetu. Všechny tyto proměnné jsou typu float, takže se do nich budou zapisovat desetinná čísla. Následně inicializujeme gyroskopický senzor na portu 2. A nastavíme proměnné angle a comulative na 0. Poté 4 sekundy počkáme z důvodu, aby uživatel odstranil po stisknutí tlačítka ruku z NXT kostky a senzor se tak nacházel v co možná největším klidu.

Dále následuje 30 měření senzoru ve smyčce for(). Celkem tedy 30x změříme hodnotu úhlové rychlosti na senzoru. A všechny tyto měření sečteme do proměnné comulative. Kde z ní následně na řádku 18 uděláme průměr naměřených hodnot, což je vlastně náš hledaný offset. Po tomto výpočtu zazní zvuk, který nám označuje konec výpočtu offsetu.

Dále se pak pustíme do měření natočení senzoru vůči poloze senzoru na začátku. Tento výpočet provedeme v nekonečné smyčce podle rovnice (3). Napřed odečteme hodnotu úhlové rychlosti ze senzoru. Následuje výpočet z rovnice (3). Výraz speed-offset je nejpřesnější hodnota úhlové rychlosti, kterou jsme schopni změřit. Je to změřená hodnota z gyroskopického senzoru, od které je odečtený offset. Hodnota 0.013 je experimentálně změřená doba trvání while cyklu. Následuje výpis na obrazovku NXT kostky.

2.5.5.1.4. Ukázkový kód na určení hodnoty offset (řádek 9-12) a určení natočení senzoru (řádek 18-27) v jazyce PyNXC

```
1   def main():
2
3       SetSensorHTGyro(IN_2) # inicalizace gyroskopickeho senzoru na portu 2
4       angle = 0 #nastaveni uhlu na nulu
5       comulative = 0 #nastaneni comulative na nulo
6
7       Wait(4000) #cekani z duvodu ustaleni senzoru
8
9       i=0
10      for i in range(0,30): #smycka se 30 opakovanimi
11          comulative = comulative + SensorHTGyro(IN_2) #nacteni dat a jejijich pricteni ke komulative
12          Wait(100) #cekani
13
14      offset = comulative / 30 #vypocet offsetu
15
16      PlaySound(SOUND_LOW_BEEP) #oznameni o ukoncneni mereni offsetu
17
18      while(1): #nekonecna sycka
19          speed = SensorHTGyro(IN_2) #odecteni hodnoty ze senzoru
20
21          angle = angle + 0.013 * (speed-offset) #vypocet uhlu
22
23          TextOut(10, LCD_LINE2, "Speed:", true) #vypis na displej
24          NumOut(50, LCD_LINE2, speed-offset) #vypis na displej
25          TextOut(10, LCD_LINE4, "Angle:", false) #vypis na displej
26          NumOut(50, LCD_LINE4, angle) #vypis na displej
27          Wait(10) #cekani
```

Program začíná inicializací gyroskopického senzoru na řádku 3. Dále se nastavují na nulu proměnné angle pro zapisování úhlu natočení gyroskopického senzoru a comulative, jako pomocná proměnná pro výpočet offsetu. Dále následuje příkaz Wait(), kdy čekáme, až uživatel vzdálí po spuštění programu ruku od NXT kostky a ta se bude nacházet v co největším klidu.

Ve for cyklu se provede 30x měření gyroskopickým senzorem. Naměřené hodnoty se postupně sečtou v proměnné comulative na řádku 11. Po for cyklu následuje výpočet průměru naměřených hodnot.

Jelikož byl senzor po dobu měření v klidu, jedná se o offset senzoru. Dále následuje přehrání zvuku, aby robot oznámil, že skončila kalibrační část.

Od řádku 18 je pak kód pro výpočet úhlu natočení senzoru oproti původní poloze na začátku. Na řádku 21 se počítá rovnice (3). Výraz speed-offset je nejpřesnější hodnota úhlové rychlosti, kterou jsme schopni změřit. Je to vlastně změřená hodnota z gyroskopického senzoru, od které je odečtený offset. Hodnota 0,013 je experimentálně změřená doba trvání while cyklu. Následuje výpis na obrazovku NXT kostky aktuálního úhlu natočení a aktuální rychlosti.

2.5.5.2. Užitečná rada o nulování úhlu

I když senzor bude v klidu a od změřených hodnot budeme odečítat offset, pořád neměří nenulovou hodnotu úhlové rychlosti. Proto dochází k postupnému načítání hodnot úhlu natočení. Proto doporučuji tuto hodnotu nulovat tak často, jak je to možné.

2.5.5.3. Užitečná rada o stanovení doby běhu smyčky

Jelikož je nepříjemné zjišťovat dobu trvání smyčky a u některých složitějších programů se může doba trvaní smyčky lišit mezi jednotlivými jejími iteracemi. Nabízí se možnost určovat dobu trvání smyčky pomocí vnitřního času NXT kostky. Vnitřní čas NXT kostky se určí pomocí příkazu CurrentTick(). Udává se v jednotce Tick. Jednotka Tick je oproti klasickému pojetí v programátorské a hráčské komunitě 1 milisekunda. V těchto komunitách je zpravidla 20 Ticků 1 sekunda. Ukázka kódu v jazyce NXC pro počítání úhlu pomocí vnitřního času:

```
19 task main(){
20     float speed; //inicializace promenne
21     float tickNow; //inicializace promenne
22     float tickLast; //inicializace promenne
23     float angle; //inicializace promenne
24     float offset; //inicializace promenne
25     SetSensorHTGyro(IN_2); //inicializace gyroskpickeho senzoru
26     offset = vypocti_offset(); //vypocet offsetu gyroskopickeho senzoru
27     tickLast = CurrentTick();  //zjisteni aktualniho casu NXT kostky
28     angle = 0; //nastaveni promenne angle na nulu
29     while (true){ //nekonecna smycka
30         tickNow = CurrentTick(); //zjisteni aktualniho casu NXT kostky
31         speed = SensorHTGyro(IN_2); //odecteni hodnoty z gyroskopickeho senzoru
32         angle = (angle+abs(tickNow -tickLast)*(speed-offset)/1000); //vypocet uhlu
33         TextOut(10, LCD_LINE4, "Angle:", false); //vypis na displej
34         NumOut(50, LCD_LINE4, angle); //vypis na displej
35         Wait(10); //cekani
36         tickLast = tickNow;
37     } //konec nekonecne smycky
38 } //konec programu
```

Ukázka kódu v jazyce PyNXC pro počítání úhlu pomocí vnitřního času:

```
11  def main():
12      SetSensorHTGyro(IN_2) #inicalizace gyroskopickeho senzoru na portu 2
13      offset = vypocti_offset() #vypocet offsetu
14      tickLast = CurrentTick() #zjisteni aktualniho casu NXT kostky
15      angle = 0 #nastaveni promenne angele na nulu
16      while (True): #nekonecna smycka
17          tickNow = CurrentTick() #zjisteni aktualniho casu NXT kostky
18          speed = SensorHTGyro(IN_2) #odecteninhodntoty z gyroskopickeho senzoru
19          angle = angle + abs(tickNow-tickLast)*(speed-offset)/1000 #vypocet uhlu
20          TextOut(10, LCD_LINE4, "Angle:", false) #vypis na displej
21          NumOut(50, LCD_LINE4, angle) #vypis na displej
22          Wait(10) #cekani
23          tickLast = tickNow #prepis kvuli novemu vypoctu uhlu
```

Výpočet offsetu je v obou ukázkových kódech provedený pomocí funkce vypocti_offset().

2.6. Závěr této kapitoly

V této kapitole jsme se seznámili se stavebnicí Lego Mindstorms NXT. Na začátku jsme si popsali její části. Seznámili se s programovacími prostředky, které můžeme k programování této stavebnice použít. Z nabídky programovacích jazyků jsem si vybral tři. Všechny tyto jazyky se vyznačují tím, že jsou jednoduché na pochopení, jsou zdarma, a navíc běží na originálním firmware od Lego. NXC je programovací jazyk podobný jazyku C. NXT-G je grafický jazyk přímo od společnosti lego. Jako poslední jsem si vybral PyNXC, i když se příliš neliší od jazyka NXC, ale má alespoň pro autora přehlednější syntaxi a obecně se s ním lépe pracuje. Druhý důvod, proč jsem se ho rozhodl zařadit do této práce, je z důvodu, že se studenti učí tento jazyk v předmětu Algoritmy a programování, který běží paralelně s výukou předmětu Roboti. Programovací jazyk C se vyučuje až ve druhém semestru. Dále je v této kapitole vysvětleno, jak se přenáší a kompilují soubory z prostředí PyNXC do NXT kostky. Popsal jsem aktuátory a senzory NXT stavebnice, včetně způsobu jejich programování v jazyce PyNXC. Tyto výklady jsem doplnil o ukázkové kódy.

3. Sborník úkolů

V této kapitole bych rád představil některé úkoly, se kterými se studenti setkají v průběhu předmětu ROBOTI na ČVUT FEL. Tyto úlohy vychází již ze zadaných úkolů k tomuto předmětu. Jejich zadání můžeme nalézt na Moodle stránce předmětu ROBOTI dostupné {2}. V tomto sborníku došlo k úpravě a zjednodušení těchto zadaní, aby formy zadání více odpovídali samostatným úkolům než popisu pravidel soutěže, které se jednotliví roboti účastní, a aby bylo možné bodové ohodnocení studentů. Cílem této kapitoly není nalézt pro jednotlivé úkoly složitá řešení, ale vysvětlit principy chování robotů, a ty dále implementovat do různých programovacích prostředí srozumitelnou cestou tak, aby tito roboti mohli splnit zadání. Toto zjednodušení jsem zavedl, protože předmět ROBOTI se vyučuje v prvním semestru bakalářského studia a měl by sloužit jako motivační předmět. Některé úlohy v tomto sborníku rovněž obsahují návod na stavbu robotů. Modely jednotlivých robotů jsem vytvořil v prostředí Lego Digital Designer. Následné vytvoření plánů na stavbu probíhalo v prostředí Studio 2.0.

3.1. Konstrukce robota

V této kapitole si uvedeme obecné požadavky na roboty, které musí roboti splňovat, aby mohl být daný úkol považován za splněný. Tyto požadavky jsou shodné pro všechny další úkoly a měli by zabránit poškození robotů či prostředí, ve kterých se nachází. Rovněž by měli studenty navést k nalezení konstrukčně propracovaného robota. Pokud by se s roboty pořádaly soutěže, jsou zde uvedeny i některé požadavky na zaručení regulérnosti těchto soutěží. Následující požadavky vychází z požadavků, které jsou odzkoušené na Robosoutěži. Výčet těchto pravidel nalezneme na stránce {3}.

3.1.1. Obecné požadavky na roboty

Definujme si obecné podmínky, které by měl robot splňovat pro řešení následujících úloh. Aby nedošlo k poničení okolního prostředí nebo samotného robota. Zároveň těmito podmínkami chci definovat požadavky, aby byla zaručena regulárnost v případě konání soutěže.

Robot by měl být sestaven pouze z takových dílů, které se nachází právě v jedné sadě 9797, jedné sadě 9648 nebo jedné sadě 9695 doplněny pouze o jeden senzor třetích stran, čímž je modul gyroskopu. Uvedené sady lze mezi sebou libovolně kombinovat, musí však být dodržena podmínka pouze jedné stavebnice od každého typu. Rovněž mohou díly stavebnice držet pohromadě pouze standartními spojovacími prvky Lego. Nemůže být tedy ke spojování použito například lepidlo, šrouby, vázací pásky, provazy nebo izolepa, či jiné spojovací materiály nebo díly.

Půdorys robota se před začátkem daného úkolu musí vejít do čtverce o hraně 28 cm. Po zahájení řešení úkolu, je dovoleno půdorys libovolně měnit.

Samotný robot musí vážit maximálně 1,5 kg, aby nedocházelo k nadměrnému opotřebování servomotorů v důsledku pohybu robota.

Ve všech úkolech je zakázáno použití kluzného podvozku. Při užití tohoto typu podvozku by mohlo dojít k poničení ploch, po kterých se robot pohybuje. Za kluzný podvozek považujeme, jakoukoliv část robota, která je ve styku s podložkou a na níž leží část váhy robota a zároveň se po ploše neodvaluje, ale klouže.

V úkolech, kde se soutěží v projetí soutěžní dráhy, není dovoleno robota vybavovat jakýmkoliv zařízením, které by dříve protnulo cílovou čáru za cenu změny půdorysu robota oproti situaci na startovní čáře.

3.1.2. Programování robota

K programování NXT kostky lze využít libovolný programovací jazyk. Pokud si vyberete programovací jazyk, kvůli kterému musíte přehrát originální firmware NXT, nahrajte do NXT kostky před odevzdání zpět originální firmware.

3.1.3. Napájení robota

Při programování a ladění chodu robota, lze použít libovolnou možnost napájení. Po dokončení programovaní a plnění samotného úkolu, bude robot napájen výhradně z 6 baterií AA, nebo z akumulátorové baterie.

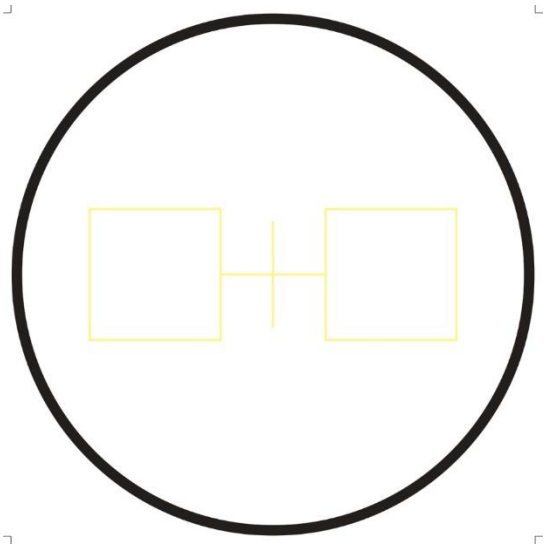
3.2. Úkol 1: Sumo

V této podkapitole se zaměříme na definování zadání, rozbor problému a konstrukci robota pro sumo zápasy. Zároveň si ukážeme napsání základního programu v rozličných programovacích prostředích a podíváme se na první testování robota.

3.2.1. Zadání

Naším úkolem je vytvořit sumo robota, který zvládne vytlačit z kruhu soupeřova sumo robota, aniž by sám vyjel větší částí svého půdorysu z této hrací plochy. Za vytlačení protivníka z hrací plochy se považuje situace, když se žádná část jeho podvozku nedotýká vymezené hrací plochy.

Roboti budou zápasit na hrací ploše ve tvaru kruhu o průměru 1 m. Hrací plocha je ohraničena černou čárou o šířce 2 centimetry a uvnitř jsou žlutou barvou vyznačeny startovní pozice jednotlivých robotů. Tato hrací plocha je vyobrazena na Obr. 23.



Obr. 23 Hrací plocha pro sumo zápasy

Obr. 23, rovněž jako Obr. 41, Obr. 64, Obr. 65 a Obr. 66, mi byly poskytnuty vedoucím práce.

Před startem zápasu budou roboti umístěni do vyznačených žlutých čtverců a to tak, aby směřovali zády k sobě. Je zakázáno, aby robot ihned po odstartování zápasu, vyjel směrem k soupeři. Za záda robota se považuje opačná strana, než na které je radlice a ultrazvukový senzor. V případě pochybností může rozhodčí zápas ukončit a požadovat změnu orientace robota ve žlutém čtverci.

Robot po odstartování programu musí 10 sekund vyčkat na svém místě. Během této doby se všechny osoby vzdálí do bezpečné vzdálenosti od hrací plochy.

Po uplynutí doby, potřebné k bezpečnému odchodu z hrací plochy, začíná samotný sumo zápas. A trvá tak dlouho, dokud jeden z robotů nevyjede mimo hrací plochu.

Zápas končí v okamžiku, kdy je jeden z robotů vytlačený za černou čáru. Zároveň nesmí vítězný robot vyjet z tohoto kruhu. Když se po skončení zápasu nachází oba roboti mimo hrací plochu, dochází k remíze. Zápas rovněž končí v okamžiku, kdy jeden z robotů není schopný v zápasu pokračovat.

Rozhodčí si vyhrazuje právo zápas kdykoliv ukončit, nebo nepřipustit začátek souboje, pokud nabude dojmu, že jsou porušována pravidla, nebo dochází k materiálním škodám na zařízeních LEGO Mindstorms NXT.

3.2.2. Rozbor problému

Během stavby a prvního testování sumo robota, nebudeme mít k dispozici soupeře a rovněž si nemůžeme být zcela jistí, jak se náš sumo robot bude chovat. Z těchto dvou důvodů nahradíme soupeřova robota za kostku, která je vyobrazená na Obr. 24.

Obr. 24 Kostka představující soupeřova robota

Původ obrázků Obr. 24, Obr. 30 a Obr. 58 je na Moodle stránkách předmětu Roboti. Tato stránka je dostupné na odkaze {2}.

Robot bude muset vytlačit kostku z hrací plochy, ale zároveň na této ploše bude muset sám zůstat. Abychom mohli problém začít řešit, musíme si nejprve definovat, co všechno robot musí umět. Jeho dovednosti určíme ze zadání. Robot musí být schopen vytlačit robota soupeře z hrací plochy a zároveň chceme zabránit tomu, aby se robot stal snadnou „kořistí“ pro soupeřova robota. To znamená, že se robot bude muset umět pohybovat po hrací ploše. K tomuto účelu bude robot muset být vybavený dvěma servomotory, které mu zajistí volnost pohybu.

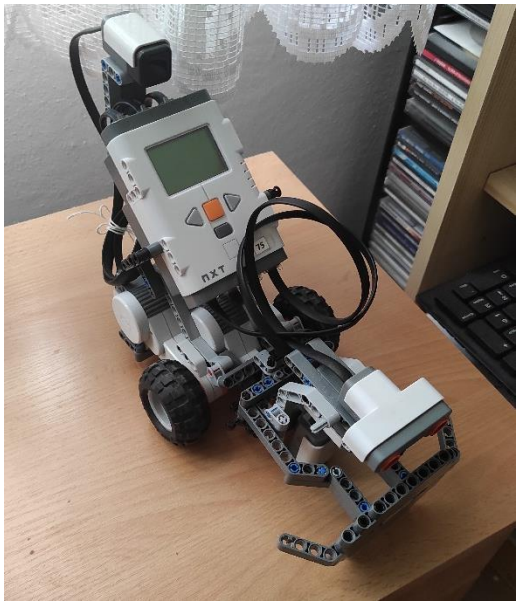
Robot se nyní umí pohybovat, ale otázka je, kam by se měl pohybovat. Náhodně jezdit po hrací ploše, a doufat, že narazíme na soupeře asi není příliš strategické. Je tedy třeba soupeře nějakým způsobem lokalizovat a k tomu použijeme ultrazvukový senzor.

Zároveň ale musíme dbát na to, abychom neopustili hrací plochu vymezenou kruhem z černé čáry. Zde využijeme detektoru světla v módu měření odraženého světla, kde využijeme rozdílných odrazných vlastností bílé barvy hrací plochy a černé barvy jejího ohraničení. Žluté čáry startovních míst díky značnému rozdílu černá a bílé barvy nebudou překážkou. Černý okraj bude totiž mít na detektoru hodnotu zhruba 30, oproti tomu bílá plocha hrací desky bude mít přibližně hodnotu 70 a žluté se bude přibližovat hodnotám bílé barvy.

3.2.3. Konstrukce

Na stavbu robota jsem použil sadu LEGO 9797 a sadu 9695. Při konstrukci robota jsem vyšel z modelu Tribota, který je navržený firmou Lego a návod na jeho sestavení je k nalezení {4}. Oproti původnímu návodu jsem zpevnil zadní kolo. Při pohybu s původním Tribotem často docházelo k odpadnutí přední části robota, na které se nacházeli pomyslné ruce robota. Tato část byla na hlavní konstrukci jen lehce nasazená a nic nebránilo jejímu odpadnutí. Z tohoto důvodu jsem spoj mezi hlavní části a rukama robota vyměnil za robustnější. Pořád jsem při stavbě robota pamatoval na pravidla soutěže. Robot je oproti původní konstrukci doplněn o radlici, která je umístěná na původních „rukách“ robota. Foto tohoto robota je na Obr. 25 a Obr. 26.

Obr. 25 Foto sumo robota I.



Obr. 26 Foto sumo robota II.

Při pohledu na robota zepředu, kdy se díváme přímo do ultrazvukového senzoru, je pravý motor připojený na port C a levý motor na port A. Ultrazvukový senzor je připojený na port 4 a světelný senzor je připojený na port 3. Dále je robot vybavený dotykovým senzorem na portu 1. My dotykový senzor pro tento úkol nepoužijeme. V pozdější fázi byl k robotovi připojený gyroskopický senzor na port 2. Takto připojený senzor je vidět na Obr. 26.

3.2.4. Příprava konstant

Změříme si přesné hodnoty odraženého světla od jednotlivých barevných podkladů, s nimiž se setkáme při řešení úlohy. Měření provedeme pomocí senzoru světla a kostky NXT. Stručně zde popíšu, jak se jednotlivé hodnoty dají přesně změřit. V hlavním menu najedeme na ikonu **View**. Potvrzením se dostaneme na výběr měřených veličin. Zde si vybereme **Reflected Light**. Po potvrzení nás robot vyzve, abychom vybrali **port**. My vybereme takový port, na kterém máme připojený senzor světla.

Nyní provedeme samotné měření. Osobně doporučuji provést měření na několika místech té stejné barvy a naměřené hodnoty zprůměrovat. Toto průměrování provádíme z toho důvody, aby se například nestala následující situace. Budeme měřit na místě, kde je bílá barva podkladu lehce

znečištěná, a my naměříme jinou hodnotu. Tato hodnota se může od skutečné hodnoty lišit až v řádu desítek. Tabulka naměřených hodnot je k vidění v Tabulka 2.

Podklad	1. měření	2. měření	3. měření	Průměr
Černý	30	35	33	32,666
Žlutý	65	67	69	67
Bílí	72	75	70	72,333

Tabulka 2: Naměřené hodnoty jednotlivých barev

3.2.5. Programování

Popíšeme si zde nejjednodušší program, který je schopný splnit zadání. Případnou úpravu kódu s volbu strategie včetně jejího naprogramování, již nechám na samotných studentech.

3.2.5.1. Pseudo-kód/diagram

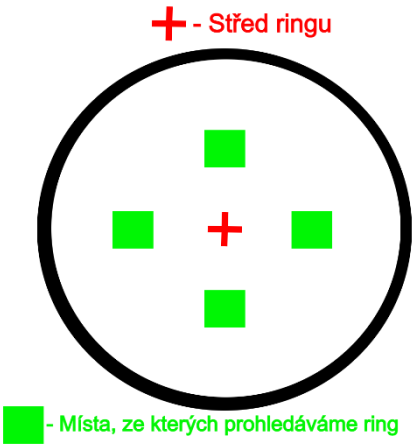
Následující kód udává myšlenky, jak by se měl takový sumo bojovník sestavený z Lego chovat. V následujících částech si ukážeme, jak ho naprogramovat v různých programovacích jazycích.

```
Zapnutí programu
Počkej 10 sekund
Otoč se do výchozí pozice

Prováděj pořád do kola:
        Odečti honotu ze senzoru světla a ulož ji do proměné světlo
        Odečti hodnotu z ultrazvukového senzoru a ulož ji do proměné vzdálnost
        Když vzdálenost < 50:
                Jeď do předu
                Dokavaď(vzdálenost < 50 a světlo < 50):
                        Odečti honotu ze senzoru světla a ulož ji do proměné světlo
                        Odečti hodnotu z ultrazvukového senzoru a ulož ji do proměné vzdálnost
        Jinak se:
                Otáčej se kolem vlastní osy

        Když světlo < 50
                Popojeď do zadu
                Cas_otaceni = 0

        Když Cas_otaceni > TODO:
                Otoc se o predem definovany uhel
                Popojeď o předem definouvanou vzálenost dopředu
                Cas_otoceni = 0

        Cas_otaceni = Cas_otaceni + 1
```

Myšlenka tohoto kódu je následující. Zapneme program a počkáme 10 sekund, aby se všechny osoby stihly bezpečně vzdálit od robota. Přejedeme s robotem ze startovního pole do výchozí polohy, ze které následně začneme provádět vyhledávání protivníka. V nekonečné smyčce budeme vykonávat následující. Načteme hodnotu ze senzorů. Když bude nějaký objekt blíže než 50 cm vydáme se jeho směrem a pokusíme se ho vytlačit ze hřiště. V opačném případě se robot dál pootáčí za stálého hledání cíle. Pokud dojde k pohybu robota vpřed, je třeba zároveň kontrolovat, zda se neblíží k hranici hrací plochy. Když zachytíme na senzoru světla hodnotu menší jak 50, je velice pravděpodobné, že se robot dostal na hranici hřiště, a proto musí mírně couvnout. Část pseudokódu za podmínkou Když Cas_otaceni > TODO je zde z důvodu, abychom mohli změnit stanoviště, ze kterého hledáme soupeře a prohledat tak větší plochu hřiště. TODO zde znázorňuje, že zadaná podmínka se bude lišit v různých programovacích jazycích. Aktivuje se pouze v případě, když po dobu dvou otočení robota kolem vlastní osy nespatříme soupeře na ultrazvukovém senzoru. To, jakým způsobem budeme pokrývat hrací plochu se liší v jednotlivých programovacích prostředích.

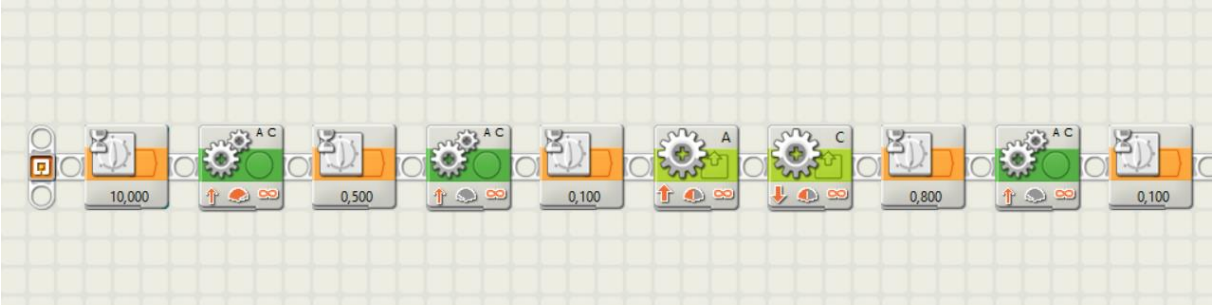
3.2.5.2. NXT-G

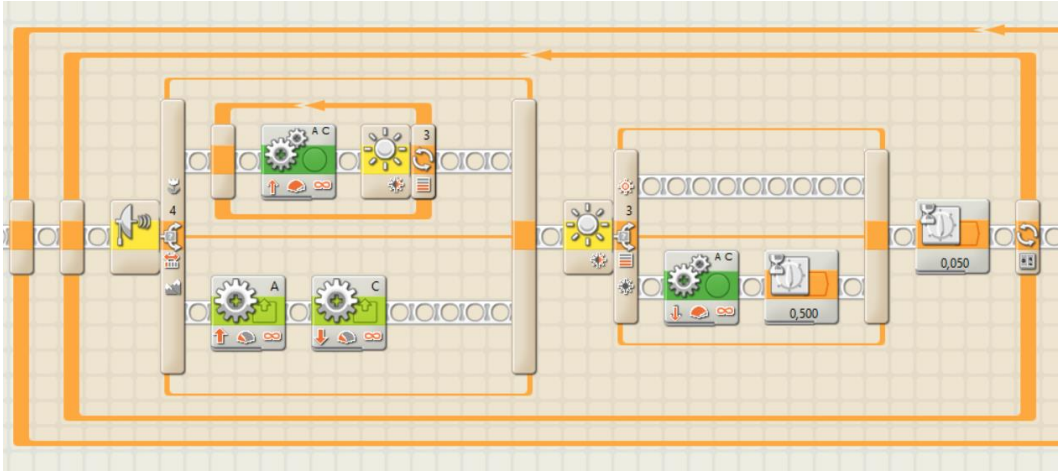
Tento kód bude prohledávat hrací plochu z míst, které jsou naznačeny na Obr. 27.

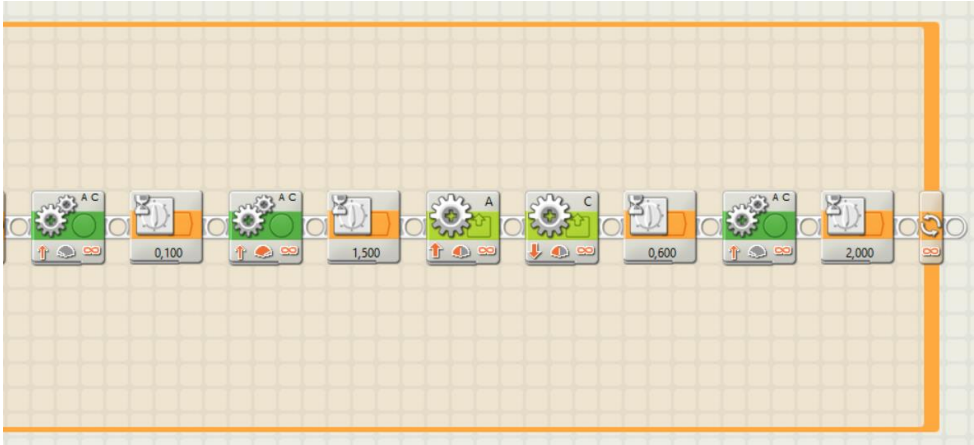


Obr. 27 Místa, ze kterých robot prohledává hrací plochu za použití NXT-G

Program, který je napsaný v NXT-G, je příliš dlouhý na to, aby se mohl umístit na jeden řádek. Při tomto rozdělení kódu, ale není narušena posloupnost jednotlivých bloků.

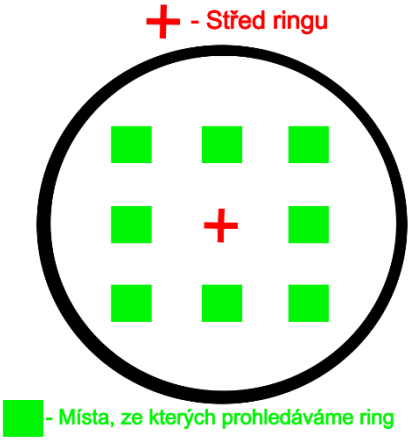






3.2.5.3. NXC

V tomto kódu se bude robot přemísťovat do poloh znázorněných v Obr. 28, aby lépe pokryl hrací plochu a bylo tak možné nalézt soupeřova robota.



Obr. 28 Místa, ze kterých robot prohledává hrací plochu za použití NXC

Samostatný kód napsaný v jazyce NXC pro řešení úlohy sumo je vidět níže. V tomto kódu se vyskytuje funkce turn(), která má za následek změnu polohy robota. Funkce turn90left() otáčí robota o 90° doleva.

```
1 void turn90left(float offset){
2       float angle = 0;
3       float speed;
4
5       long tickLast;
6       long tickNow;
7       tickLast = CurrentTick();
8
9       while(abs(angle) < 80){
10              tickNow = CurrentTick();
11              angle = angle + abs(tickNow-tickLast)*(speed-offset)/1000;
12              OnFwd(OUT_A, 50);
13              OnRev(OUT_C, 50);
14              speed = SensorHTGyro(IN_2);
15
16              Wait(10);
17              tickLast = tickNow;
18      }
19      Off(OUT_AC);
20      Wait(200);
21 }
```

```
23 void turn(int position, float offset){
24      Off(OUT_AC);
25      Wait(100);
26
27      if(position == 0){
28                  turn90left(offset);
29      }
30      RotateMotorEx(OUT_AC, 75, 314*3/2, 0,true,true);
31      Off(OUT_AC);
32      Wait(100);
33 }
```

Zajímavostí je, že funkce turn() mění polohu dvěma způsoby podle nastavení argumentu position. Pokud je tento argument 1, robot popojede pouze dopředu, pokud je 0 robot se otočí o 90° doleva a teprve po tomto úkonu popojede dopředu. Toto dvojí otáčení je z důvodu, aby bylo možné měnit pozice podle Obr. 28. Spočtení offsetu gyroskopického senzoru a následné měření úhlu je popsané v kapitole 2.5.5.. Ve funkci turn90left() se bude robot otáčet doleva tak dlouho, dokud na gyroskopickém senzoru nenaměří hodnotu 95°. Rezerva 5° stupňů je zde z důvodu nedokonalosti použitého senzoru.

Hlavní funkce tohoto programu vypadá následovně.

```
35 task main(){
36      SetSensorLowspeed(IN_4); //inicializace ultrazvukoveho senzoru
37      SetSensorLight(IN_3); //inicializace senzoru svetla
38      SetSensorHTGyro(IN_2); //inicalizace gyroskopickeho senzoru
39
40      int distance; //inicilazace promenne
41      int light; //inicilazace promenne
42      int position = 1; //inicilazace promenne
43
44      float speed; //inicalizace promenne speed pro zapisovani namerenych dat
45      float angle; //inicalizace promenne angle pro zapisovani namereneho uhlu
46      float comulative; //inicalizace promenne comulative pro vypocet offsetu
47      float offset; //inicalizace promenne offset pro ulzeni vypocitaneho offsetu
48
49      angle = 0; //nastaveni uhlu na nulu
50      comulative = 0; //nastaneni comulative na nulo
51
52      Wait(4000); //cekani z duvodu ustaleni senzoru
```

```
54      for(int i = 0; i <30; i++ ){ // smycka se 30 opakovanimi
55              comulative = comulative + SensorHTGyro(IN_2); //nacteni dat a jejijich pricteni ke comulative
56              Wait(100); //cekani
57      }
58
59      offset = comulative/30; //vypocet offsetu
60
61      PlaySound(SOUND_LOW_BEEP); //oznameni o ukononeni mereni offsetu
62
63      Wait(3000); //cekani do 10 sekund
```

```
65       Off(OUT_AC); //vypnuti motoru
66       Wait(100); //cekani
67
68       turn90left(offset); //natoceni se doleva o 90 stupnu
69
70       long tickLast; //promenna pro urceni casu
71       long tickNow;  //promenna pro urceni casu
72
73       tickLast = CurrentTick();
```

```
75      while(true){//opakuj dokola
76              distance = SensorUS(IN_4); //odecteni hodnoty ze senzoru
77              light = Sensor(IN_3); //odecteni hodnoty ze senzoru
78              speed = SensorHTGyro(IN_2); //odecteni hodnoty ze senzoru
79
80              tickNow = CurrentTick(); //zjisteni aktualniho casu
81
82              angle = angle + abs(tickNow-tickLast)*(speed-offset)/1000; //vypocet uhlu
83
84              if(distance < 20){ //pokud uvidi blyzky objekt
85                      OnFwd(OUT_AC, 100); //zapni motory
86                      while(distance < 50 && light > 50){ //dokud nenajedes na cernou caru
87                              distance = SensorUS(IN_4); //odecteni hodnoty ze senzoru
88                              light = Sensor(IN_3); //odecteni hodnoty ze senzoru
89                      }
90                      angle = 0; //vynuluj uhel
91              }else{ // jinak
92                      OnRev(OUT_A, 25); //otacej se doporava
93                      OnFwd(OUT_C, 25); //otacej se doporava
94              }
```

```
96              if(light < 50){ //pokud narazis na cernou caru
97                      OnRev(OUT_AC, 100); //jed dozadu
98                      Wait(500); //počkej
99                      angle = 0; //vynuluj uhel
100             }
101
102             if(abs(angle) > 710){ //pokud se 2x otoeci kolem vlastni osy
103                     turn(position, offset); //prejed
104                     position = (position+1)%2; //zmen pozici
105                     angle = 0; //vynuluj uhel
106             }
107
108             Wait(5);
109             tickLast = tickNow;
110
111     } //konec nekonečne smyčky
112 } //konec programu
```

Po zahájení programu musíme správně inicializovat senzory, se kterými budeme pracovat. Inicializujeme nejprve Ultrazvukový senzor na portu 4, kterého následuje inicializace světelného senzoru na portu 3 a inicializace gyroskopického senzoru na portu 2. Následně si inicializujeme proměnné distance pro zapisování hodnoty vzdálenosti změřené ultrazvukovým senzorem, light pro zapisování hodnoty odraženého světla. Nakonec následuje inicializace proměnných pro potřebu určování úhlu.

Po inicializaci proměnných následuje změření offsetu gyroskopického senzoru. A následné čekání, než robot může opustit startovní pozici a začít zápasit. Po tomto čekání se robot přesouvá do výchozí polohy hledání.

Dle if/else podmínky ve while cyklu se bude robot otáčet za předpokladu, že nenajde žádný objekt, který je mu blíže než 50 cm. Když takový objekt najde rozjede se proti němu a pokusí se ho vytlačit z hrací plochy. Když náhodou sumo zápasník narazí na konec hrací plochy, kterou pozná podle nižší hodnoty odraženého světla na světelném senzoru, pokusí se zacouvat dozadu, aby se vyhnul možnosti, že ji sám opustí. Když robot provede 2 otočení kolem vlastní osy a nenalezne na ultrazvukovém senzoru blízký objekt, změní svou pozici. Pozice, ze kterých bude robot nacházet nepřátelského robota, jsou k vidění na Obr. 28.

Výpočet úhlu natočení ve smyčce while se provádí pomocí přesného měření času trvání jedné smyčky.

3.2.5.4. PyNXC

V této části vznikl přepis programu z NXC do PyNXC, čímž jsem chtěl ukázat, že se toto programové prostředí hodí pro tento úkol. Podobně jako u NXC jsem dekomponoval kód do funkcí. Vezměte prosím na vědomí, že rozdíl je v tom, že v PyNXC se funkce deklarují pomocí klíčového slova def. Nepíše se zde jaký typ proměnné daná funkce vrací, ani se zde nepíše typ argumentů, které funkci předáváme. A proto si myslím, že Python případně PyNXC je pro studenty začínající s programováním snadnější na pochopení.

Jelikož se jedná o téměř doslovný přepis z jazyka NXC, nebudu kód napsaný v PyNXC již komentovat. Opět zde ale naleznete poznámky formou komentářů v samotném kódu. Oproti kódu v NXC jsem zde použil konstantu pro čas při určování úhlu natočení. Tento program se musel déle odlaďovat a nechová se tak přesně jako v případě NXC. Toto ale není způsobeno použitím PyNXC, ale volbou způsobu naprogramování.

Funkce turn() přemísťuje robota napříč jednotlivými místy, ze kterých náš sumo robot hledá soupeře. Funkce turn90left() pak otáčí s robotem o 90° doleva, aby bylo možné přejíždět mezi jednotlivými místy znázorněnými na Obr. 28.

```
1   def turn90left(offset):
2       angle = 0
3       while (abs(angle) < 95):
4           OnFwd(OUT_A, 50)
5           OnRev(OUT_C, 50)
6           speed = SensorHTGyro(IN_2)
7           angle = angle + 0.013 * (speed-offset)
8           Wait(10)
9       Off(OUT_AC)
10      Wait(200)
```

```
12  def turn(position, offset):
13      Off(OUT_AC)
14      Wait(100)
15      if (position == 0):
16          turn90left(offset)
17      RotateMotorEx(OUT_AC, 75, 312, 0, True, True)
18      Off(OUT_AC)
19      Wait(100)
```

Nyní se již podívejme na stěžejní část programu.

```
def main():
    SetSensorLowspeed(IN_4) #inicializace ultrazvukoveho senzoru
    SetSensorLight(IN_3) #inicializace senzoru svetla
    SetSensorHTGyro(IN_2) #inicalizace gyroskopickeho senzoru

    position = 1 #nastaveni promenne na 1

    angle = 0 #nastaveni uhlu na nulu
    comulative = 0 #nastaneni comulative na nulo

    Wait(4000) #cekani z duvodu ustaleni senzoru
```

```
    for i in range(0,30): #smycka se 30 opakovanimi
            comulative = comulative + SensorHTGyro(IN_2) #nacteni dat a jejijich pricteni ke comulative
            Wait(100) #cekani

    offset = comulative / 30 #vypocet offsetu

    PlaySound(SOUND_LOW_BEEP) #oznameni o ukoncneni mereni offsetu
    Wait(3000) #cekani do 10 sekund

    RotateMotorEx(OUT_AC, 75, 104, 0, True, True) #popojeti dopredu o 5 cm
    Off(OUT_AC) #vypnuti motoru
    Wait(100) #cekani

    turn90left(offset) #natoceni se doleva o 90 stupnu
```

```
    while (True): #opakuj dokola
        distance = SensorUS(IN_4) #odecteni hodnoty ze senzoru
        light = Sensor(IN_3) #odecteni hodnoty ze senzoru
        speed = SensorHTGyro(IN_2) #odecteni hodnoty ze senzoru

        angle = angle + 0.024 * (speed-offset) #vypocet uhlu

        if (distance < 50): #pokud uvidi blizky objekt
            OnFwd(OUT_AC, 100) #zapni motory
            while (distance < 50 and light > 50): #dokud nenajedes na cernou caru
                distance = SensorUS(IN_4) #odecteni hodnoty ze senzoru
                light = Sensor(IN_3) #odecteni hodnoty ze senzoru
        else: #jinak
            OnRev(OUT_A, 25) #otacej se doprava
            OnFwd(OUT_C, 25) #otacej se doprava

        if (light < 50): #pokud narazis na cernou caru
            OnRev(OUT_AC, 100) #jed dozadu
            Wait(500) #počkej

        if (abs(angle) > 740): #pokud se 2x otoeci kolem vlastni osy
            turn(position, offset) #prejed
            position = (position + 1) % 2 #zmen pozici
            angle = 0 #vynuluj uhel

```

Po otestování, tento kód vykonává očekáváné úkony. Proto prohlašuji, že se PyNXC hodí na naprogramování tohoto úkolu.

3.2.6. Použití gyroskopického senzoru

V případě psaní programu v NXC jsem se rozhodl využít gyroskopický senzor. V případě programu pro sumo zápasníka napsaného v NXT-G se otáčení o určitý úhel provádělo pomocí zapnutí motorů na portech A a C v opačném směru otáčení po určitý čas. Díky tomu se robot pomalu otáčí. Jelikož se, ale rychlost motorů liší podle stavu napětí baterie dochází k otáčení o nekonstantní úhel při několika zapnutích programu. Tato skutečnost stěžuje řešení tohoto úkolu. Jelikož je k měření úhlu otočení v případě NXC použit gyroskop, robot se otáčí o téměř konstantní úhel. Tato skutečnost následně zjednodušila psaní této úlohy. V případě programu NXC je potřeba kratší čas k nastavení funkcí pro otáčení, než v případě programu v NXT-G.

3.2.7. Závěr

Na začátku úkolu jsme si představili jeho zadání. Dále jsme si definovali jednotlivé podúkoly ze zadání a nalezli jejich řešení. Podúkoly byly především. Najít cíl a vytlačit ho z hrací plochy. Náš robot ale musel zůstat na hrací ploše po celou dobu trvání soutěže. Dále jsme si představili konstrukční stránku robota. Prohledávání soutěžní plochy a lokalizaci nepřítele jsme navrhli pomocí ultrazvukového sensoru. Zde jsme navrhli, aby robot nestál a nehledal soupeře pouze z jednoho místa, ale aby se sofistikovaně přemísťoval po hrací ploše. Na to, aby robot samovolně nevyjel z pracovní plochy jsme použili světelný senzor. Dále jsme navrhli pseudokód, který definuje základní chování našeho sumo robota. Kvůli nepřítomnosti druhého sumo robota, se kterým bychom mohli trénovat, jsme navrhli zjednodušení a nepřítele nahradili za papírovou krychli. Toto zjednodušení nás v počátečních fázích programování nijak neomezovalo. Ovšem pro finální odladění strategie na zápasy doporučuji zápasit proti skutečnému robotovi. Následně jsme napsali program v NXC a úspěšně ukázali přepis kódu v PyNXC. V NXT-G jsme si program kvůli přehlednosti zjednodušili na úkor prohledávání soutěžní plochy, kdy se pohybujeme pouze do 4 bodů, ze kterých provádíme hledání soupeře oproti původním 8. Kódy představené v tomto řešení jsou minimální kódy, pomocí kterých se dá splnit zadání. Ovšem každému studentovi doporučuji, aby si do programů doplnil vlastní strategii. Například je možné, aby hned na začátku robot opustil žlutý čtverec, ve kterém musí startovat. Aby se vyhnul možnosti, že soupeřův robot hned na začátku pojede právě do tohoto čtverce. Rovněž si studenti můžou upravit konstrukční stránku robota.

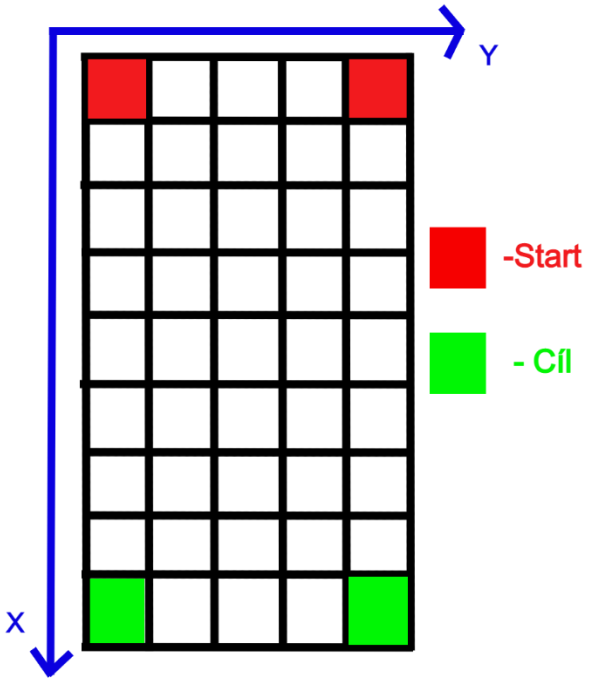
3.3. Úkol 2: Bludiště

V této úloze bude naším cílem sestavit a naprogramovat robota tak, aby co možná nejrychleji projel bludiště od začátku do cíle.

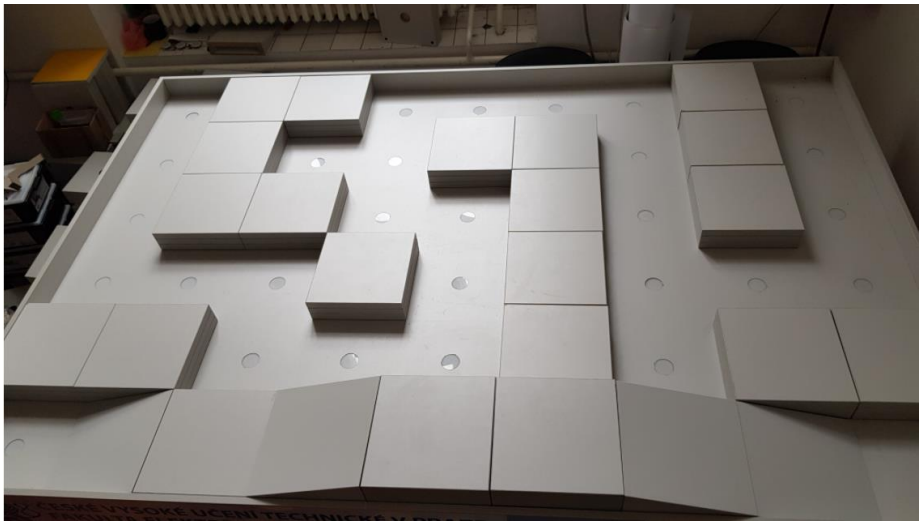
3.3.1. Zadání

Bludiště se skládá z desky o rozměrech 256 centimetrů krát 172 centimetrů a je rozděleno na 9 x 5 čtverců o délce hrany 28 centimetrů. Tyto čtverce tvoří síť polí, na níž bude bludiště sestaveno z překážek o velikosti 28 na 28 centimetrů s výškou 7,5 centimetru. Tyto překážky budou rozmístěny libovolně tak, aby volná pojezdová plocha utvořila průjezd od startu do cíle se zatáčkami a různými slepými uličkami. Robot se tedy může pohybovat pouze po pojezdových plochách, v žádném případě nesmí dojít k přejezdu přes překážku. Zároveň musí být zaručeno, že mezi startem a cílem vede trasa, kterou robot může projet.

Startovní a cílový prostor se vždy nachází v rohu této sítě tak, že mezi startem a cílen nevede přímá cesta. Možné umístění startu a cíle nalezneme na Obr. 29. Příklad postaveného bludiště pak můžeme vidět na Obr. 30.



Obr. 29 Možné umístění startu a cíle v bludišti



Obr. 30 Fotografie možného rozmístění bludiště

Roboti se při orientaci v bludišti smí dotýkat jeho stěn. Pro rychlostní zkoušku, podle které se budou udělovat body, platí následující. Student spustí robota stisknutím tlačítka, zároveň s tímto stisknutím bude aktivována časomíra. Během této soutěže je zakázána, jakákoliv komunikace mezi studentem a robotem. Zároveň je zakázáno do robota, před samotným startem, zadávat explicitní popis bludiště či jeho částí.

3.3.2. Rozbor problému

Naším úkolem bude vymyslet takový algoritmus, aby byl robot schopný projet bludiště. Jelikož se jedná o bludiště, o kterém na začátku nevíme nic, budeme muset zvolit náhodné prohledávání. K tomu, abychom mohli přímo projet bludiště, museli bychom znát jeho uspořádání dopředu a robota naprogramovat tak, aby si našel cestu virtuálně a pak se vydal nejkratší cestou. Nicméně my pracujeme bez předchozí znalosti bludiště, a proto musíme pro průjezd trasy od startu do cíle použít jeden ze dvou postupů průjezdu trasy v bludišti. První přístup, bude sledování jedné stěny. Máme na výběr, jestli budeme sledovat pravou nebo levou stěnu. Druhý přístup bude spíše ukázkový, kdy použijeme prohledávání do hloubky, přesněji jeho implementaci DFS. K tomu, abychom mohli implementovat DFS, budeme potřebovat nejprve prozkoumat bludiště a uložit si ho do paměti kostky, teprve potom bude možné tento algoritmus na prohledávání a nalezení nejkratší cesty v bludišti provést.

K tomu abychom se mohli v bludišti pohybovat ,budeme potřebovat dva servomotory, abychom docílili volné pohybu robota ve všech směrech. K zjišťování, zda můžeme čtvercem v síti tvořícím bludiště projet, nebo zda se v něm nachází překážka, použijeme ultrazvukový nebo dotykový senzor.

Především při prozkoumávání bludiště pro algoritmus DFS budeme potřebovat dělat přesné 90° zatáčky, k tomuto úkolu použijeme gyroskopický senzor, který nám umožní úhel otočení robota měřit, a tak eliminovat nepřesnost servomotorů.

3.3.3. Konstrukce

Robot je konstruovaný tak, aby mohl projíždět bludištěm a hledat v něm cestu. Robot obsahuje tři servomotory. Dva servomotory slouží k pohybu robota, třetí pak slouží k natáčení ultrazvukového senzoru, aby se robot nemusel natáčet celý v případě hledání další možné cesty v následujícím poli. Model robota uzpůsobeného k prohledávání bludiště je na Obr. 31 a Obr. 32.

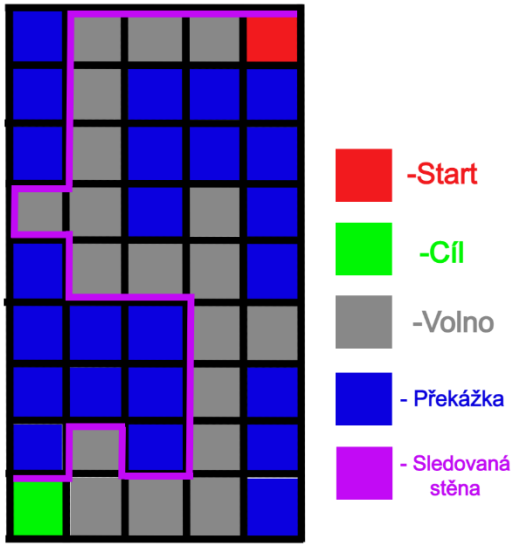
Obr. 31 Konstrukce robota na průjezd bludištěm pohled I.

Obr. 32 kostrukce robota na průjezd bludištěm pohled II.

Ke kostce NXT jsou připojené servomotory na portech A a C pro jízdu robota v bludišti. Na portu B je připojený servomotor, který otáčí s ultrazvukovým senzorem. Tento senzor je připojený na port 4. Na portu 2 je připojený gyroskopický senzor. Pomocí tohoto senzoru bude robot zatáčet podobně jako v kapitole 3.2. Ke stavbě robota pro jízdu v bludišti vznikl návod, který je k nalezení v digitální příloze této práce.

3.3.4. Sledování stěny

V tomto případě jsem se rozhodl sledovat pravou stěnu z pohledu robota. Ilustrace, jak takový robot pojede od startu do cíle, je na Obr. 33.



Obr. 33 Ilustrace bludiště společně se stěnou, která se sleduje

Sledování stěny je účinná, i když může být pomalá, metoda, jak prohledat bludiště a dostat se od startu do cíle. Jedno vylepšení může být například v tom, že nemusíme zajíždět do slepých uliček, které jsou pouze jedno pole hluboké. Tyto uličky nemusíme prozkoumávat robotem, stačí zkontrolovat ultrazvukovým senzorem, že jsou tyto uličky skutečně hluboké pouze jedno pole. Pokud jsou tyto pole skutečně hluboké jen jedno pole a pokud máme možnost jet jiným směrem, budeme tímto směrem pokračovat. Jestliže tuto možnost nemáme do čtverečku stejně zajedeme. Pokud se tím dostaneme do slepé uličky a pořád nejsme v cíli, musíme se otočit o 180 stupňů, z uličky vyjet a pokračovat v hledání cesty.

To, zda jsme v cíli, musíme poznat podle toho, že start a cíl se nachází v jednom z rohů. Budeme nám tedy stačit počítat souřadnice v x-ové a y-ové ose, tak jak jsou naznačené na Obr. 33. Podle toho, že dosáhneme souřadnic rohů, pak můžeme říct, že se nacházíme v cíli.

3.3.4.1. Programovací část

Jako obvykle začnu pseudokódem, následně popíšu NXT-G s drobnými zjednodušeními kvůli složitosti napsání kódu v tomto grafickém programu. Dále si ukážeme kód napsaný v NXC. Jako poslední si ukážeme poupravený kód v jazyce PyNXC oproti NXC. Samotné kódy jsou popsané v předložené ukázce programu formou poznámek.

3.3.4.1.1. Nxt-G Pseudokód

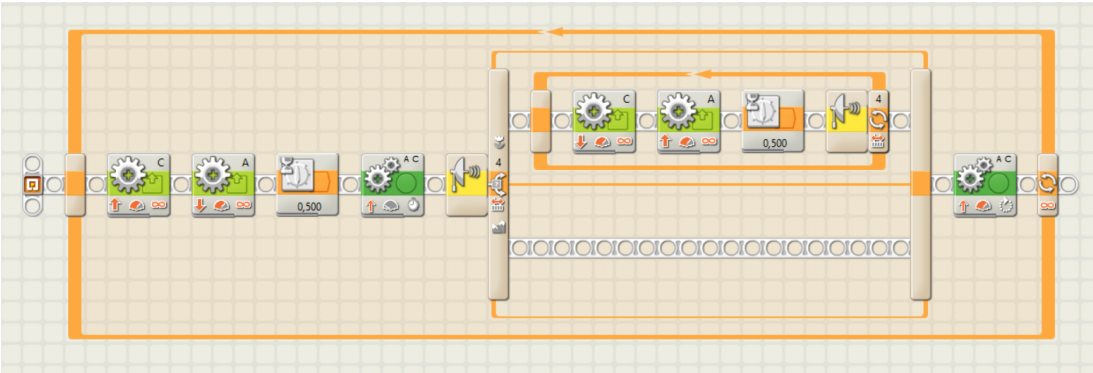
Jelikož je kód v jazyce NXT-G značně odlišný od kódu napsaného ve NXC a PyNXC, rozhodl jsem se, že vytvořím odlišný pseudokód, na kterém vše lépe popíšu.

```
Zapnutí programu
Prováděj pořád dokola:
        Otoč se doleva o 90°
        Prováděj dokavaď nenaměříš na ultrazvukovém senzoru více než 10 cm:
                Otoč se doprava o 90
        Jeď dopředu o 28 cm.
```

3.3.4.1.2. NXT-G

Jako obvykle zavedeme v programu NXT-G zjednodušení. Zde ukážu pouze ten nejednoduší kód, který je schopný sledovat stěnu. Nebude zde naprogramované přesné zatáčení pomocí gyroskopického senzoru, nebo nezajíždění do uliček hlubokých jedno pole. Kód je osekaný na základní chování robota, aby sledoval pravou stěnu. Robot tedy nedělá nic jiného, než že se otočí doprava a zkontroluje, zda nemůže jet. Pokud může jet dopředu tak jede, čímž se posune na další pole. Pokud se před ním nachází překážka robot se otočí doprava o 90° a zkontroluje, zda může jet. Doprava se otáčí tak dlouho dokud nenarazí na volnou cestu, po které by se mohl posunout o čtvereček dopředu. Až takovou cestu najde vydá se po ní. Takto se to opakuje v průběhu kódu několikrát.

Zároveň se při plnění úkolu pomocí robota z 3.3.3 předpokládá, že ultrazvukový senzor je natočený přímo před robota.



3.3.4.1.3. Pseudokód pro NXC a PyNXC

V této části si ukážeme, jak by se měl chovat takový robot, aby projel bludiště co možná nejrychleji, aniž by měl libovolné informace o povaze bludiště. Pseudokód odpovídá chování robota popsaného na začátku této podkapitoly.

```
pockej 2 sekundy
spocitej offset gyroskopickeho senzoru

Opakuj dokola:
        Když dojedu do cíle:
                Vypni program

        Když se robot nepohnul:
                Otoč ultrazvukovým senzorem doleva
                Změř vzdálenost od překážky a uzlož ji
                Vrať ultrazvukový senzor do výchozí polohy

        Když je vzdálenost větší jak 10 cm a robot se nepohnul:
                Když je vzádlenost větší jak 35:
                        Robot se otočí o 90° vlevo
                        Popojeď dopředu
                Jinak:
                        ulož si volno vlevo od robota

        Změř vzdálenost od překážky a uzlož ji

        Když je vzdálenost větší jak 10 cm a robot se nepohnul:
                Když je vzádlenost větší jak 35:
                        Popojeď dopředu
                Jinak:
                        ulož si volno před robotem


        Když se robot nepohnul:
                Otoč ultrazvukovým senzorem doprava
                Změř vzdálenost od překážky a uzlož ji
                Vrať ultrazvukový senzor do výchozí polohy

        Když je vzdálenost větší jak 10 cm a robot se nepohnul:
                Když je vzádlenost větší jak 35:
                        Robot se otočí o 90° vpravo
                        Popojeď dopředu
                Jinak:
                        ulož si volno vpravo od robota

        Pokud se robot nepohnul a je vedle něj místo:
                Zajeď to volného místa

        Pokud se robot nepohnul:
                otoč se o 180° doleva a posuň se dopředu

        Změn souřadnice robota, z důdoukontorly dosažení cíle
```

Když se robot bude posouvat dopředu chceme, aby se posunul o délku strany jednoho pole a tím se posunul do pole následujícího. Musí tedy urazit vzdálenost 28 centimetrů. Jelikož neexistuje příkaz, který by umožnil, aby se robot posunul přímo o 28 centimetrů. Musíme spočítat o kolik stupňů se musí kola robota otočit. Vzorec pro počet stupňů je v (4).

$$stupně = \frac{potřebnáVzálenost}{2 \cdot \pi \cdot poloměrKola} \cdot 360° \quad (4)$$

Číselné dosazení pro robota z 3.3.3, který má průměr kol 55 mm je v (5).

$$stupně = \frac{280}{2 \cdot \pi \cdot \frac{55}{2}} \cdot 360° = 584° \quad (5)$$

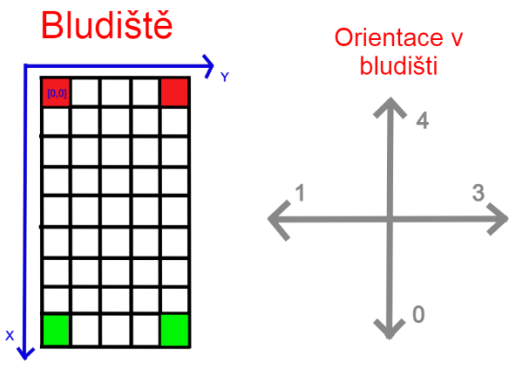
3.3.4.1.4. NXC

Následující kód je rozdělený do funkcí pro větší přehlednost. Zároveň jsem se zde rozhodl využívat globální proměnné, z důvodu čitelnějšího kódu. Použité globální proměnné:

```
float offset;
int orientation;

int positionY;
int positionX;

```

Proměnná offset slouží k uložení offsetu vypočteného pro potřeby přesného měření úhlové rychlosti.

Proměnná orientation uchovává natočení robota. Natočení robota je znázorněno na Obr. 34.



Obr. 34 Orientace robota v bludišti

Proměnné positionX a positionY ukládá pozici robota v mřížce, tak jak je znázorněná v Obr. 29.

Dále uvedu funkce a stručně popíšu jejich účel.

```
void vypocti_offset(){
        float comulative;
        for(int i = 0; i <30; i++ ){
                comulative = comulative + SensorHTGyro(IN_2);
                Wait(100);
        }
        offset = comulative/30;
        PlaySound(SOUND_LOW_BEEP);
}
```

Vypocti_offset() vypočítá offset pro práci s gyroskopickým senzorem. Popis této funkci nalezneme v 2.5.5.1.3.

```
17 void turn90left(){
18      float angle = 0;  //inicializace promenne
19      float speed = 0;  //inicializace promenne
20      while(abs(angle) < 95){ //opakuj porad dokola, dokavad nenameris vetsi uhel
21             OnFwd(OUT_A, 50); //pustim motor A dopredu
22             OnRev(OUT_C, 50); //pustim motor C dozadu
23             speed = SensorHTGyro(IN_2); //odecteni hodnoty z gyroskopickeho senzoru
24             angle = angle + 0.013*(speed-offset); //vypocet uhlu
25             Wait(10); //kratka doba cekani
26      } //konec smycky
27      Off(OUT_AC); //vypni motory A a B
28      if(orientation == 0){ //změna orientace robota z duvodu otoceni
29            orientation = 3;
30      }else{
31            orientation = orientation-1;
32      }
33 }
```

Turn90left() otočí robotem o 90° doleva a zároveň upraví orientaci robota.

```
35 void turn180(){
36      float angle = 0; //inicializace promenne
37      float speed = 0; //inicializace promenne
38      while(abs(angle) < 190){ //opakuj porad dokola, dokavad nenameris vetsi uhel
39             OnFwd(OUT_A, 50); //pustim motor A dopredu
40             OnRev(OUT_C, 50); //pustim motor C dozadu
41             speed = SensorHTGyro(IN_2); //odecteni hodnoty z gyroskopickeho senzoru
42             angle = angle + 0.013*(speed-offset); //vypocet uhlu
43             Wait(10); //kratka doba cekani
44      }
45      Off(OUT_AC); //vypni motory A a B
46
47      orientation = abs(orientation+2)%4;  //změna orientace robota z duvodu otoceni
48 }
```

Turn180() otočí robotem o 180° doleva a zároveň upraví orientaci robota.

```
50 void turn90right(){
51      float angle = 0; //inicializace promenne
52      float speed = 0; //inicializace promenne
53      while(abs(angle) < 95){ //opakuj porad dokola, dokavad nenameris vetsi uhel
54             OnFwd(OUT_C, 50); //pustim motor C dopredu
55             OnRev(OUT_A, 50); //pustim motor A dozadu
56             speed = SensorHTGyro(IN_2); //odecteni hodnoty z gyroskopickeho senzoru
57             angle = angle + 0.013*(speed-offset); //vypocet uhlu
58             Wait(10); //kratka doba cekani
59      }
60      Off(OUT_AC); //vypni motory A a B
61      orientation = abs(orientation+1)%4; //změna orientace robota z duvodu otoceni
62 }
```

Turn90right() otočí robotem o 90° doprava a zároveň upraví orientaci robota.

```
64 void changePosition(){
65      if (orientation==0){ //pokud je orientation 0
66         positionX = positionX+1; //změna pozice v ose x
67         positionY = positionY; //pozice y zustava
68      }else if (orientation==1){ //pokud je orientation 1
69         positionX = positionX; //pozice x zustava
70         positionY = positionY+1; //změna pozice v ose y
71      }else if (orientation==2){ //pokud je orientation 2
72         positionX = positionX-1; //změna pozice v ose x
73         positionY = positionY; //pozice y zustava
74      }else{ //pokud je orientation 3
75         positionX = positionX; //pozice x zustava
76         positionY = positionY-1; //změna pozice v ose y
77      }
78 }
```

ChangePosition() změní pozici robota z důvodu ukončení programu, pokud robot dojede do cíle.

```
80 task main(){
81     SetSensorHTGyro(IN_2); //inicalizace gyroskopickeho senzoru
82     SetSensorLowspeed(IN_4); //inicalizace ultrazvukoveho senzoru
83
84     Wait(2000); //pocka se na vzadaleni ruky od robota
85     vypocti_offset(); //vypocita se offset
86
87     int move; //inicializace promenne
88     int free; //inicializace promenne
89     int distance; //inicializace promenne
```

```
91     while(true){ //nekonecna smycka
92                 if((positionX==8 && positionY == 0) || (positionX==8 && positionY == 4)){ //kdyz je robot v cili
93                                 PlaySound(SOUND_LOW_BEEP); //robot oznami dojeti
94                                 Wait(1000); //pocka se na dohrati tonu
95                                 Stop(true); //robot zastavi
96                 }
97
98                 move = 0; //nasatveni move na nulu
99                 free = 0; //nastaveni free na nulu
100
101                 if(move == 0){ //pokud se robot nepohnul
102                     RotateMotor(OUT_B, 75, 90); //robot natoci ultrazvukovy senzor
103                     distance = SensorUS(IN_4); //nacteni hodnoty ze ultrazvukoveho senzoru
104                     RotateMotor(OUT_B, 75, -90); //robot vrati ultrazvukovy senzor do puvodni polohy
105                 }
106
107                 if(distance > 10 && move == 0){ //pokud se robot nepohnul a je vlevo volno
108                             if(distance > 35){ //pokud je vzdalenost vetsi nez 35
109                                 turn90left(); //robot se otoci o 90 stupnu doleva
110                                 RotateMotorEx(OUT_AC, 75, 584, 0,true,true); //otoci motory A a C dopredu o 584 stupnu
111                                 move = 1; //robot se uz pohnul
112                             }else if(free == 0){ //pokud neni volno pred robotem
113                                  free = 2; //volno je nalevo od robota
114                             }
115                 }
```

```
116
117                 distance = SensorUS(IN_4); //nacteni hodnoty ze ultrazvukoveho senzoru
118
119                 if(distance > 10){ //pokud je vzdalenost vetsi nez 10
120                             if(distance > 35){ //pokud je vzdalenost vetsi nez 35
121                                 RotateMotorEx(OUT_AC, 75, 584, 0, true, true); //otoci motory A a C dopredu o 584 stupnu
122                                 move = 1; //robot se uz pohnul
123                             }else{ //jinak
124                                  free = 1; //volno je pred robotem
125                             }
126                 }
127
```

```
128                 if (move == 0){
129                      RotateMotor(OUT_B, 75, -90); //robot natoci ultrazvukovy senzor
130                      distance = SensorUS(IN_4); //nacteni hodnoty ze ultrazvukoveho senzoru
131                      RotateMotor(OUT_B, 75, 90); //robot vrati ultrazvukovy senzor do pivodni polohy
132                 }
133
134                 if(distance > 10 && move == 0){ //pokud se robot nepohnul a je vpravo volno
135                            if(distance > 35){ //pokud je vzdalenost vetsi nez 35
136                                 turn90right(); //robot se otoci o 90 stupnu doprava
137                                 RotateMotorEx(OUT_AC, 75, 584,0,true,true); //otoci motory A a C dopredu o 584 stupnu
138                                 move = 1; //robot se uz pohnul
139                            }else if(free == 0){ //pokud neni volno napravo nebo pred robotem
140                                  free = 3; //volno je napravo od robota
141                            }
142                 }
143
```

```
144                 if(move == 0){ //pokud se robot nepohnul
145                         if(free == 1){ //pokud je volno pred robotem
146                                 RotateMotorEx(OUT_AC, 75, 584,0,true,true); //otoci motory A a C dopredu o 584 stupnu
147                                 move = 1; //robot se uz pohnul
148                         }else if(free == 2){ //pokud je volno nalevo od robota
149                               turn90left(); //otoceni robota doleva o 90 stupnu
150                               RotateMotorEx(OUT_AC, 75, 584,0,true,true); //otoci motory A a C dopredu o 584 stupnu
151                               move = 1; //robot se uz pohnul
152                         }else if(free == 3){ //pokud je volno napravo od robota
153                               turn90right(); //otoceni robota doprava o 90 stupnu
154                               RotateMotorEx(OUT_AC, 75, 584,0,true,true); //otoci motory A a C dopredu o 584 stupnu
155                               move = 1; //robot se uz pohnul
156                         }
157                 }
158
159                 if(move == 0){ //pokud se robot nepohnul
160                         turn180(); //otoceni robota doleva o 180 stupnu
161                         RotateMotorEx(OUT_AC, 75, 584, 0, true, true); //otoci motory A a C dopredu o 584 stupnu
162                         move = 1; //robot se uz pohnul
163                 }
164
165                 changePosition(); //aktualizuj souradnice robota
166
167   }//konec nekonecne smycky
168 }//konec programu
```

Robot zbytečně nezajíždí do slepých uliček. Jinak se chová jako sledovač pravé stěny. Jeho funkce je popsaná v 3.3.4.1.3 v pseudokódu.

Pro posun robota dopředu je použitý příkaz RotateMotorEx(). Tento příkaz slouží k nastavení úhlu otočení několika servomotorů a zároveň zajišťuje synchronizaci jejich pohybu. Mezi jeho nastavení patří například turnPoint, který je obdobou turning z NXT-G. Určuje, na kterou stranu má robot zatáčet. Já ho ponechal nastavený na 0, tedy robot jede rovně. První true v argumentech říká, zda se mají motory při běhu synchronizovat a druhé pak zda se servomotory mají po otočení o 584° vypnout a držet pozici.

3.3.4.1.5. PyNXC

Následující kód tak, jako kód v NXC, je rozdělený do funkcí pro větší přehlednost. Zároveň jsem se zde rozhodl, trošku netradičně pro jazyk Python, použít globální proměnné, z důvodu čitelnějšího kódu. Jedná se o přepis programu z kapitoly 3.3.4.1.4. do jazyka PyNXC.

Použité globální proměnné:

```
1   offset = 0 #inicializace promenne
2   orientation = 0 #inicializace promenne
3   positionY = 0 #inicializace promenne
4   positionX = 0 #inicializace promenne
5
```

Proměnná offset slouží k uložení offsetu vypočteného pomocí vypočti_offset() pro potřeby přesného měření úhlové rychlosti.

Proměnná orientation uchovává natočení robota v souřadné mřížce. Natočení robota a jeho číselné kódování je znázorněno na Obr. 34.

Proměnné positionX a positionY ukládá pozici robota v mřížce tak, jak je znázorněná v Obr. 29.

Dále uvedu funkce a stručně popíšu jejich účel.

```
6   def vypocti_offset():
7       comulative = 0
8       for i in range(0,30):
9           comulative = comulative + SensorHTGyro(IN_2)
10          Wait(100)
11      offset = comulative / 30
12      PlaySound(SOUND_LOW_BEEP)
```

Vypocti_offset() vypočítá offset pro práci s gyroskopem. Popis této funkci nalezneme v 2.5.5.1.3.

```
14  def turn90left():
15      angle = 0 #nastaveni promenne angle na nulu
16      while (abs(angle) < 95): #opakuj porad dokola, dokavad nenameris vetsi uhel
17          OnFwd(OUT_A, 50) #pustim motor A dopredu
18          OnRev(OUT_C, 50) #pustim motor C dozadu
19          speed = SensorHTGyro(IN_2) #odecteni hodnoty z gyroskopickeho senzoru
20          angle = angle + 0.013 * (speed-offset) #vypocet uhlu
21          Wait(10) #kratka doba cekani
22      Off(OUT_AC) #vypni motory A a B
23      if (orientation == 0): #zmena orientace robota z duvodu otoceni
24          orientation = 3
25      else:
26          orientation = orientation-1
```

Turn90left() otočí robotem o 90° doleva a zároveň upraví orientaci robota.

```
28  def turn180():
29      angle = 0 #nastaveni promenne angle na nulu
30      while (abs(angle) < 190): #opakuj porad dokola, dokavad nenameris vetsi uhel
31          OnFwd(OUT_A, 50) #pustim motor A dopredu
32          OnRev(OUT_C, 50) #pustim motor C dozadu
33          speed = SensorHTGyro(IN_2) #odecteni hodnoty z gyroskopickeho senzoru
34          angle = angle + 0.013 * (speed-offset) #vypocet uhlu
35          Wait(10) #kratka doba cekani
36      Off(OUT_AC) #vypni motory A a B
37      orientation = abs(orientation + 2) % 4 #zmena orientace robota z duvodu otoceni
```

Turn180() otočí robotem o 180° doleva a zároveň upraví orientaci robota.

```
def turn90right():
    angle = 0 #nastaveni promenne angle na nulu
    while (abs(angle) < 95): #opakuj porad dokola, dokavad nenameris vetsi uhel
        OnFwd(OUT_C, 50) #pustim motor C dopredu
        OnRev(OUT_A, 50) #pustim motor A dozadu
        speed = SensorHTGyro(IN_2) #odecteni hodnoty z gyroskopickeho senzoru
        angle = angle + 0.013 * (speed-offset) #vypocet uhlu
        Wait(10) #kratka doba cekani
    Off(OUT_AC) #vypni motory A a B
    orientation = abs(orientation + 1) % 4 #zmena orientace robota z duvodu otoceni
```

Turn90right() otočí robotem o 90° doprava a zároveň upraví orientaci robota.

```
def changePosition():
    if (orientation == 0): #pokud je orientation 0
        positionX = positionX+1 #zmena pozice v ose x
        positionY = positionY #pozice y zustava nezmenena
    elif (orientation == 1): #pokud je orientation 1
        positionX = positionX #pozice x zustava nezmenena
        positionY = positionY-1 #zmena pozice v ose y
    elif (orientation == 2): #pokud je orientation 2
        positionX = positionX-1 #zmena pozice v ose x
        positionY = positionY #pozice y zustava nezmenena
    else: #pokud je orientation 3
        positionX = positionX #pozice x zustava nezmenena
        positionY = positionY+1 #zmena pozice v ose y
```

ChangePosition() změní pozici robota z důvodu ukončení programu, pokud robot dojede do cíle.

```
def main():
    SetSensorHTGyro(IN_2) #inicalizace gyroskopickeho senzoru
    SetSensorLowspeed(IN_4) #inicalizace ultrazvukoveho senzoru

    vypocti_offset() #pocka se na vzadaleni ruky od robota
    Wait(2000) #vypocita se offset

    while (True): #nekonecna smycka
        if((positionX == 8 and positionY == 0) or (positionX == 8 and positionY == 4)): #kdyz je robot v cili
            PlaySound(SOUND_LOW_BEEP) #robot oznami dojeti
            Wait(1000) #pocka se na dohrati tonu
            Stop(true) #robot zastavi

        move = 0 #nastaveni promenne move na nulu
        free = 0 #nastaveni promenne free na nulu
```

```
        if (move == 0): #pokud se robot nepohnul
            RotateMotor(OUT_B, 75, 90) #robot natoci ultrazvukovy senzor
            distance = SensorUS(IN_4) #nacteni hodnoty ze ultrazvukoveho senzoru
            RotateMotor(OUT_B, 75, -90) #robot vrati ultrazvukovy senzor do pivodni polohy

        if (distance > 10 and move == 0): #pokud se robot nepohnul a je vlevo volno
            if (distance > 35): #pokud je vzdalenost vetsi nez 35
                turn90left() #robot se otoci o 90 stupnu doleva
                RotateMotorEx(OUT_AC, 75, 584, 0,  True, True) #otoci motory A a C dopredu o 584 stupnu
                move = 1 #robot se uz pohnul
            elif(free == 0): #pokud neni volno pred robotem
                free = 2 #volno je nalevo od robota
```

```
93          distance = SensorUS(IN_4)  # nacteni hodnoty ze ultrazvukoveho senzoru
94
95          if (distance > 10): #pokud je vzdalenost vetsi nez 10
96              if (distance > 35): #pokud je vzdalenost vetsi nez 35
97                  RotateMotorEx(OUT_AC, 75, 584, 0, True, True) #otoci motory A a C dopredu o 584 stupnu
98                  move = 1 #robot se uz pohnul
99              else: #jinak
100                 free = 1 #volno je pred robotem
101
102         if (move == 0): #pokud se robot nepohnul
103             RotateMotor(OUT_B, 75, -90) #robot natoci ultrazvukovy senzor
104             distance = SensorUS(IN_4) #nacteni hodnoty ze ultrazvukoveho senzoru
105             RotateMotor(OUT_B, 75, 90) #robot vrati ultrazvukovy senzor do pivodni polohy
106
107         if (distance > 10 and move == 0): #pokud se robot nepohnul a je vpravo volno
108             if (distance > 35): #pokud je vzdalenost vetsi nez 35
109                 turn90right() #robot se otoci o 90 stupnu doleva
110                 RotateMotorEx(OUT_AC, 75, 584, 50,  True, True) #otoci motory A a C dopredu o 584 stupnu
111                 move = 1 #robot se uz pohnul
112             elif(free == 0): #pokud neni volno pred robotem
113                 free = 3 #volno je napravo od robota
```

```
115         if(move == 0): #pokud se robot nepohnul
116             if(free == 1): #pokud je volno pred robotem
117                 RotateMotorEx(OUT_AC, 75, 584, 0,  True, True) #otoci motory A a C dopredu o 584 stupnu
118                 move = 1 #robot se uz pohnul
119             elif(free == 2): #pokud je volno nalevo od robota
120                 turn90left() #otoceni robota doleva o 90 stupnu
121                 RotateMotorEx(OUT_AC, 75, 584, 0,  True, True) #otoci motory A a C dopredu o 584 stupnu
122                 move = 1 #robot se uz pohnul
123             elif (free == 3): #pokud je volno napravo od robota
124                 turn90right() #otoceni robota doprava o 90 stupnu
125                 RotateMotorEx(OUT_AC, 75, 584, 0,  True, True) #otoci motory A a C dopredu o 584 stupnu
126                 move = 1 #robot se uz pohnul
127
128         if (move == 0): #pokud se robot nepohnul
129             turn180() #otoceni robota doleva o 180 stupnu
130             RotateMotorEx(OUT_AC, 75, 584, 0,  True, True) #otoci motory A a C dopredu o 584 stupnu
131             move = 1 #robot se uz pohnul
132
```

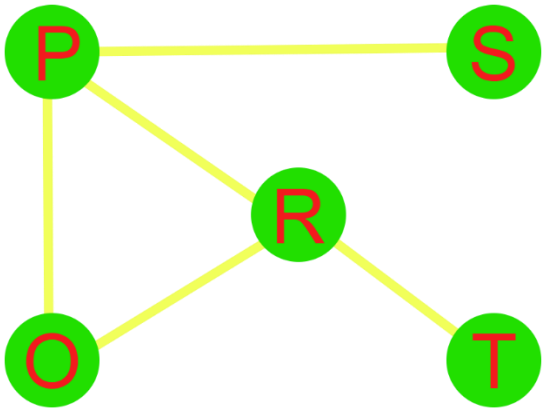
Robot zbytečně nezajíždí do slepých uliček. Jinak se chová jako sledovač pravé stěny. Jeho funkce je popsaná v 3.3.4.1.3 v pseudokódu.

Pro posun robota dopředu je použitý příkaz RotateMotorEx(). Tento příkaz slouží k nastavení úhlu otočení servomotorů a zároveň zajišťuje synchronizaci jejich pohybu. Mezi jeho nastavení patří například turnPoint, který je obdobou turning z NXT-G. Určuje, na kterou stranu má robot zatáčet. Já ho ponechal nastavený na 0, kdy robot jede rovně. První True v tomto příkazu říká, zda se mají servomotory při běhu synchronizovat a druhé pak, zda se motory mají po otočení o 584° vypnout.

Kód napsaný v jazyce NXC se podařilo úspěšně přepsat do PyNXC a následně otestovat na jízdě v bludišti.

3.3.5. Deep First Search (DFS)

Prohledávání do hloubky a nalezení prvního cíle tak by se dal přeložit Deep First Search. Tento algoritmus slouží k nalezení cesty v grafu. Za předpokladu, že je graf acyklický neboli že neobsahuje smyčky, je nalezená cesta i nejkratší možnou cestou. Napřed se podívejme, co je to prohledávání do hloubky. Vezměme si jako příklad mapu měst na Obr. 35.



Obr. 35 Mapa měst

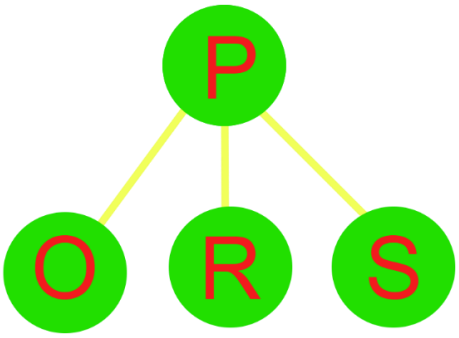
Město	Zkratka
Prostějov	P
Ostrava	O
Rakytník	R
Svitavy	S
Tábor	T

Tabulka 3: Legenda měst

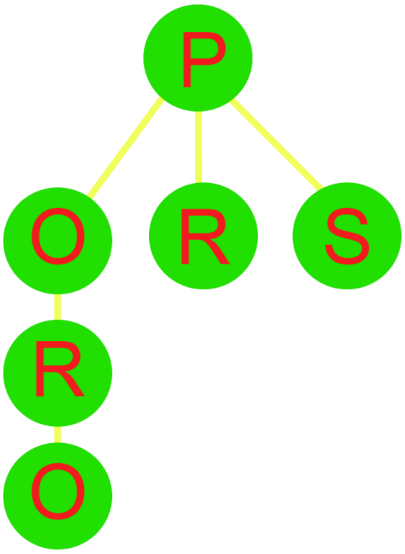
Jednotlivé body v grafu na Obr. 35 znázorňují města a jednotlivé úsečky mezi nimi znázorňují cesty mezi těmito městy. Legenda je k nalezení v Tabulka 3. Pro naše účely předpokládejme, že chceme najít cestu z Prostějova do Tábora. Začneme tedy v Prostějově, to je náš první bod (Obr. 36). Jak vidíme z Prostějova můžeme jít do Ostravy, Svitav, nebo Rakytníku, proto tyto města zobrazíme do našeho grafu (Obr. 37). Nyní si náhodně vybereme mezi Rakytníkem, Ostravou nebo Svitavami. Zvolíme třeba Ostravu. Jak vidíme, tak z Ostravy můžeme jít do Rakytníku, nebo do Prostějova. Z Prostějova jsme vyšli a do Rakytníku se můžeme dostat přímo z Prostějova, tak je již nezahrneme do našeho grafu (Obr. 37). Kdybychom je totiž opět zahrnuli do našeho stromu zacyklili bychom se, protože následně bychom z nich chtěli jít zpět do Ostravy. A takhle pořád dokola. Ilustraci tohoto problému můžeme nalézt na Obr. 38. Z Ostravy tedy nemůžeme jít nikam dál. Proto si ji označíme jako uzavřený uzel a vrátíme se zpět do Prostějova. Z Prostějova tedy můžeme jít do Ostravy, kterou máme označenou jako uzavřený uzel. Dále můžeme jít již pouze do Rakytníku nebo Svitav. Z Rakytníku můžeme jít do Ostravy, kam nepůjdeme, protože jsme ji už navštívili, nebo můžeme jít do Tábora. Půjdeme tedy do Tábora a tím pádem dorazíme do cíle (Obr. 39). Povšimněte si, že z Prostějova můžeme jít ještě do Svitav, ale zde by se opakovala situace jako u Ostravy, a dospěli bychom k tomu, že je také uzavřeným uzlem. Kdybychom se touto cestou vydali, pokračovali bychom v prohledávání do hloubky, pokud ne skončíme v Táboře a provedli jsme algoritmus DFS. Tedy při prvním nalezení cíle končíme v prohledávání.

Samozřejmě graf popsaný v Obr. 35 není acyklický, ale obsahuje smyčku Prostějov, Rakytník a Ostrava.

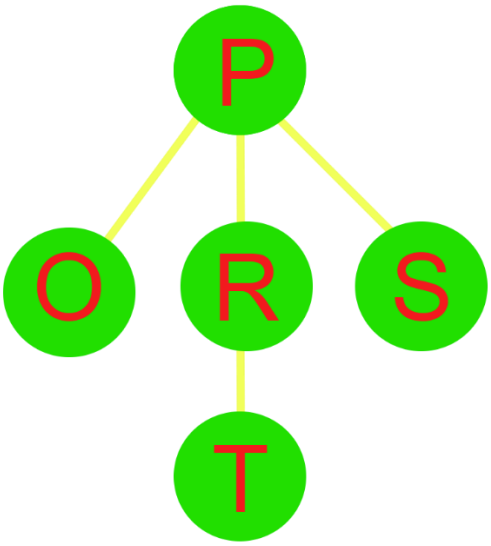
Obr. 36 První krok



Obr. 37 Druhý krok



Obr. 38 Ilustrace zacykleni



Obr. 39 Finální graf DFS

3.3.5.1. Pseudokód, jak DFS algoritmus funguje

V této podkapitole si popíšeme na pseudokódu, jak funguje algoritmus DFS, který je použitý v této bakalářské práci.

```
dalšíSouřadnice = startovníSouřadnice

Provaděj dokola, dokud nejsi v cili:
        staráSouřadnice = dalšíSouřadnice

        nalezni, kam se dá z tohoto bodu jit a jeste jsi tam nebyl
        když lze kam jit:
                Ulož si odkud jsi přišel
        Jinak:
                Vrať se o políčko zpět
```

3.3.5.2. Pseudokód pro programovaní robota

V tomto případě musíme rozdělit náš úkol na dva samostatné programy. V prvním kódu si robot sám prohledá bludiště. Druhý program pak nalezne v zjištěném popisu bludiště nejkratší cestu mezi startem a cílem pomocí algoritmu DFS. Po nalezení nejkratší cesty bude tento program řídit robota, tak aby touto cestou jel. Od těchto programů si slibujeme, že po prvotním prohledání bludiště budou robota ovládat tak, že robot projede celé bludiště rychleji než za použití programu na sledování zdi.

3.3.5.2.1. Prohledávací část pro prozkoumání bludiště

V této části se prohledá bludiště. Zvolili jsme, že prohledáme 30 políček, pomocí sledování pravé stěny. V takovémto případě projedeme 66 % bludiště, za předpokladu, že v bludišti nebude žádná překážka. Což považuji, za rozumnou část prohledaného bludiště za předpokladu, že se v bludišti dozajisté musí nacházet alespoň nějaké překážky, aby mezi startem a cílem nevedla přímá cesta. Tento program vychází ze sledování pravé stěny bludiště, kdy vzhledem k povolené konfiguraci robota pokryji co možná největší plochu bludiště.

```
Začátek programu
inicializace gyroskopického senzoru
inicializace ultrazvukoveho senzoru

pockej 2 sekundy
spocitej offset gyroskopickeho senzoru

inicalizuje pole maze pro ulozeni rozmístění bludiště

Opakuj 35:
        Když jsi se ještě nepohnul:
                Otoč ultrazvukovým senzorem doprava
                Změř vzdálenost a ulož ji
                Vrať ultrazvkový senzor do výchozí polohy

        Když je vzálenost větší jak 10 a ještě se robot nepohnul:
                Otoč se o 90° doprava
                Popojeď dopředu

        Změř vzdálenost a ulož ji
        Když je vzálenost větší jak 10:
                Popojeď dopředu

        Když jsi se ještě nepohnul:
                Otoč ultrazvukovým senzorem doleva
                Změř vzdálenost a ulož ji
                Vrať ultrazvkový senzor do výchozí polohy

        Když je vzálenost větší jak 10 a ještě se robot nepohnul:
                Otoč se o 90° doleva
                Popojeď dopředu

        Když se robot ještě pohnul:
                Otoč se o 180° doleva

        Aktualizj polohu robota a ulož prohledane pole do maze

Ulož pole maze do paměti robota
```

3.3.5.2.2. Program jízdy bludištěm

Robot nejprve prohledá nalezenou mapu z předchozího úkolu a následně řídí jízdu robota.

```
Začátek programu
Načti bludiště
Připrav pole pro zapisovani hodnot z mapy

Zadej souřadnice startu, cíle a orientaci robota

dalšíSouřadnice = startovníSouřadnice

Provaděj dokola, dokud nejsi v cili:
        staráSouřadnice = dalšíSouřadnice

        nalezni, kam se dá z tohoto bodu jit a jeste jsi tam nebyl
        když lze kam jit:
                Ulož si odkud jsi přišel do pole neighbor
        Jinak:
                Vrať se o políčko zpět

cesta[0] = souřadniceKonce
dalšíSouřadnice = souřadniceKonce

Prováděj pořád dokola, dokud nejsi ve startu:
        rodiče = neighbor[souřadniceKonce]
        cesta[x] = rodiče
        x++
        dalšíSouřadnice = rodiče

výpis cesty na kostku NXT

inicializuj gyroskopicky senzor
vypocitej offset gyroskopickeho senzoru

Dokavad nejsi v cili:
        Jezdi v bludisti podle definovaneho pole cesty
```

Řízení jízdy provádíme podle rozdílu současné polohy a polohy, které chceme dosáhnout v následujícím kroku. Podle tohoto rozdílu následně poznáme, jak se má robot natočit a následně se posunout o jedno políčko dopředu.

3.3.5.3. Programování v NXC

V této části jsem se rozhodl tento kód naprogramovat pouze v NXC, jelikož tento kód není vhodné psát v jazyce NXT-G, kde je velice komplikovaný a celý kód by se stal nepřehledný. Navíc se v jazyce NXT-G nedají zavádět pole, to samé platí pro kódy psané v programovacím prostředí PyNXC.

Program sdílí některé funkce s programem z 3.3.4.1.4, proto je zde nebudu uvádět.

3.3.5.3.1. Prohledávací část pro prozkoumání bludiště

Tento program prohledává bludiště. Do pole maze[] si uloží na prozkoumanou pozici pomocí 1, zda v bludišti našel na těchto souřadnicích volnou pojezdovou plochu. 0 pak znázorňuje oblast, která je neprozkoumaná, nebo se v ní nachází překážka.

```
74 void changePosition(){
75      if (orientation==0){ //pokud je orientation 0
76         positionX = positionX+1; //zmena pozice v ose x
77         positionY = positionY; //pozice y zustava
78      }else if (orientation==1){ //pokud je orientation 1
79         positionX = positionX; //pozice x zustava
80         positionY = positionY-1; //zmena pozice v ose y
81      }else if (orientation==2){ //pokud je orientation 2
82         positionX = positionX-1; //zmena pozice v ose x
83         positionY = positionY; //pozice y zustava
84      }else{ //pokud je orientation 3
85         positionX = positionX; //pozice x zustava
86         positionY = positionY+1; //zmena pozice v ose y
87      }
88      maze[5*positionX+positionY] = 1;
89 }
```

Změnila se funkce changePosition(), kdy se zde přidal příkaz, který uloží do pole maze[] 1, jako volnou pojezdovou plochu pro právě aktualizovanou souřadnici.

```
67 void mazeInit(){
68      for(int i = 0; i < 45; i++){ //projdi pole maze
69              maze[i] = 0; //vsude uloz nulu
70      }
71      maze[5*positionX+positionY] = 1; //na startovni pozici uloz 1
72 }
```

Funkce mazeInit() má za úkol připravit pole maze[] na zápis nově nalezených pojezdových ploch.

Následuje již jen hlavní část kódu, která je rozdělená na menší části a každá část, je okomentovaná zvlášť.

```
91 task main(){
92      SetSensorHTGyro(IN_2); //inicalizace gyroskopickeho senzoru
93      SetSensorLowspeed(IN_4); //inicalizace ultrazvukoveho senzoru
94
95      Wait(2000); //pocka se na vzadaleni ruky od robota
96      vypocti_offset(); //vypocita se offset
97
98      mazeInit(); //inicalizace pole pro zapis rozmisteni bludiste
99
100     int count = 0; //inicializace promenne pro pocitani poctu pruchodu smyckou
101     int move; //inicializace promenne
102     int free; //inicializace promenne
103     int distance; //inicializace promenne
```

Klasická inicializace senzorů a proměnných použitých dále v kódu.

```
104     while(count < 35){ //opakuj 35
105             move = 0; //nasatveni move na nulu
106
107             if(move == 0){ //pokud se robot nepohnul
108                 RotateMotor(OUT_B, 75, -90); //robot natoci ultrazvukovy senzor
109                 distance = SensorUS(IN_4); //nacteni hodnoty ze ultrazvukoveho senzoru
110                 RotateMotor(OUT_B, 75, 90); //robot vrati ultrazvukovy senzor do pivodni polohy
111             }
112
113             if(distance > 10 && move == 0){ //pokud se robot nepohnul a je vpravo volno
114                     turn90right(); //otoceni robota o 90 stupnu doprava
115                     RotateMotorEx(OUT_AC, 75, 584, 0,true,true); //otoci motory A a C dopredu o 584 stupnu
116                     move = 1; //robot se uz pohnul
117             }
118
119             if(move == 0){ //pokud se robot nepohnul
120                     distance = SensorUS(IN_4); //nacteni hodnoty ze ultrazvukoveho senzoru
121             }
122
123             if(distance > 10){
124                     RotateMotorEx(OUT_AC, 75, 584, 0,true,true); //otoci motory A a C dopredu o 584 stupnu
125                     move = 1; //robot se uz pohnul
126             }
127
128             if(move == 0){ //pokud se robot nepohnul
129                     RotateMotor(OUT_B, 75, 90); //robot natoci ultrazvukovy senzor
130                     distance = SensorUS(IN_4); //nacteni hodnoty ze ultrazvukoveho senzoru
131                     RotateMotor(OUT_B, 75, -90); //robot vrati ultrazvukovy senzor do puvodni polohy
132             }
```

```
134                 if(distance > 10 && move == 0){ //pokud se robot nepohnul a je vlevo volno
135                         turn90left();  //otoceni robota o 90 stupnu doleva
136                         RotateMotorEx(OUT_AC, 75, 584, 0,true,true); //otoci motory A a C dopredu o 584 stupnu
137                         move = 1; //robot se uz pohnul
138                 }
139
140
141                 if(move == 0){  //pokud se robot nepohnul
142                         turn180(); //otoceni robota doleva o 180 stupnu
143                         RotateMotorEx(OUT_AC, 75, 584, 0,true,true); //otoci motory A a C dopredu o 584 stupnu
144                         move = 1; //robot se uz pohnul
145                 }
146                 changePosition(); //zmena pozice robota a ulozeni nove nalezeneho pole do maze
147                 count++; //zviseni promenne count
148         } //konce smycky pro opakovani
```

Tato část kódu má za následek sledovaní pravé stěny. A zápis nově nalezených pojezdových ploch do pole maze[].

```
150     byte handle = 0; //inicializace promenne pro praci se souborem
151     DeleteFile("maze.txt"); //smazani stareho souboru maze.txt
152     CreateFile("maze.txt", 1024, handle); //vytvoreni noveho souboru maze.txt
153
154     for(int i = 0; i <45;i++){ //projdi cele pole
155             if(maze[i]==1){ //pokud je v poli 1
156                             Write(handle, "1"); //zapis 1
157             }else{  //pokud je v poli 0
158                             Write(handle, "0"); //zapis 0
159             }
160             if(i%5==4){ //pokud si napsal jeden radek
161                             Write(handle, "\n"); //napis znak pro enter
162             }
163     }
164     CloseFile(handle); //zavri soubor
165 }
```

Nakonec následuje zápis nově nalezeného pole do souboru maze.txt. DeleteFile() smaže příslušný soubor z paměti robota. CreateFIle() naopak soubor vytvoří a rovnou ho otevře pro zápis. CloseFile() pak zavírá soubor na konci běhu programu. Příkaz Write() zapisuje do otevřeného souboru. Proměnná handle pak udává, o jaký soubor se jedná a zda dochází ke správnému plnění jednotlivých příkazů.

3.3.5.3.2. Program jízdy bludištěm

Tento program nejprve načte rozložení bludiště z NXT kostky. Následně v bludišti naplánuje cestu od stratu do cíle dle algoritmu DFS. Následně pak řídí pohyb robota, tak aby robot dojel od startu do cíle. Jako v předchozím případě, zde budu popisovat pouze neznámé funkce, které nebyly popsané v kapitolách 3.3.4.1.4 nebo 3.3.5.3.1.

Jak je již zvykem v této kapitole, jsem se zde opět rozhodl použít globální proměnné.

```
1 char load[120];
2 int maze[45];
3
4 int nextX;
5 int nextY;
6
7 int neighborX[45];
8 int neighborY[45];
9
10 int pathX[45];
11 int pathY[45];
12
13 int visitedX[45];
14 int visitedY[45];
15
16 int visitedcount = 0;
17
18 int orientation=1;
19
20 int offset;
```

Pole load[] slouží pro načtení hodnot ze souboru. Pole maze[] ukládá podobu bludiště ve formátu přijatelném pro NXC. Proměnné nextX a nextY slouží pro ukládání souřadnic následujících políček. Pole neighborX[] a neighborY[] slouží k ukládání předchůdců čtverečků. Pole pathX[] a pathY[] slouží pro ukládání nalezené cesty. Pole visitedX[] a visitedY[] slouží pro ukládání již navštívených čtverečků. Proměnná visitedcount slouží pro ukládání počtu navštívených polí. Proměnná orientation pak určuje orientaci robota dle Obr. 34 a offset slouží pro uložení offsetu pro potřeby měření úhlu pomocí gyroskopického senzoru. Celý kód by šel zjednodušit, kdyby se v NXC dali inicializovat 2D pole. Bohužel 2D pole jazyk NXC neumí, a proto bylo třeba pracovat s 1D poli.

```
26 void loadMaze(){
27      byte handle = 1; //inicializace promenne
28      int fsize; //inicializace promenne
29      OpenFileRead("maze.txt", fsize, handle); //otevreni souboru maze.txt
30      Read(handle, load); //nacteni dat ze souboru
31
32      int count = 0; //inicializace promenne
33
34      for(int i = 0; i < 120; i++){ //projeti nachteneho pole
35              if(load[i] == 49){ //pokud je v nactem poli "1"
36                      maze[count] = 1; //pridani do pole maze volna pojezdova plocha
37                      count++; //zvyseni promenne count
38              }else if(load[i] == 48){ //pokud je v nactem poli "0"
39                      maze[count] = 0; //pridani do pole maze prekazka
40                      count++; //zvyseni promenne count
41              }
42      }
43      CloseFile(handle); //zavreni souboru maze.txt
44 }
```

Funkce loadMaze() načte ze souboru maze.txt podobu bludiště a následně ji uloží do pole maze[]. Read() načte hodnotu jako string, my ji tedy musíme převést do číselné podoby, o což se stará část kódu mezi řádky 34-42.

```
46 int visit(int posiceX, int posiceY){
47     for(int i = 0; i < visitedcount; i++){ //projde vsechny polozky v poli visitedX a visitedY
48             if(posiceX == visitedX[i] && posiceY == visitedY[i]){//pokud se vstupni pozice nachazi v poli
49                     return 1; //vrat 1
50             }
51     }
52     return 0; //vrat 0
53 }
```

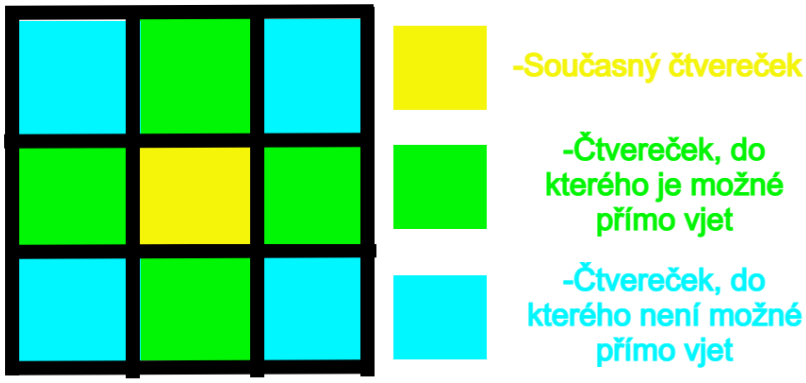
Funkce visit(souřadniceX,souřadniceY) se podívá do pole visited[] a pokud jsme už na souřadnicích souřadniceX a souřadniceY byli vrátí nám tato funkce 1. V opačném případě dostaneme jako výstup této funkce 0.

```
77 int desision(int endX, int nextX, int endY, int nextY){
78     if(endX == nextX){ //pokud se vstupy rovnaji
79             if(endY == nextY){ //pokud se vstupy rovnaji
80                     return 0; //vrat 0
81             }
82     }
83     return 1; //vrat 1
```

Funkce desision(souřadnice1X, souřadnice2X, souřadnice1Y, souřadnice2Y) vrací 0 pokud se souřadnice1X rovná souřadnici2X a souřadnice1Y se rovná souřadnici2Y. V opačném případě vrací 1.

```
52 void findNeighbor(int startX, int startY){
53     int poseX[] = {1, -1, 0, 0}; //rozhlidnuti do vsech smeru
54     int poseY[] = {0, 0, 1, -1}; //rozhlidnuti do vsech smeru
55
56     nextX = -1; //nastaveni nextX na nenalezene
57     nextY = -1; //nastaveni nextY na nenalezene
58
59     for(int i = 0; i < 4; i++){ //projiti vsech poli v okoli
60             int posiceX = startX + poseX[i]; //postupne rozhlidnuti do vsech smeru
61             int posiceY = startY + poseY[i]; //postupne rozhlidnuti do vsech smeru
62             if(posiceX < 9 && posiceX > -1 && posiceY < 5 && posiceY > -1){ //pokud se nachazime v bludisti
63                 if(maze[5*posiceX+posiceY] == 1){ //pokud je v bludisti pojezdova plocha
64                                         int visited = visit(posiceX, posiceY); //podivame se zda uz jsme toto pole nanvstivili
65                                         if(visited == 0){ //podivame se zda uz jsme toto pole nanvstivili
66                                                 nextX = posiceX; //nastavim next na dalsi mozne policko v ceste
67                                                 nextY = posiceY; //nastavim next na dalsi mozne policko v ceste
68                                                 visitedX[visitedcount] = posiceX; //pridam toto policko mezi navstivene
69                                                 visitedY[visitedcount] = posiceY; //pridam toto policko mezi navstivene
70                                                 visitedcount++; //zvisim pocet navstivenych
71                                         }
72                 }
73             }
74     }
75 }
```

Funkce findNeighbor(souřadniceX, souřadniceY) se podívá zda se v okolí souřadniceX a souřadniceY nachází volné políčka na které by mohl robot vjet a zároveň není ještě prohledané. Ilustrace, na které políčko se díváme je na Obr. 40.



Obr. 40 Čtverečky, na které může robot vjet

```
118 void turn(int need){
119     if(need < orientation || (need == 3 && orientation == 0)){
120             turn90left();
121     }else{
122             turn90right();
123     }
```

Funkce turn(potřebná) pootočí s robotem, tak aby jeho orientace odpovídala parametru potřebná.

```
138 void preparArray(){
139     for(int i = 0; i < 120; i++){ //projde cele pole pathX, ´pathY, visitedX, visitedY
140             pathX[i] = -1; //inicializace pole
141             pathY[i] = -1; //inicializace pole
142             visitedX[i] = -10; //inicializace pole
143             visitedY[i] = -10; //inicializace pole
144     }
145 }
```

Funkce prepareArray() připraví pole pathX[], pathY[], visitedX[] a visitedY[].

```
147 void turnAndMove(int directX, int directY){
148                 if(directX == 1){ //pozadovany smer v ose X 1
149                        while(orientation != 2){ //dokavad se nenchazim v pozadovane orientaci
150                                        turn(2); //otoceni do orientace 2
151                        }
152
153              }else if(directX == -1){ //pozadovany smer v ose X -1
154                    while(orientation != 0){ //dokavad se nenchazim v pozadovane orientaci
155                                   turn(0); //otoceni do orientace 1
156                    }
157
158              }else if(directY == 1){ //pozadovany smer v ose Y 1
159                    while(orientation != 1){ //dokavad se nenchazim v pozadovane orientaci
160                                   turn(1); //otoceni do orientace 1
161                    }
162              }else{ //pozadovany smer v ose Y -1
163                    while(orientation != 3){ //dokavad se nenchazim v pozadovane orientaci
164                                   turn(3); //otoceni do orientace 3
165                    }
166              }
167
168              RotateMotorEx(OUT_AC, 75, 584, 0,true,true); //posunuti robota o pole vpred
169 }
```

Funkce turnAndMove() pootočí robota do příslušné orientace podle zadaného směru a zároveň s ním přejede o políčko dopředu. Orientace robota je znázorněná na Obr. 34.

Hlavní funkce programu bude rozdělena na jednotlivé dílčí části, které zde budou popsány.

```
172 task main(){
173      loadMaze(); //nacteni bludiste
174      preparArray(); //priprava poli pro DFS
175
176      int startX = 8; //startovni souradnice X
177      int startY = 0; //startovni souradnice Y
178
179      int endX = 8; //cilova souradnice X
180      int endY = 5; //cilova souradnice Y
181
182      orientation=1 //orientace robota
```

Inicializace proměnných a nastavení souřadnic startu a cíle společně s nastavením orientace robota podle Obr. 30.

```
184      nextX = startX; //prvni krok v DFS
185      nextY = startY; //prvni krok v DFS
186
187      int oldX; //inicializace promenne
188      int oldY; //inicializace promenne
189
190      visitedX[visitedcount] = startX; //prvni nastivene
191      visitedY[visitedcount] = startY; //prvni nastivene
192
193      visitedcount++; //navstivenych uz je 1
194
195      while(desision(endX, nextX, endY, nextY)){ //opakuj dokavad nejsi v cili
196                   oldX = nextX; //soucasna pozice X
197                   oldY = nextY; //soucasna pozice Y
198
199                   findNeighbor(nextX, nextY); //nalezeni nasledujici pozice
200
201                   if(nextX != -1 && nextY != -1){ //pokud se nekam jit
202                         neighborX[5*nextX+nextY] = oldX; //ulozeni pozice X predchudce
203                         neighborY[5*nextX+nextY] = oldY; //ulozeni pozice X predchudce
204                   }else{ //jinak
205                         nextX = neighborX[5*nextX+nextY]; //vrat se o pole zpet
206                         nextY = neighborY[5*nextX+nextY]; //vrat se o pole zpet
207                   }
208      } //konec smycky
```

Nalezení stromu cesty dle algoritmu DFS. Tedy nehledáme ještě samostatnou cestu, ale provádíme rozvoj podle začátku podkapitoly 3.3.5.

```
210     nextX = endX; //dalsi je krok je cil
211     nextY = endY; //dalsi je krok je cil
212
213     int j = 1; //inicializace promenne
214     pathX[0] = endX; //prvni krok cesty je konec
215     pathY[0] = endY; //prvni krok cesty je konec
216
217     int parentX; //inicializace promenne
218     int parentY; //inicializace promenne
219     int pathcount = 0; //inicializace promenne
220
221     while(desision(startX, nextX, startY, nextY)){ //opakuj dokavad nejsi v
222                 parentX = neighborX[nextX*5+nextY]; //nalezeni predchudce
223                 parentY = neighborY[nextX*5+nextY]; //nalezeni predchudce
224                 pathX[j] = parentX; //ulozeni prdechudce do cesty
225                 pathY[j] = parentY; //ulozeni prdechudce do cesty
226                 j++; //cesta se prodlouzila
227                 nextX = parentX; //posunuti se o krok zpet
228                 nextY = parentY; //posunuti se o krok zpet
229
230                 pathcount++; //cesta je o jedno delsi
231     } //konec smycky
```

Nalezení cesty ze stromu cesty, který jsme vytvořili v předchozím bloku kódu. Základem je, že jdeme od konce a pomocí souřadnic předchozích polí se snažíme dostat do startu.

```
240
241     pathcount--; //zkraceni delky cesty
242
243     nextX = startX; //zaciname na startu
244     nextY = startY; //zaciname na startu
245
246     SetSensorHTGyro(IN_2); //inicializace gyroskopickeho senzoru
247     vypocti_offset(); //vypocet offsetu
248
249     int directX; //smer v ose X
250     int directY; //smer v ose Y
251
252     while(pathcount > -1){ //opakuj dokavad nejsi v cily
253             directX = pathX[pathcount] - nextX ; //vypocet smeru v ose X
254             directY = pathY[pathcount] - nextY; //vypocet smeru v ose Y
255
256             turnAndMove(); //pohnu se ve spravnem smeru o pole dopredu
257
258             nextX = pathX[pathcount]; //aktualizce polohy robota
259             nextY = pathY[pathcount]; //aktualizce polohy robota
260             pathcount--; //poposel jsem o jedno pole dopredu
261     } //konec smycky
262 }
```

Následně se podle nalezené cesty rozhodujeme, jak s robotem otočíme, abychom se mohli posunout do dalšího políčka. Toto provádíme, tak dlouho dokud s robotem nedojedeme do cíle.

3.3.5.3.3. Prohledávání bludiště DFS v programu PyNXC

V programovacím jazyku PyNXC nelze vytvářet pole. Tudíž se mi nepodařilo program na prohledávání bludiště přepsat do tohoto jazyka. Hodnotím tedy tento jazyk pro tuto konkrétní úlohu nevhodný. Obecně nemožnost inicializovaní polí v tomto jazyku je jeho největší nevýhoda a znemožňuje použití tohoto jazyka pro složitější programy, které předpokládají použití polí.

3.3.6. Závěr

V tomto úkolu, byly představené dva programy na průjezd bludištěm. Jako první byl představený program, který sleduje stěny bludiště a snaží se tak podél této stěny dostat do cíle. Druhý program je pak rozdělený do dvou částí. První část si prohledá bludiště a druhá část pak najde v takto prohledaném bludišti nejrychlejší trasu od začátku do cíle. V případě nutnosti není potřeba spouštět první část programu a mapu bludiště lze explicitně zapsat do paměti robota.

Jako nevětší problém se při plnění tohoto úkolu ukázala potřeba přesného zatáčení, které jsem se snažil vyřešit pomocí gyroskopického senzoru. Jelikož druhá možnost zatáčení pomocí spuštění motorů v protichůdném režimu a čekání po určitou dobu, než se robot otočí, se ukázala značně nevhodná, jelikož rychlost otáčení motorů je závislá na napětí baterie. Toto napětí však klesá s tím, jak se baterie vybíjí a tím se snižuje i rychlost motorů a mění se úhel, o který se robot za určitý čas otočí.

Druhý problém nastal v případě přímé jízdy robota dopředu o předem stanovenou dráhu. Pohyb o stanovenou dráhu jsem vyřešil pomocí otáčení hřídele servomotorů o stanovený úhel, který je spočítaný v rovnici (5). Synchronizaci chodu servomotorů za mě nakonec vyřešil příkaz RotateMotorEx(). Aby robot jel zcela rovně je ještě možné v tomto příkazu nastavit argument **turn** na jinou hodnotu než na 0 a tímto korigovat pohyb robota.

3.4. Úkol 3: Sledování čáry pomocí regulátorů

V tomto úkolu se podíváme na různé typy regulátorů. Pomocí těchto regulátorů budeme sledovat černou čáru.

3.4.1. Zadání

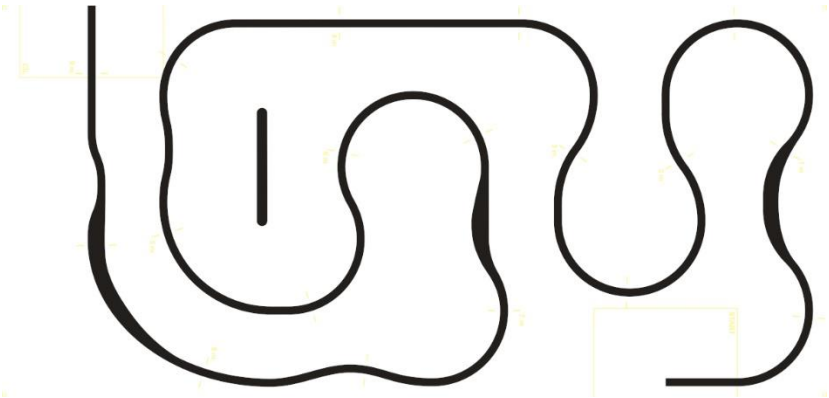
Cílem tohoto úkolu je naprogramovat robota tak, aby samostatně a co možná nejrychleji, projel vyznačenou dráhu od startu do cíle. Mezi startem a cílem je na podložce vyznačena černá čára, která znázorňuje trajektorii, podle které se bude robot pohybovat. Během plnění úkolu je nepřípustné, aby robot jakýmkoliv způsobem komunikoval se studenty. Zároveň je nepřípustné, aby robot sjel z plochy, na které je vyznačená dráha.

3.4.1.1. Popis soutěžní plochy

Plocha pro soutěž má celkový rozměr 250 x 150 cm a je tvořena bílou laminátovou deskou. Na ní je umístěn bílý papír s vyznačenou černou čárou, která vyznačuje soutěžní dráhu od startovního prostoru do cíle.

Délka dráhy se může pohybovat v libovolném rozmezí od 1 m po 20 m. Černá čára může mít rozdílnou tloušťku, která se má pohybovat v rozmezí od 1 cm do 5 cm. Dráha může mít libovolný tvar, avšak s následujícími omezeními. Poloměr zatáčky může být maximálně 10 cm, čáry se nemohou křížit a vzájemná vzdálenost dvou sousedních čar je minimálně 10 cm. Start a cíl je na soutěžní ploše vyznačen čtvercem o rozměrech 40 x 40 cm se žlutými okraji. Za startovní a cílovou čáru se pak považuje strana příslušného čtverce, která se protíná s černou čárou neboli trajektorií robota. Stěny čtverců, které vyznačují startovní a cílový prostor, se kříží s černou linkou právě jednou.

Ukázka možného uspořádání dráhy je vidět na obrázku Obr. 41. Uspořádání se však může bez jakéhokoli varování změnit.



Obr. 41 Možné uspořádání soutěžní dráhy

Budou připraveny minimálně dvě uspořádání dráhy. Jedno uspořádání dráhy bude pro sestavení a testování robota. Druhé uspořádání dráhy bude připraveno pro finální soutěž.

3.4.1.2. Finální hodnocení

Ve finálním hodnocení robotů se bude hodnotit čas jízdy robotů mezi startem a cílem. Sjetí z černé čáry je důvodem k diskvalifikaci. Za sjetí z černé linie se považuje situace, kdy se robot bude pohybovat více, než 15 cm od této linie. Rovněž se za sjetí považuje situace, kdy si robot významně zkrátí soutěžní dráhu. Roboti, kteří neprojedou celou dráhu, budou hodnoceni na základě ujeté vzdálenosti od startu. Roboti s úspěšným projetím dráhy budou hodnoceni podle času průjezdu mezi startem a cílem. Roboti, kteří projeli celou dráhu, budou hodnoceni vždy lépe než roboti, kteří z dráhy sjeli.

Při pořádání soutěže v rychlosti jízdy robotů nebo ve finálním hodnocení mohou být použity světelné závory k určení doby jízdy robota po vyznačené dráze. Tyto závory budou umístěny na krajích startovní a cílové čáry a nebudou jakýmkoliv způsobem bránit robotovi v průjezdu.

3.4.2. Rozbor problému

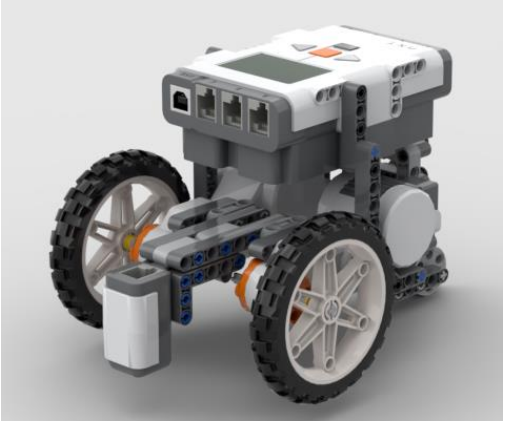
Naším úkolem je projet soutěžní dráhou, která je vyznačena černou barvou. Budeme tedy potřebovat dva servomotory, abychom se po této dráze mohli pohybovat a zároveň mohli s robotem jezdit do všech směrů.

Další problém, který musíme vyřešit, je jízda po černé čáře nebo jinak řečeno sledování černé čáry. Toto sledování provedeme pomocí světelného senzoru. Z prvního úkolu již víme, že při měření odraženého světla lze pomocí tohoto senzoru poměrně jednoduše odlišit černou a bílou barvu. Tyto dvě barvy jsou dominantní barvy na podložce. Místy se k těmto barvám přidává ještě žlutá barva. Hodnota žluté barvy na tomto senzoru však není příliš rozdílná oproti bílé barvě.

Dále si musíme zvolit, jak si bude robot určovat, jakou trasou pojede. Tento úkol budeme řešit pomocí regulátorů. Regulátor nám udává to, jakým způsobem se robot po soutěžní ploše pohybuje. Z nepřeberného množství regulátorů se společně podíváme na cik-cak a regulátor PID/PD. Jelikož se v setech stavebnic 9797, 9648 a 9695 nachází celkem jeden senzor světla, zaměříme se na regulátory jednosenzorové. Kdybychom měli k dispozici více senzorů světla, mohli bychom vytvořit i regulátory dvousenzorové případně i vícesenzorové. Výhodou těchto vícesenzorových regulátorů je zpravidla jejich větší robustnost. Naopak nevýhodu tohoto typu regulátorů je složitost jejich naprogramování.

3.4.3. Konstrukce

Robot je konstruovaný tak, aby mohl jezdit po soutěžní ploše a sledovat černou čáru. Aby se mohl pohybovat libovolným směrem, je vybavený dvěma servomotory. Kvůli sledování čáry je vybavený světelným senzorem. Při jeho konstrukci byla použita velká 82 mm kola. Motory jsou umístěny pod samotnou kostkou NXT. Světelný senzor je umístěný před motory a konstrukce je doplněna vlečným kolečkem. Ke stavbě robota vznikl návod, který je dostupný v digitální příloze této práce.



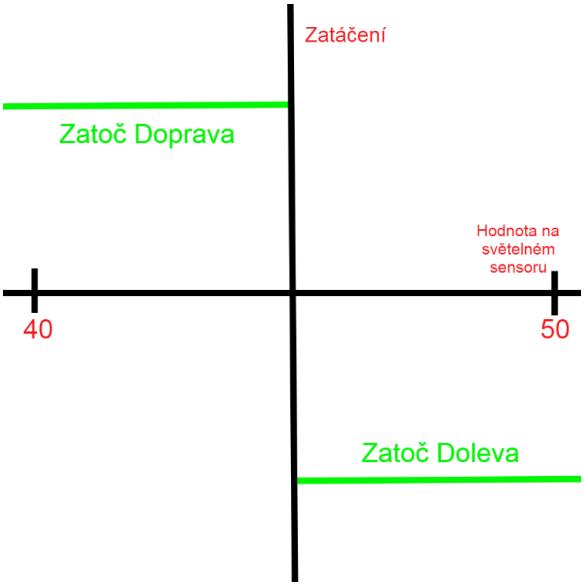
Obr. 42 Sledovač čáry pohled I.

Obr. 43 Sledovač čáry pohled II.

Pravý motor sledovače čáry je připojený k portu C na NXT kostce. Levý motor je pak připojený k portu A. Světelný senzor sloužící k nalezení černé čáry je připojený na port 3.

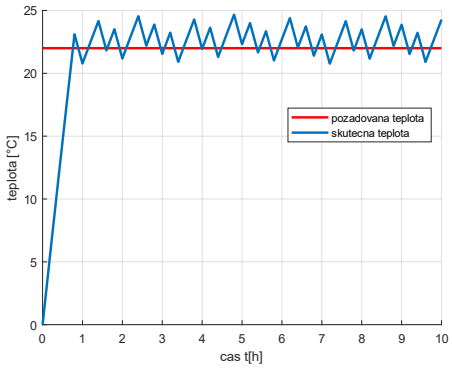
3.4.4. Zig-Zag

Toto je nejednoduší typ regulátoru. Spadá do skupiny regulátorů Bang-Bang. Vlastností těchto regulátorů je, že u nich existují jen dva stavy, spuštěno a nespuštěno. A proto tedy pracují nespojitě, jak je vidět na Obr. 44, kde je ukázán takový regulátor, který by mohl sledovat černou čáru, na základě hodnot naměřených na senzoru světla.



Obr. 44 Zig Zag regulátor

Pomocí tohoto typu regulátoru se nedá docílit přesné regulace pohybu robota a robot tudíž bude vykonávat zbytečné pohyby. Jak tento regulátor funguje, si vysvětlíme na příkladu vytápění. Graf teploty v jednom pokoji v domě je znázorněn na Obr. 45.



Obr. 45 Graf teploty

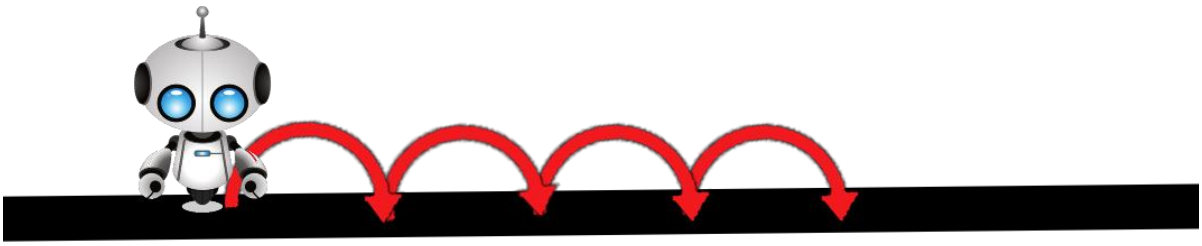
Kotel v tomto domě je starý a funguje u něj regulace pouze zapnuto/vypnuto. Na začátku pustíme kotel a čekáme, až se místnost vyhřeje. Pro nás je příjemná teplota 23 °C, počkáme tedy, než se teplota dosáhne 23,5 °C, následně kotel vypneme a počkáme, než se teplota v místnosti ochladí na 22,5 °C. Poté kotel opět pustíme a čekáme na teplotu 23,5 °C. A opět kotel vypínáme. Jelikož jsme líní kotel zapínat a vypínat ručně, nastavíme si počítač, aby tento úkol dělal za nás. Pseudokód takovéhoto programu by vypadal následovně.

```
Opakuj dokola:
    Zjisti teplotu v místnosti
    Když teplota je nižší než 22,5 °C:
        Zapni kotel
    Když teplota je vyšší než 23,5 °C:
        Vypni kotel
```

Jak je vidět na Obr. 45 teplota v místnosti pořád kolísá mezi 24 °C a 22 °C. To je rozdíl 2 stupně mezi nejvyšší a nejnižší teplotou. Prakticky nelze teplotu 23 °C použít na současné vypínání a zapínání kotle, vždy zde musí dojít k určité prodlevě, aby byl jasně stanoven příkaz zapnutí nebo vypnutí. Při nedodržení této prodlevy by se kotel dostal do stavu zapnuto-vypnuto, což by způsobilo poruchu kotle.

Pro účely sledování čáry si lehce upravíme Bang-Bang regulátor, čímž dostaneme typ regulátoru, který se běžně označuje jako Zig-Zag nebo také cik-cak. Bude fungovat tak, že na robotovi bude spouštět jen pravý nebo jen levý motor. Jak se budou jednotlivé motory spouštět, bude dáno tím, jakou hodnotu odraženého světla naměříme pomocí světelného senzoru. Nejprve si musíme vybrat, na jaké straně černé čáry chceme jezdit. V tomto úkolu se počítá s tím, že se černá čára sleduje z leva. Když najede robot na černou čáru, uvidíme na světelném senzoru nízkou hodnotu odraženého světla. V takovém okamžiku robot spustí pouze pravý motor. Když na senzoru naměříme naopak vysokou hodnotu odraženého světla, zapíná se pouze levý motor robota. Tím, jak robot jezdí jen jedním kolem, vykonává zatáčky, tak jak je znázorněno na Obr. 46. Při těchto zatáčkách se světelný senzor pohybuje po hrací ploše a měří buď nízké hodnoty odraženého světla, je-li nad černou čárou, nebo vysoké hodnoty nad bílou okolní plochou. Podle naměřených hodnot se pak spouští jednotlivé motory. Když se tento postup

spouštění a vypínaná motoru opakuje, tak se robot pomocí těchto malých zatáček posouvá pomalu dopředu.



Obr. 46 Pohyb robota při zig zag regulátoru

3.4.5. Programování Zig-Zag regulátoru

Tyto kódy nebudu podrobně komentovat, jelikož jsou jednoduché a krátké. Všechny kódy jsou upravené na to, aby robot sledoval černou čáru z leva. K tomu, aby tuto čáru sledoval robot zprava je potřeba přehodit hodnoty podmínek.

Obecné doporučení. Nedoporučuje se motory pouštět na maximální výkon, jelikož dochází k jejich nadměrnému opotřebovávání.

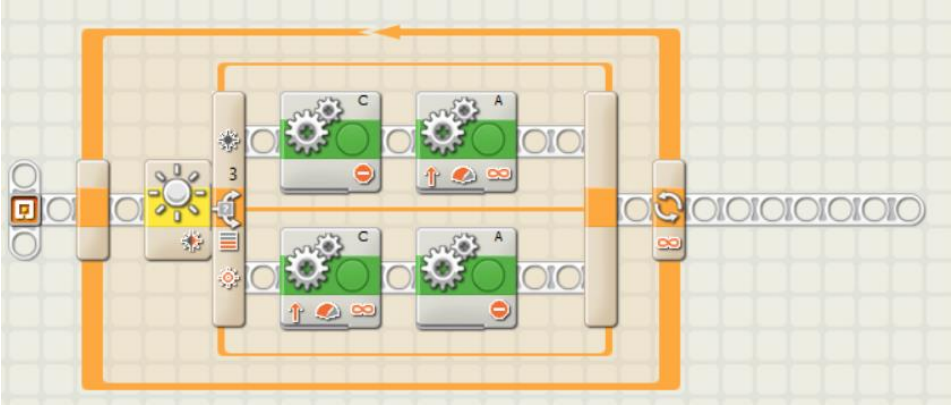
3.4.5.1. Pseudokód

```
Zig-Zag
Zapnutí programu
Prováděj pořád do kola:
        Odečti honotu ze senzoru světla a ulož ji do proměné světlo
        Když svetlo < 50:
                Spusti levý motor
                Vypni pravý motor
        Jinak:
                Spusti pravý motor
                Vypni levý motor
```

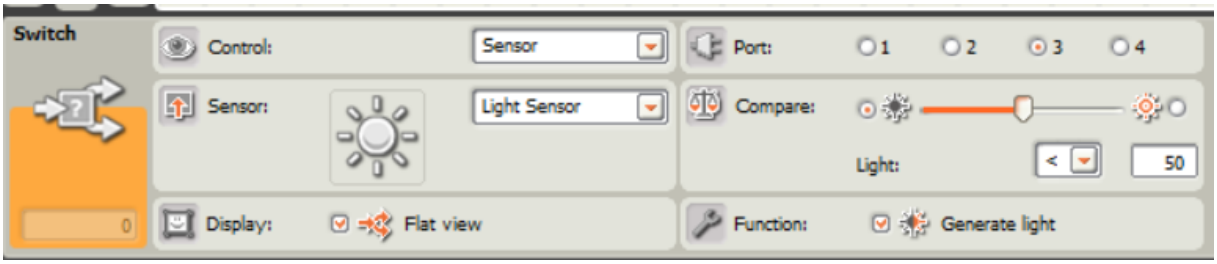
Úkoly se provádí postupně v nekonečném cyklu. Nejprve se odečte hodnota ze světelného senzoru. Podle této hodnoty se rozhoduje, co robot udělá. Když bude na senzoru nízká hodnota, bude se robot nacházet nad černou čárou a spustí levý motor a pravý motor naopak vypne. Tím se robot natočí směrem od černé čáry a zároveň se posune kousek dopředu. Opakováním tohoto úkolu se světelný senzor robota postupně dostane až nad bílou část plochy a bude vysílat do NXT kostky vysokou hodnotu odraženého světla. V takovém případě se neaktivuje první podmínka na „když“ if, ale aktivuje se podmínka „jinak“ else. V takovém případě robot spustí pouze pravý motor, a levý naopak vypne. Tímto způsobem robot bude zatáčet doleva, čímž se opět dostane nad černou čáru, přičemž se o malý kousek posune dopředu.

Jak robot vykonává tento kód, pomalu se pohybuje dopředu. A přitom se pohybuje pouze po černé čáře. Konstanta 50 u podmínky je zde, protože hrana černé čáry má hodnotu přibližně 50. Tato skutečnost je podrobněji rozebrána v kapitole 3.4.9.

3.4.5.2. NXT-G



Nastavení bloku **Switch**.



Blok **Switch** je nastavený tak, aby při zaznamenání nízké hodnoty odraženého světla na světelném senzoru, aktivoval horní větev kódu. V opačném případě aktivuje spodní větev kódu. V horní větvi je nastaveno pomocí dvou bloků **Motor**, aby se aktivoval jen pravý motor a levý se naopak vypnul. Ve spodní větvi kódu je to právě naopak, zapíná se levý motor a vypíná se pravý.

3.4.5.3. NXC

```
1  int light;
2
3  task main(){
4        SetSensorLight(IN_3);
5        while(true){
6                    light = Sensor(IN_3);
7                    if(light < 50){
8                              OnFwd(OUT_A, 80);
9                              OnFwd(OUT_C, 0);
10                   }else{
11                             OnFwd(OUT_C, 80);
12                             OnFwd(OUT_A, 0);
13                   }
14       }
15 }
```

V kódu probíhá nekonečná smyčka. Na začátku průběhu každé smyčky se odečte hodnota odraženého světla ze světelného senzoru. Pokud je odražená hodnota menší jak 50 spouští se pouze pravý motor a levý se vypíná. Pokud je odčítaná hodnota naopak větší, aktivuje se podmínka else. Díky této podmínce se zapne pouze levý motor a pravý se vypíná. Jak se zapínají motory, robot se v zatáčkách pohybuje pomalu dopřede podél černé čáry.

3.4.5.4. PyNXC

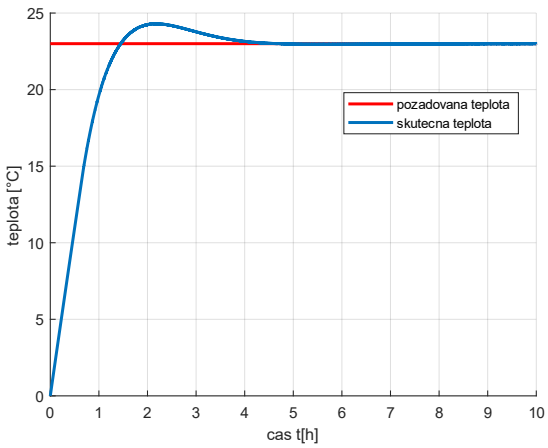
Následující kód vznikl jako přepis kódu NXC z předchozí kapitoly do PyNXC.

```
def main():
    SetSensorLight(IN_3)
    while (true):
        light = Sensor(IN_3)
        if light < 50:
            OnFwd(OUT_A, 80)
            OnFwd(OUT_C, 0)
        else:
            OnFwd(OUT_C, 80)
            OnFwd(OUT_A, 0)
```

Kód se podařilo úspěšně přeložit a spustit na NXT kostce. Po spuštění kód vykonával jízdu podél černé čáry za použití cik-cak regulátoru. Proto hodnotím programovací prostředí PyNXC jako vhodné pro tento úkol.

3.4.6. Regulátory PID/PD

Jak jistě uznáte pohyb robota s naprogramovaným Zig-Zag regulátorem není ideální. Robot se posouvá dopředu jen velice pomalu. Rovněž chování regulátorů typu bang bang není ideální, jelikož u nich existují pouze dva stavy zapnuto a vypnuto. Tato skutečnost nám může vadit u řady průmyslových procesů, které jsou náročné na přesnost. My asi jen stěží poznáme výkyvy teploty způsobené spouštěním topení z předchozí úlohy. Nicméně, kdyby se obdobně ovládaly pece pro tištěné spoje, mohlo by dojít k nenávratnému poškození výrobků, proto potřebujeme lepší formy regulace. Jeden z nejlepších regulátorů je PID regulátor. PID regulátor pracuje na principu regulace výkonu. Tedy není již dvoustavový. Příklad regulace teploty pomocí PID regulátoru a kotle, který dokáže měnit svůj výkon je vidět na Obr. 47.

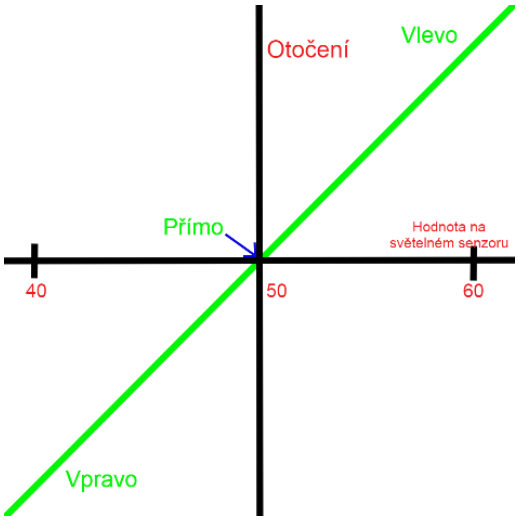


Obr. 47 Příklad regulace teploty pomocí PID

Jak je vidět z Obr. 47 PID na začátku teplotu nedodrží a ohřeje místnost na větší teplotu, než požadujeme. Nicméně po této drobné nedokonalosti již teplotu drží na stabilní úrovni.

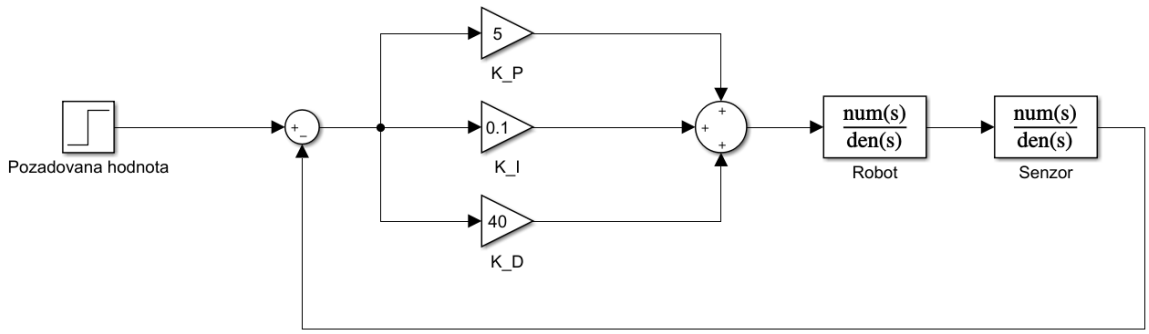
My použijeme PID regulátor na regulaci pohybu při sledování černé čáry. Skutečná jeho výhoda totiž tkví v tom, že tento regulátor je zpětnovazební a dokáže tedy regulovat odchylku od naměřené hodnoty a tu udržovat na stabilní úrovni. Když se podíváme na černou čáru skrze světelný senzor, naměříme na ní hodnotu přibližně 30, naopak na bílé ploše hodnotu 70. My se budeme snažit pohybovat po okraji černé čáry. Pokud se na kraji této čáry pokusíme změřit hodnotu odraženého světla, naměříme hodnotu 50. A právě po této hodnotě se bude snažit jezdit. Přesněji se ji náš regulátor bude snažit najít a sledovat.

V následujícím příkladu předpokládám, že se černá čára sleduje z leva. Když na senzoru robot uvidí větší hodnotu než 50, zatočí doleva. Když naopak na senzoru robot uvidí menší hodnotu než 50, bude robot zatáčet doprava. Když robot bude mít štěstí a na senzoru uvidí hodnotu 50, pojede přímo rovně. Ve své podstatě tento regulátor dělá to samé, co dělal už cik-cak regulátor, ale jeho výhodou je, že si sám reguluje, jak moc má robot zatáčet. Naměří-li se na senzoru hodnota 70, robot zatočí jinak, než kdyby se na senzoru naměřila jen hodnota 60. Tento regulátor tedy není již nespojitý jako cik-cak, ale jedná se o spojitý regulátor. Graf Obr. 48 znázorňuje, jak funguje PID regulátor a jak reguluje zatáčení robota.



Obr. 48 PID regulátor

Zkratka PID označuje následující P – za proporcionální část, I – za integrační část a D – za derivační část. Blokové schéma, jak vypadá uspořádání PID regulátoru, nalezneme na Obr. 49.



Obr. 49 blokové schéma PID regulátoru

Jak je vidět na Obr. 49, PID regulátor se skládá z několika částí, a to proporcionální části, integrační části, derivační části a zápornou zpětnou vazbu ze senzoru na vstup regulátoru. Pomocí výstupu PID regulátoru se reguluje pohyb motorů, jehož výsledkem jsou nově naměřené hodnoty ze senzoru světa. Dále je zde tzv. *target value* (Pozadovana hodnota), tedy hodnota, na které bychom rádi robota ustálili. Následuje popis jednotlivých částí regulátoru.

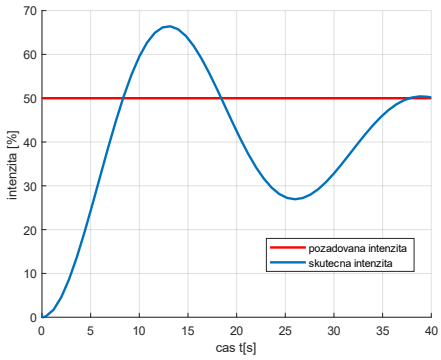
3.4.6.1. Proporcionální část

Pro nás je to asi nejzajímavější část PID regulátoru. Existují i samostatné P regulátory. P regulátor se podívá na aktuální naměřenou hodnotu na světelném senzoru. Tuto hodnotu odečte od *target value* a po vynásobení proporcionální konstantou dostane velikost akčního zásahu. Akční zásah nám říká, jak se mají motory otáčet. Rovnice takovéhoto regulátoru je v (6).

$$U = K_P \cdot e_r \tag{6}$$

U označuje akční zásah, K_P je takzvaná proporcionální konstanta a e_r určuje, o kolik se liší hodnota právě naměřená hodnota od *target value*. Proměnné e_r se také někdy říká regulační odchylka.

Tento typ regulátoru tedy vlastně jen zesiluje regulační odchylku a podle toho ovládá motory robota. Funguje tedy pouze jako zesilovač. Nevýhodu tohoto regulátoru je, že nikdy nedosáhneme nulové regulační odchylky. Průběh jízdy takového robota můžeme vidět na Obr. 50.



Obr. 50 Proporcionální regulátor a jeho časový vývoj v okolí target value

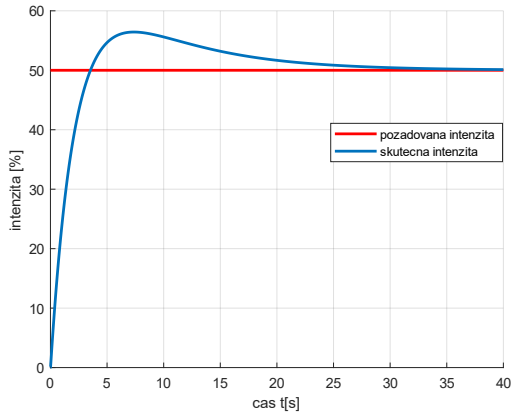
3.4.6.2. Derivační složka.

Na rozdíl od P složky samostatný D regulátor neexistuje. Není to z důvodu, že by nešel zkonstruovat, ale takový regulátor z hlediska jeho vlastností nedává smysl. Proto zde budu mluvit pouze o derivační složce. Derivační složka se dívá na rozdíl mezi současnou a minulou regulační odchylkou. Tuto odchylku přenásobuje derivační konstantou. Derivační složka je vlastně takový věštec a snaží se určit, jak se bude do budoucna vyvíjet regulační odchylka, tedy jak bude vypadat trasa robota a snaží se ji již s předstihem upravovat akční zásah. Rovnice takovéhoto regulátoru je popsaná v (7).

$$U = K_D \cdot (e_r - e_m) \tag{7}$$

U označuje akční zásah, K_D je takzvaná derivační konstanta a e_r je aktuální regulační odchylka, e_m je pak regulační odchylka z minulého cyklu běhu regulátoru. $(e_r - e_m)$ budu dále v textu označovat jako *derivative*.

K P regulátoru se často přidává D složka a vzniká PD regulátor. Takový PD regulátor má oproti P regulátoru výhodu, v tom že se rychleji přiblíží k požadované hodnotě, tzv. *target value*. Průběh chování tohoto regulátoru je vidět na Obr. 51.



Obr. 51 PD regulátor a jeho časový vývoj v okolí target value

Rovnice akčního zásahu PD regulátoru je vidět v (8).

$$U = K_P \cdot e_r + K_D \cdot (e_r - e_m) \tag{8}$$

U označuje akční zásah, K_P je takzvaná proporcionální konstanta a K_D je takzvaná derivační konstanta. e_r je aktuální regulační odchylka. e_m je pak regulační odchylka z minulého cyklu běhu regulátoru.

3.4.6.3. Integrální složka

Integrální regulátor sice existuje, ale jeho uplatnění se najde jen ve velice specifických podmínkách. Proto si zde integrační složku popíšeme obdobně jako derivační. Integrační složka se dívá do minulosti. A na základě tohoto minulého odhadu zkouší určit, jak se bude dráha vyvíjet dál. Její funkce je taková, že se dívá na všechny hodnoty regulační odchylky, které si sčítá do proměnné *integral*. Podle této proměnné se pak určuje akční zásah jejím přenásobení integrační konstantou. Rovnice integrální složky je vidět v rovnici (9).

$$U = K_I \cdot integral \tag{9}$$

K_I je integrační konstanta. *Integral* je pak suma všech regulačních odchylek od začátku běhu programu.

Velikost akčních zásahů budu označovat podle jejich původu s přídomkem error. Budu tedy mít proporcionální error, derivační error a integrální error.

3.4.6.4. Anti-windup

Co se stane, když se v *integral* v rovnici (9) nasčítá velký počet chyb? To se stane například v případě, kdy na dráze budou jen pravotočivé, nebo jen levotočivé zatáčky. Tento error by výrazně předčil ostatní dva errory a mohlo by dojít k tomu, že by po motorech chtěl, aby se roztáčeli rychleji, než je to konstrukčně možné. Proto se zavádí do regulátorů metoda anti-windup. Tato metoda jednou za čas vynuluje *integral* z rovnice (9). Nejčastěji se toto nulování provádí v okamžiku, kdy bychom po motoru chtěli, aby se točil rychleji, než je konstrukčně schopný.

3.4.6.5. PID regulátor

Spojením proporčního erroru, derivačního erroru a integrálního erroru vzniká PID regulátor. Rovnici výpočtu celkového akčního zásahu PID regulátoru nalezeme v (10).

$$U = K_P \cdot e_r + K_D \cdot (e_r - e_m) + K_I \cdot integral \qquad (10)$$

Jednotlivé proměnné vyjadřují stejné věci jako jsme již zvyklí. Tento typ regulátoru je nejlepší regulátor, jaký si zde představíme. Je lepší než všechny popsané regulátory ve všech měřitelných aspektech. Jeho použití však není vždy nejlepší variantou.

3.4.6.6. Nastavení konstant K_P, K_D a K_I

Prozatím jsme mluvili pouze o chování jednotlivých složek a jejich vynásobení konstantami. Ovšem nikdy jsme si neřekli, jakých hodnot jednotlivé konstanty nabývají. To není tak jednoduchý úkol. Tyto konstanty se musí nastavovat ručně a neexistuje jednoznačné nastavení těchto konstant. Každý robot se chová jinak, a tak každý robot bude muset mít i jiné nastavení těchto konstant. Jak se nastavují tyto konstanty, se podíváme po kapitole o programování. Je to z toho důvodu, že k tomu abychom určili přibližné hodnoty těchto konstant, napřed potřebujeme postavit robota a pomocí něho změřit několik údajů.

3.4.7. Programování PD regulátoru

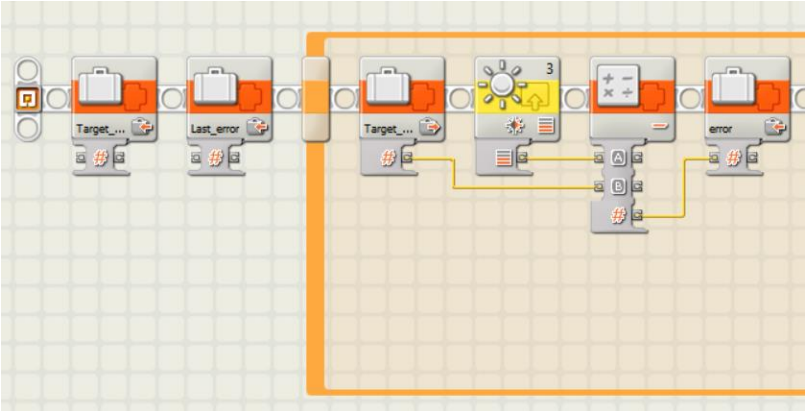
V této podkapitole si ukážeme naprogramování PD regulátoru, který je pro splnění úkolu sledování černé čáry postačující. Jednotlivé kódy jsou okomentovány poznámkami v kódu, a proto je již zde nebudu popisovat. Určitou výjimku zde tvoří NXT-G, kde by bylo značně nepřehledné tento kód popisovat v textové podobě.

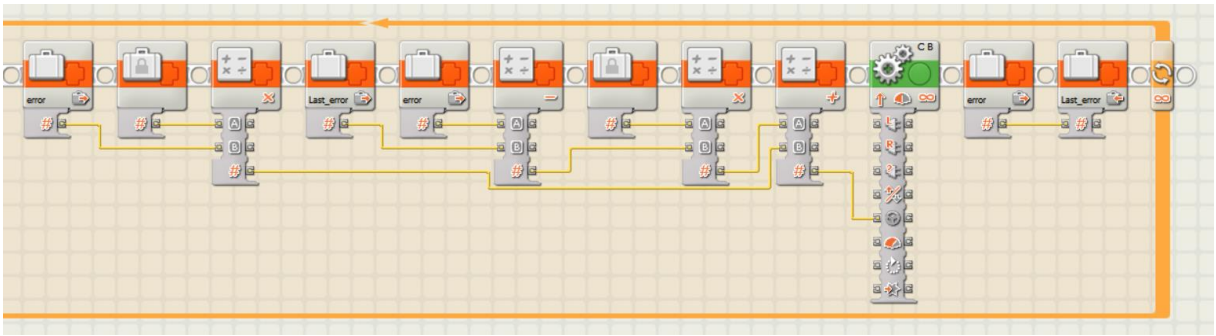
3.4.7.1. Pseudokód

```
//Inicializace konstant
Kp = 0 //konstanta P regulátoru
Kd = 0 //konstanta D regulátoru
target_value = 0 //žádaná hodnota, rozhraní černé a bílé
rychlost_offset = 50 //offset pohybu motorů

//inicializace pomocných proměných
LastError = 0 //proměná na uložení poslední odchylky
Derivate = 0 //uložení derivátu, změny odchylky ve dvou stavech

Prováděj pořád do kola:
        Odečti honotu ze senzoru světla a ulož ji do proměné světlo //Aktuální hodnota senzoru
        error = LightValue – target_value //vypočítej regulační odchylku
        Derivate = error – LastError //spočtení změny regulační odchylky pro potřebi D regulátoru
        Turn = Kp * error+Ki*Integral + Kd*Derivate //Výpočet konstanty pro zatáčení motorů (akční zásah)
        speed_A = rychlost_offset + Turn //spočtení rychlosti pro motor A
        speed_C = rychlost_offset – Turn //spočtení rychlosti pro motor C

        Odeslání hodnot speed_A a speed_C do motoru
        LastError = error //uložení aktuální odchylky pro potřeby budoucího výpočtu
        Počkej 5 milisekund
```

3.4.7.2. NXT-G

Jelikož je kód PD regulátoru v tomto programovacím prostředí značně dlouhý, je rozdělen do několika samostatných obrázků. Bloky v tomto regulátoru jdou po sobě, tak jako jdou po sobě jednotlivé obrázky.





3.4.7.3. NXC

Následující kód ukazuje naprogramování PD regulátoru v prostředí NXC. Samotný kód je okomentovaný formou poznámek.

```
1 //konstanty PD
2 int K_p, K_d;
3
4 //konstanty
5 int target_value, speed_offset;
6
7 //pomocne promene
8 int last_error, derivate, light_value, current_error, turn;
9 int speed_A, speed_C;
10
11 //hlavni cast
12 task main(){
13     K_p = 5; //nastaveni velikosti proporcionalni konstanty
14     K_d = 5; //nastaveni velikosti derivacni konstanty
15
16     target_value = 49; //hodnota, kterou chceme sledovat
17
18     speed_offset = 50; //offset rychlosti
19
20     SetSensorLight (IN_3); //inicializace senzoru světla na poru 3
21     while(true){ //zacatek nekonecne smycky
22                 light_value = Sensor(IN_3); //odecteni hodnoty ze svetelneho senzoru
23                 current_error = light_value - target_value; //regulacni odchylka
24                 derivate = current_error - last_error; //vypocet derivative
25                 turn = K_p*current_error + K_d*derivate; //vypocet akcniho zasahu
26
27                 speed_C = speed_offset - turn; //spocteni rychlosti motoru C
28                 speed_A = speed_offset + turn; //spocteni rychlosti motoru A
29
30                 OnFwd(OUT_C, speed_C); //nastaveni rychlosti motoru C
31                 OnFwd(OUT_A, speed_A); //nastaveni rychlosti motoru A
32                 last_error = current_error; //ulozeni soucasne regulacni odchylky pro D regulator
33                 Wait(5);  //pockej
34     } //konec nekonecne smycky
35 } //konec programu
```

3.4.7.4. PyNXC

Následující kód vznikl jako přepis kódu PD regulátoru z prostředí NXC do PyNXC.

```
1  def main():
2      K_p = 5 #nastaveni velikosti proporcionalni konstanty
3      K_d = 5 #nastaveni velikosti proporcionalni konstanty
4
5      target_value = 49 #hodnota, kterou chceme sledovat
6
7      speed_offset = 50 #offset rychlosti
8      last_error = 0 #nastaveni pro potrebu D regulatoru
9
10     SetSensorLight(IN_3) #inicializace senzoru svetla na poru 3
11
12     while(True):
13         light_value = Sensor(IN_3) #odecteni hodnoty ze svetelneho senzoru
14         current_error = light_value - target_value #regulacni odchylka
15         derivate = current_error - last_error #vypocet
16         turn = K_p*current_error + K_d*derivate #vypocet akcniho zasahu
17
18         speed_C = speed_offset - turn #spocteni rychlosti motoru C
19         speed_A = speed_offset + turn #spocteni rychlosti motoru A
20
21         OnFwd(OUT_C, speed_C) #nastaveni rychlosti motoru C
22         OnFwd(OUT_A, speed_A) #nastaveni rychlosti motoru A
23         last_error = current_error #ulozeni soucasne regulacni odchylky pro D regulator
24         Wait(5) #pockej
```

Tento kód funguje a je odzkoušený na NXT kostce a sledování černé čáry.

3.4.8. Programování PID regulátoru

V této podkapitole si ukážeme, jak naprogramovat PID regulátor s anti-windup v prostředí NXC a PyNXC. Anti-windup je zde důležitý proto, aby nedocházelo k požadavkům NXT kostky na vyšší než maximální možnou rychlost pohybu motorů. V NXT-G si ukážeme z důvodu přehlednosti pouze naprogramování PID regulátoru. Podobně jako v předchozí kapitole kódy jsou okomentované formou poznámek v samotném kódu.

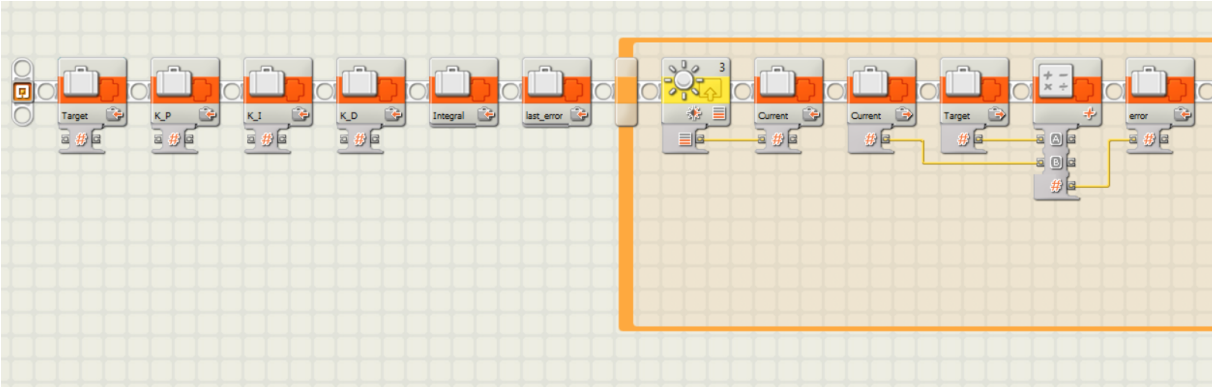
3.4.8.1. Pseudokód

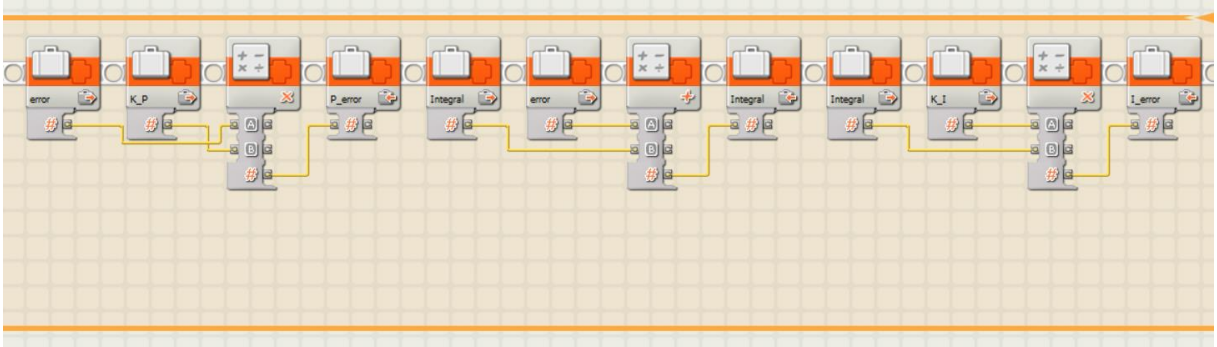
Zde si popíšeme, jak vypadá obecné naprogramování PID regulátoru, v libovolném programovacím prostředí.

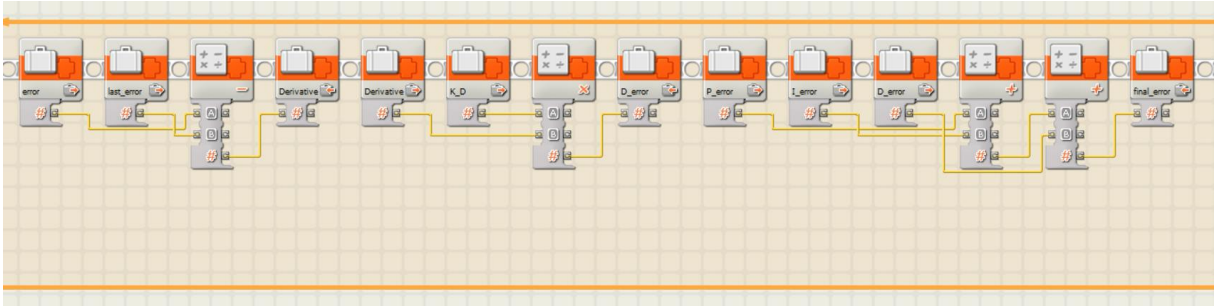
```
//Inicializace konstant
Kp = 0 //konstanta P regulátoru
Ki = 0 //konstanta I regulátoru
Kd = 0 //konstanta D regulátoru
target_value = 0 //žádaná hodnota, rozhraní černé a bílé
rychlost_offset = 50 //offset pohybu motorů

//inicializace pomocných proměných
integral = 0 //proměná na sčítání odchylek
LastError = 0 //proměná na uložení poslední odchylky
Derivate = 0 //uložení derivátu, změny odchylky ve dvou stavech

Prováděj pořád do kola:
        Odečti honotu ze senzoru světla a ulož ji do proměné světlo //Aktuální hodnota senzoru
        error = LightValue – target_value //vypočítej regulační odchylku
        integral = integral + error //sčítání odchylek pro potřeby I regulátoru
        Derivate = error – LastError //spočtení změny regulační odchylky pro potřebi D regulátoru
        Turn = Kp * error+Ki*Integral + Kd*Derivate //Výpočet konstanty pro zatáčení motorů (akční zásah)
        speed_A = rychlost_offset + Turn //spočtení rychlosti pro motor A
        speed_C = rychlost_offset – Turn //spočtení rychlosti pro motor C

        //anti-windup
        Když(speed_C > 100): //když by se motor C měl otáčet dopředu rychleji než může
                speed_C = 100 //nastavení maximální rychlosti na motru
                integral = 0 //vynulování integral
        Když(speed_A > 100) //když by se motor A měl otáčet dopředu rychleji než může
                speed_A = 100 //nastavení maximální rychlosti na motru
                integral = 0 //vynulování integral
        Když(speed_C < -100) //když by se motor C měl otáčet dozadu rychleji než může
                speed_C = -100 //nastavení maximální rychlosti na motru
                integral = 0 //vynulování integral
        Když(speed_A < -100) //když by se motor A měl otáčet dozadu rychleji než může
                speed_A = -100 //nastavení maximální rychlosti na motru
                integral = 0 //vynulování integral
        //konec anti-windup

        Odeslání hodnot speed_A a speed_C do motoru
        LastError = error //uložení aktuální odchylky pro potřeby budoucího výpočtu
        Počkej 5 milisekund
```

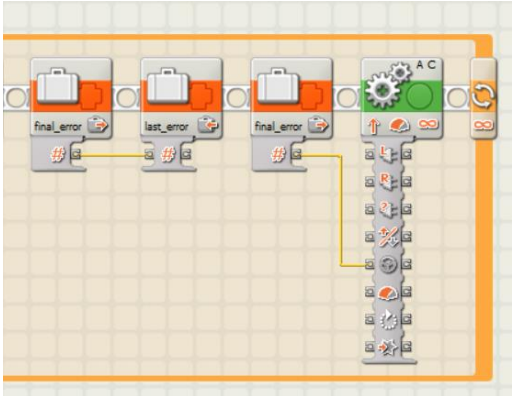
3.4.8.2. NXT-G

Programování PID regulátoru je sice možné, avšak značně nepřehledné. Jelikož je tento kód značně dlouhý, je rozdělen do několika samostatných obrázků. Bloky v tomto regulátoru jdou po sobě, tak jako jdou po sobě jednotlivé obrázky.









3.4.8.3. NXC

Zde je ukázáno, jak se implementuje PID regulátor pro sledování černé čáry v jazyce NXC. Kód je okomentovaný formou poznámek v samotném kódu.

```
 1 //konstanty PID
 2 int K_p, K_i, K_d;
 3
 4 //konstanty
 5 int target_value, speed_offset;
 6
 7 //pomocne promene
 8 int integral, last_error, derivate, light_value, current_error, turn;
 9 int speed_A, speed_C;
10 int counter;
11
12 //hlavni cast
13 task main(){
14      K_p = 7.2; //nastaveni velikosti proporcionalni konstanty
15      K_i = 0.1; //nastaveni velikosti integralni konstanty
16      K_d = 112.5; //nastaveni velikosti derivacni konstanty
17
18      target_value = 47; //hodnota, kterou chceme sledovat
19
20      speed_offset = 25; //offset rychlosti
21
22      integral = 0; //kumulativni error pro potřeby I regulátoru
23
24      SetSensorLight (IN_3); //inicializace senzoru světla na poru 3
```

```
24     while(true){ //zacatek nekonecne smycky
25                 light_value = Sensor(IN_3); //odecteni hodnoty ze svetelneho senzoru
26                 current_error = light_value - target_value; //regulacni odchylka
27                 integral = integral + current_error; //vypocet integral
28                 derivate = current_error - last_error; //vypocet derivative
29                 turn = K_p*current_error + K_i*integral + K_d*derivate; //vypocet akcniho zasahu
30
31                 turn = -turn;
32                 speed_C = speed_offset - turn; //spocteni rychlosti motoru C
33                 speed_A = speed_offset + turn; //spocteni rychlosti motoru A
34
35                 //anti windup
36                 if(speed_C > 100){ //když by se motor C měl otáčet dopředu rychleji než může
37                            speed_C = 100; //nastaveni maximální rychlosti na motru
38                            integral = 0; //nulovani integral
39                 }
40
41                 if(speed_A > 100){ //když by se motor A měl otáčet dopředu rychleji než může
42                            speed_A = 100; //nastaveni maximální rychlosti na motru
43                            integral = 0; //nulovani integral
44                 }
45
46                 if(speed_C < -100){ //když by se motor C měl otáčet dozadu rychleji než může
47                            speed_C = -100; //nastaveni maximální rychlosti na motru
48                            integral = 0; //nulovani integral
49                 }
50
51                 if(speed_A < -100){ //když by se motor A měl otáčet dozadu rychleji než může
52                            speed_A = -100; //nastaveni maximální rychlosti na motru
53                            integral = 0; //nulovani integral
54                 }
55                 //konec anit-windup
```

```
57                 OnFwd(OUT_C, speed_C); //nastaveni rychlosti motoru C
58                 OnFwd(OUT_A, speed_A); //nastaveni rychlosti motoru A
59                 last_error = current_error; //ulozeni soucasne regulacni odchylky pro D regulator
60                 Wait(5); //pockej
61     } //konec nekonecne smycky
62 } //konec programu
```

Pomocí příkazu na řádku 31 lze měnit, zda robot sleduje černou čáru zprava nebo z leva. Stačí zde pouze změnit znaménka před proměnnou *turn*.

3.4.8.4. PyNXC

Zde je ukázáno, jak se implementuje PID regulátor pro sledování černé čáry v jazyce PyNXC. Program je komentovaný formou komentářů v samostatném kódu. Tento kód vznikl jako přepis kódu NXC z předchozí kapitoly.

```
def main():
    K_p = 7.2 #nastaveni velikosti proporcionalni konstanty
    K_i = 0.1 #nastaveni velikosti integralni konstanty
    K_d = 112.5 #nastaveni velikosti derivacni konstanty

    target_value = 47 #hodnota, kterou chceme sledovat

    speed_offset = 25 #offset rychlosti

    integral = 0 #kumulativni error pro potreby I regulatoru

    SetSensorLight(IN_3) #inicializace senzoru svetla na poru 3
    while (true):  #zacatek nekonecne smycky
        light_value = Sensor(IN_3) #odecteni hodnoty ze svetelneho senzoru
        current_error = light_value - target_value #regulacni odchylka
        integral = integral + current_error #vypocet integral
        derivate = current_error - last_error #vypocet derivative
        turn = K_p * current_error + K_i * integral + K_d * derivate #vypocet akcniho zasahu

        speed_C = speed_offset - turn #spocteni rychlosti motoru C
        speed_A = speed_offset + turn #spocteni rychlosti motoru A

        #anti windup
        if(speed_C > 100): #kdyz by se motor C mel otacet dopredu rychleji nez muzee
            speed_C = 100 #nastaveni maximalni rychlosti na motru
            integral = 0  #nulovani integral

        if (speed_A > 100): #kdyz by se motor A mel otacet dopredu rychleji nez muze
            speed_A = 100 #nastaveni maximalni rychlosti na motru
            integral = 0 #nulovani integral

        if (speed_C < -100): #kdyz by se motor C mel otacet dozadu rychleji nez muze
            speed_C = -100 #nastaveni maximalni rychlosti na motru
            integral = 0 #nulovani integral

        if (speed_A < -100): #kdyz by se motor A mel otacet dozadu rychleji nez muze
            speed_A = -100 #nastaveni maximalni rychlosti na motruu
            integral = 0 #nulovani integral
        # konec anit-windup

        OnFwd(OUT_C, speed_C) #nastaveni rychlosti motoru C
        OnFwd(OUT_A, speed_A) #nastaveni rychlosti motoru A
        last_error = current_error #ulozeni soucasne regulacni odchylky pro D regulator
        Wait(5) #pockej
```

Tento kód, podobně jako kód PD v PyNXC, je otestovaný na sledování černé čáry. Kód se podařilo přeložit a nahrát do NXT kostky. Po spuštění program pracoval stanoveným způsobem. Z těchto důvodů hodnotím prostředí PyNXC, jako vhodné prostředí k psaní regulátorů.

3.4.9. Nastavení konstant

Nejprve si musíme nastavit konstantu target_value, tedy hodnotu, kterou bychom chtěli pomocí světelného senzoru sledovat a v ideálním případě se na ní ustálit. Přestože chceme sledovat černou čáru, nenastavíme zde hodnotu 25. Nejraději bychom sledovali něco mezi černou čárou a bílím podkladem, tedy hranu této čáry. Když se světelný senzor bude nacházet nad hranou, změří hodnotu odraženého světla zhruba 50. Ostrý přechod mezi černou a bílou tedy vypadá na senzoru, jako šedá barva. Do target_value tedy uložíme hodnotu 50. Tato hodnota se nachází na poměrně malém úseku v příčném průřezu čáry. Kdybychom chtěli sledovat přímo černou čáru a nacházeli bychom se uprostřed této čáry, nevěděli bychom, kam máme zatáčet, dokud robot tuto čáru neopustí, což je již pozdě na zatáčení. Je to z toho důvodu, že v takovém případě, bychom napravo i nalevo od současné pozice senzoru, změřili pořád stejnou hodnotu odraženého světla. Není si těžké domyslet, že by robot v takovémto případě jel pořád rovně, protože hodnota *error* by byla pořád nula. Pokud nastavíme target_value na 50, pak napravo od ní bude hodnota odraženého světla 60 a nalevo od ní bude hodnota odraženého světla 40. Tím dáváme regulátoru možnost včas reagovat na změnu zakřivení černé čáry, protože i mírné zatočení dráhy výrazně změní hodnotu odraženého světla na senzoru.

Nastavování konstant PID regulátoru není jednoznačně dané. Existují různé metody a postupy, jak dané tři konstanty nastavit. Namátkou můžu zmínit Technika procesní reakční křivky, Zkušební a chybová metoda nebo Metoda Zeigler-Nichols. Z nich poslední dvě si podrobněji popíšeme. Konstanty PID regulátoru se často nastavují empiricky, nebo se často provádí nastavování podle předešlých metod a tyto získané konstanty se ještě upravují pro získání lepšího chování systému. Nastavování konstant v PID regulátoru se často říká ladění PID regulátoru. [21] [22]

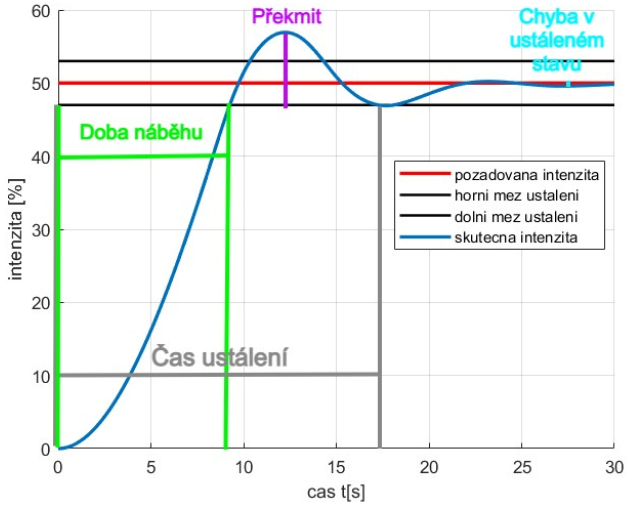
Nejprve si ukážeme, viz. Tabulka 4, jak se změny ve velikosti daných konstant PID regulátoru projeví v chování robota.

Efekt zvyšování parametrů [21]				
Konstanta	Doba náběhu	Překmit	Čas ustálení	Chyba v ustáleném stavu
K_P	Snížení	Zvýšení	Malý vliv	Snížení
K_I	Snížení	Zvýšení	Zvýšení	Eliminuje
K_D	Nedefinováno (malé snížení nebo zvýšení)	Snížení	Snížení	Nemá vliv

Tabulka 4: Význam jednotlivých konstant

Pojmy z Tabulka 4 jsou znázorněny na Obr. 52 a vysvětleny zde:

- **Doba náběhu (Rise time)** - Čas, který robot potřebuje ke změně ze specifické hodnoty odraženého světla na *target_value*.
- **Překmit (Overshoot)** – Překmit je, když robot přesáhne *target_value*, na kterou je nastavený. Po tomto překmitu se zpravidla vrací zpátky k *target_value*.
- **Čas ustálení (Settling time)** - Čas potřebný k tomu, aby robot dosáhl a zůstal v rozsahu určitého procenta (obvykle 5 % nebo 2 %) od *target_value*.
- **Chyba v ustáleném stavu (Steady-State Error**) – Chyba v ustáleném stavu je rozdíl mezi požadovanou polohou robota a skutečnou polohou robota.



Obr. 52 Vysvětleni pojmů

3.4.9.1. Zkušební a chybová metoda

Tato metoda je oproti jiným metodám jednoduchá. K odladění regulátoru podle této metody potřebujeme mít pouze možnost měnit konstanty v PID regulátoru a prostor, kde můžeme daný regulátor testovat.

Nejprve nastavíme konstanty K_P, K_I a K_D na nulu. Poté se postupně zvyšuje proporcionální konstanta K_P a robot se pouští po dráze a pozoruje se, jak se při sledování černé čáry chová. Konstanta se zvyšuje tak dlouho, dokud se nedocílí oscilačního chování robota. Tedy situace, kdy bude robot značně kmitat, kolem černé čáry. Při tomto kmitání by se robot stále měl pohybovat směrem dopředu. Když robot dosáhne tohoto oscilačního stavu, musíme se rozhodnout, zda chceme udělat PID regulátor, nebo jen PD regulátor. Ve druhém případě necháme K_I na nule, pokud je naprogramovaná, a přidáme pouze složku K_D tak, abychom snížili frekvenci kmitání a robotu se touto úpravou zvýší rychlost pohybu po černé čáře. V případě programování PID regulátoru nejprve nastavíme K_I tak, aby se kmitání robota téměř úplně zastavilo. To provedeme opět postupným zvyšováním této konstanty. Dále postupným zvyšováním K_D získáme rychlost pohybu robota. Přitom ale dáváme pozor na to, aby robot zvládal projet zatáčky na naší dráze. Správná hodnota nastavení této konstanty bude tedy poslední hodnota, při které robot dráhu ještě projel. [22]

3.4.9.2. Metoda Zeigler-Nichols

Podobně jako předchozí metoda, je tato metoda založena na postupném zvětšování konstant PID regulátoru. Oproti předchozí metodě je však toto zvyšování sofistikovanější. Ukažme si tedy postup, jak pomocí této metod odladit konstanty PID regulátoru. [21]

1. Nastavíme konstanty K_I a K_D na nulu. Náš PID regulátor se bude chovat pouze jako P regulátor.
2. Nastavíme offset motorů na rozumně malou hodnotu.
3. Nastavíme K_P na hodnotu mezi 1-20.
4. Pustíme robota po čáře a budeme sledovat jeho chování. Pokud nedokáže sledovat černou čáru a často z ní sjíždí, zvýšíme K_P. Pokud naopak robot při sledování čáry značně osciluje snížíme K_P. Takto měníme konstantu K_P do té doby, než je robot schopný sledovat čáru a zároveň znatelně osciluje, ale oscilace není příliš výrazná. Takto nastavenou konstantu K_P nazveme kritické zesílení K_C a zapíšeme si jeho hodnotu.
5. Dále si změříme periodu oscilace jednoho kmitu robota. Tedy čas v sekundách, ve kterém přejde senzor z jedné strany hrany černé čáry na druhou a zpět do výchozí polohy. Pro typické

roboty a vhodně nastavené K_C se čas trvání jedné periody pohybuje od 0,5 do 2 sekund a označíme si ho P_C.

6. Nyní potřebujeme změřit, jak dlouho trvá průběh jednoho cyklu PID regulátoru. To se dá udělat jednoduše tak, že si nastavíme fixní počet opakování a změříme si čas, jak dlouho program běží. Z takto naměřených veličin jednoduchou matematickou operací dělení spočítáme dobu trvání jedné smyčky. Nemusíme se bát, že je potřeba tento čas změřit na desetitisíciny sekundy přesně. Tato metoda je poměrně robustní a dovoluje určitou míru nepřesnosti. Čas trvání jedné smyčky v sekundách si označíme dT.
7. Nyní použijeme Tabulka 5 a spočítáme si hodnoty jednotlivých konstant PID regulátoru. Nemusíme se však omezovat jen na PID regulátory, ale máme si možnost vybrat mezi P, PD nebo PID regulátorem.

Výpočet konstant různých typů regulátorů pomocí metody Ziegler-Nichols			
Typ Regulátoru	K_P	K_I	K_D
P	$0{,}5K_C$	0	0
PD	$0{,}80K_C$	0	$\frac{K_P P_C}{8dT}$
PID	$0{,}60K_C$	$\frac{2K_P dT}{P_C}$	$\frac{K_P P_C}{8dT}$

Tabulka 5: Rovnice výpočtu jednotlivých konstant PID regulátoru

8. Vypočítané konstanty zadáme do naprogramovaného PID regulátoru a spustíme robota tak, aby sledoval černou čáru.
9. Jelikož tato metoda není zcela jednoznačná, je potřeba konstanty ručně upravit, abychom dosáhli lepších výsledků. Pomocí Tabulka 4 se pokusíme zvyšovat a snižovat konstanty, tak dlouho, dokud nebudeme s chováním robota spokojeni. Obecně se doporučuje tyto konstanty měnit zhruba v rozsahu od 30% do -30% od vypočítané hodnoty. Ale je zde možnost s konstantami pohybovat i nad nebo pod touto hranicí. Tato metoda nám tedy dává spíše ukázku, jakých hodnot mohou tyto konstanty přibližně nabývat než jejich přesné určení. Rovněž je si třeba uvědomit, že v našem případě neexistuje jednoznačné nastavení konstant.
10. Po tomto kroku zvýšíme offset rychlosti motorů a zkusíme ručně přenastavit konstanty, popřípadě se vrátíme k bodu 3. a postup zopakujeme.

3.4.9.3. Naměřené hodnoty

Pomocí robota popsaného v kapitole 3.4.3 byli změřeny následující hodnoty, které jsou potřebné k výpočtu konstant PID regulátorů pomocí Metody Ziegler-Nichols.

$$K_C = 2, \tag{11}$$

$$P_C = 0{,}75\ sekund, \tag{12}$$

$$dT = 0{,}006\ sekund. \tag{13}$$

Následně jsem vypočítal konstanty PD regulátoru.

$$K_P = 0{,}8 \cdot 12 = 1{,}6, \tag{14}$$

$$K_D = \frac{9{,}6 \cdot 0{,}75}{8 \cdot 0{,}006} = 25. \tag{15}$$

Poté jsem vypočítal konstanty PID regulátoru.

$$K_P = 0{,}6 \cdot 12 = 1{,}2, \tag{16}$$

$$K_D = \frac{7{,}2 \cdot 0{,}75}{8 \cdot 0{,}006} = 25, \tag{17}$$

$$K_I = \frac{2 \cdot 7{,}2 \cdot 0{,}006}{0{,}75} \approx 0{,}02. \tag{18}$$

3.4.10. Teoretické porovnání jednotlivých typů regulátorů z hlediska jejich výhod a nevýhod

V této podkapitole se podíváme na výhody a nevýhody jednotlivých typů regulátorů. Zaměříme se především na výhody a nevýhody, se kterými se studenti setkají během řešení této úlohy. Rozhodl jsem se sem zařadit i cik-cak regulátor se dvěma senzory, přestože v základní stavební sadě nejsou dva světelné senzory. Tento regulátor je však jednoduchý a setkáváme se s ním často na internetu při různých demonstracích. Druhý regulátor, který zde je a jeho kód není přímo popsaný v tomto sborníku je P regulátor. P regulátor je ale důležitý pro základní pochopení funkce složitějších regulátorů, stejně tak je potřebný pro ladění konstant v PID i PD regulátoru. My ho můžeme získat jednoduše tím, že buď v PD regulátoru nastavíme K_D na nulu, nebo v PID regulátoru nastavíme K_D i K_I na nulu.

Výhody	Nevýhody
- Jednoduchá implementace a porozumění jeho funkci	× Pomalý, robot vždy jezdí cik cak i na rovné čáře (zpravidla jede jen jeden motor) × Problémy na komplikovaných čárách, aby projel komplikovanější čáry je potřeba, aby jel robot pomaleji, což znamená, že se robot celkově zpomalí

Tabulka 6: Výhody a nevýhody jednosenzorového zig-zag regulátoru

Výhody	Nevýhody
- Robot, oproti případu s jedním senzorem, lépe drží sledovanou čáru, která je uzavřená mezi dvěma senzory - Jednoduchá implementace a porozumění jeho funkci	× Pomalý, robot vždy jezdí cik cak i na rovné čáře (zpravidla jede jen jedno kolo) × Problémy na komplikovaných čárách, aby projel komplikovanější čáry, je potřeba, aby jel robot pomaleji, což znamená, že se robot celkově zpomalí

Tabulka 7: Výhody a nevýhody dvousenzorového regulátoru

Výhody	Nevýhody
- Relativně jednoduchý, ale velice účinný - Robot jede oběma motory, ale natáčí se podle vzdálenosti od uspokojivé hodnoty -> projede rychle i komplikovanější čáry - Po přímé čáře jede téměř rovně	× Vyžaduje čas pro nastavení konstanty P

Tabulka 8: Výhody a nevýhody proporcionálního regulátoru

Výhody	Nevýhody
- Rychlejší doba ustálení na uspokojivé hodnotě, oproti proporcionálnímu regulátoru - Robot jede rychleji než v případě P	× Čas pro nastavení PD konstant × Robot rychle osciluje kolem ideální hodnoty

Tabulka 9: Výhody a nevýhody PD regulátoru

Výhody	Nevýhody
- Robot jede téměř hladce a neosciluje kolem rovnovážné polohy - Rychlost robota oproti předchozím regulátorům - Nejlepší regulátor ze zde jmenovaných ve všech měřitelných parametrech	× Nejkomplexnější regulátor, který jsme si představili × Vyžaduje nejvíce času ze zde jmenovaných regulátorů pro nastavení konstant × Pro pochopení jeho funkce vyžaduje znalost vysokoškolské matematiky

Tabulka 10: Výhody a nevýhody PID regulátoru

3.4.11. Závěr

V tomto úkolu jsme měli za úkol sledovat černou čáru. K tomuto úkolu jsme si nejprve sestavili jednoduchého robota. Posléze jsme si ukázali, jak se dá černá čára sledovat pomocí cik-cak regulátoru. Dlouho jsem váhal, zda sem tento typ regulátoru umístit. Rozhodl jsem se ho sem zahrnout, protože je jednoduchý na pochopení a slouží, jako úvod do regulátorů a jako odrazový můstek pro pochopení P regulátoru. Po popisu cik-cak regulátoru a jeho naprogramovaní jsem obecně popsal lineární regulátory typu PID a vysvětlil, jak se dají použít k sledování černé čáry. Samozřejmě pomocí nich nesledujeme černou čáru, ale hranu černé čáry. Ukázal jsem, jak se naprogramuje PD regulátor, který je postačující pro splnění tohoto úkolu. Dále jsem nastínil programování PID regulátoru s anti-windup, který se bohužel v průběhu testování na černé čáře v domácích podmínkách ukázal jako nezbytný. PID regulátor nabízí oproti PD regulátoru hladší chod robota bez zbytečných zákmitů. Dále jsem vysvětlil nastavování konstant regulátorů typu PID. Toto nastavování bylo ukázáno pomocí Zkušební a chybové metody a pomocí metody Ziegler-Nichols, kdy jsem ukázal i příklad výpočtu konstant. Jako poslední jsem přehledně shrnul teoretické výhody a nevýhody jednotlivých typů regulací pro sledování černé čáry.

3.5. Úkol 4: Objíždění překážky na dráze

V tomto úkolu je naším cílem objet překážku, během toho, co sledujeme černou čáru. Samotné sledování černé čáry jsme již vyřešili v předchozím úkolu. Takže naším úkolem bude pouze napsat část kódu, která zajistí vyhnutí se překážce.

3.5.1. Zadání

Cílem tohoto úkolu je naprogramovat robota tak, aby samostatně, co možná nejrychleji, projel vyznačenou dráhu od startu do cíle. Mezi startem a cílem je na podložce vyznačena černá čára, která znázorňuje trajektorii, podle které se bude robot pohybovat. Modifikace proti původní úloze sledování černé čáry je, že se tentokrát na libovolném místě černé dráhy bude nacházet překážka, kterou musí robot při průjezdu dráhou libovolně objet. Během plnění úkolu je nepřípustné, aby robot jakýmkoliv způsobem komunikoval se studenty. Zároveň je nepřípustné, aby robot sjel z plochy, na které je vyznačená dráha. Také se nesmí stát, aby se jakkoliv dotknul překážky.

3.5.1.1. Popis soutěžní plochy

Popis soutěžní plochy se nijak neliší od popisu soutěžní plochy, který nalezneme v kapitole 3.4.1. Proto vás odkážu na tento popis.

3.5.1.2. Popis překážky

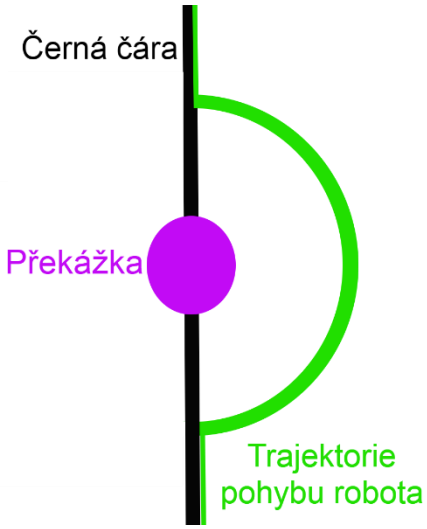
Překážka může být umístěna na libovolné místo soutěžní plochy. Musí být však umístěna přímo na černé čáře tak, aby se dala objet a při tom se robot nepřiblížil nebezpečně k okraji soutěžní plochy, čímž by hrozil jeho pád a poškození.

Překážku má tvořit válcový předmět, například láhev o dostatečné výšce, s výhodou lze použít kola robota spojená dohromady podle plánku dostupného v digitální příloze této práce. Přitom musí být zajištěno, aby ji ultrazvukový senzor umístěný na robotovi ve výšce 11 cm od podložky dokázal detekovat při přímé jízdě na ni. Při bočním průjezdu a měření již není schopnost rozlišení zaručena. Boční průjezd je definován, tak že ultrazvukový senzor má robot umístěný na boku, směřuje tedy kolmo na směr jízdy robota. Překážka bude na dráze umístěna vždy tak, aby ji robot mohl zaznamenat. Musí být tedy umístěna tak, aby na ni robot najížděl přímo.

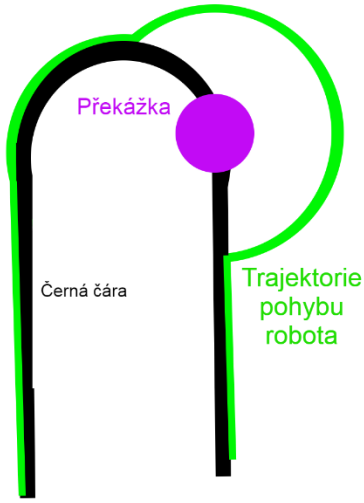
3.5.2. Rozbor problému

Naším úkolem je jet po černé čáře a detekovat překážku na ní, tu objet a pak opět pokračovat v jízdě po černé čáře. K tomuto účelu budeme potřebovat dva motory, abychom se mohli pohybovat po soutěžní ploše všemi směry. Dále budeme potřebovat světelný senzor, abychom mohli detekovat černou čáru, popřípadě její hranu, od bílého zbytku soutěžní podložky. Jako další senzor budeme potřebovat ultrazvukový senzor, abychom mohli detekovat překážku. Použití ultrazvukového senzoru je nezbytnost, protože při použití dotykového senzoru bychom se museli překážky dotknout, což je v rozporu s pravidly.

Jako nejlepší možnost, jak překážku objíždět, se jeví objíždět překážku po kružnici, čímž je zajištěno napojení zpět na čáru, tak jak je naznačeno Obr. 53 a Obr. 54.



Obr. 53 Objíždění překážky I.



Obr. 54 Objíždění překážky II.

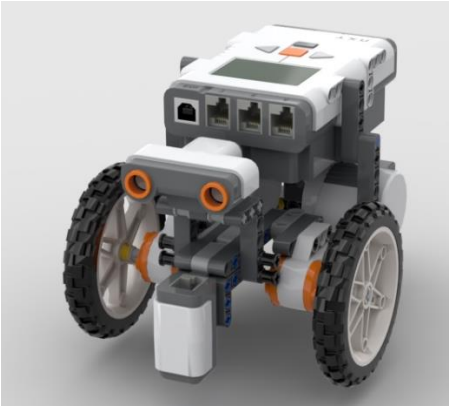
Jízdy po kružnici docílíme tím, že zapneme pravý i levý motor na různou rychlost otáčení. Toto za nás mohou provést samostatná programovací prostředí.

Dále si zde autor dovolí drobné zjednodušení. Kvůli přehlednosti psaní kódů a vysvětlování si zde rozebere pouze sledování čáry pomocí cik-cak regulátoru. Nicméně sledování čáry a objíždění překážky společně s PID regulátorem autor též vypracoval a je k nalezení v kapitole 4.4.1.

Sledovat čáru oproti minulému úkolu nebudeme zprava, ale tentokrát zleva. Jelikož budeme sledovat černou čáru zleva, budeme překážku objíždět zprava. To proto, abychom museli při návratu přejet celou šířku černé čáry a dostali bychom tak čas vhodný na správný návrat na požadovanou trasu. Kdybychom objížděli překážku z leva a černou čáru sledovali taky z leva, nedocházelo by ke správnému napojení zpět na trasu po objetí překážky.

3.5.3. Konstrukce

Při konstrukci tohoto robata jsem vyšel z konstrukce robota pro sledování černé čáry. K takto sestavenému robotu jsem připojil na port 4 ultrazvukový senzor. Tato drobná úprava je popsána v návodu, který je k nalezení v digitální příloze této práce. Obr. 55 znázorňuje postaveného robota, který je uzpůsobený pro tento úkol. Pomocí ultrazvukového senzoru bude robot měřit vzdálenost mezi ním a překážkou. Jinak se popis neliší od popisu uvedeném v kapitole 3.4.3.



Obr. 55 Sledovač černé čáry společně s ultrazvukovým senzorem

Pravý motor sledovače čáry je připojený k portu C na NXT kostce. Levý motor je pak připojený k portu A. Světelný senzor sloužící ke sledování černé čáry je připojený na port 3. Ultrazvukový senzor je pak připojený k portu 4 NXT kostky. Pomocí tohoto senzoru bude robot měřit vzdálenost mezi ním a překážkou.

3.5.4. Programování

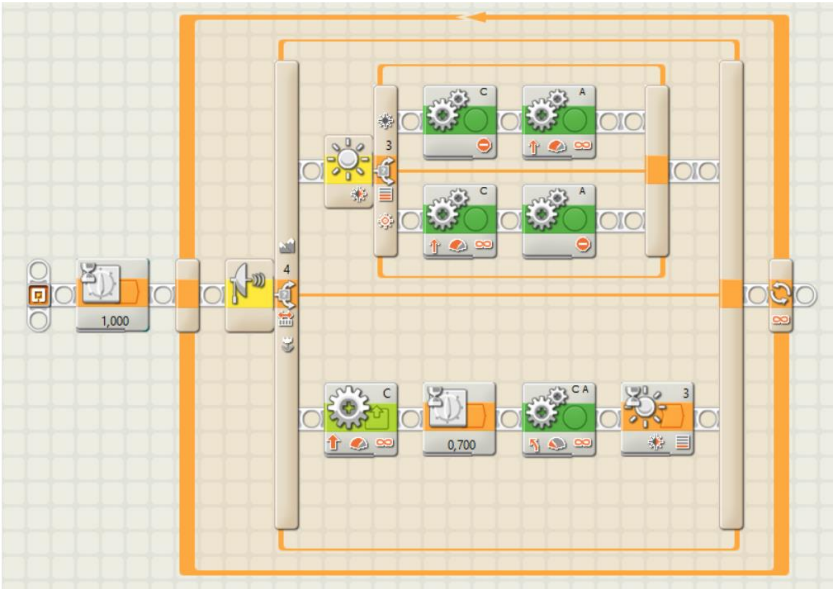
Program se skládá ze dvou hlavních větví. První větev říká, jak se mají překážky objíždět. Druhá větev pak kóduje chování robota v případě, že jede po černé čáře. Pro sledování černé čáry jsem zvolil cik-cak regulátor z důvodu jeho lehkého pochopení a přehlednosti kódu. Samozřejmě se tento regulátor dá nahradit za PD, popřípadě PID regulátor.

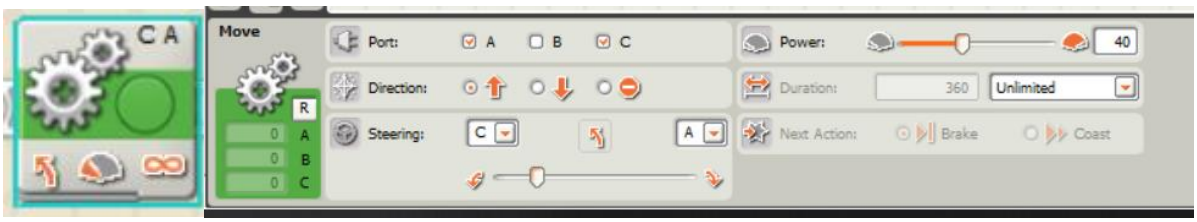
3.5.4.1. Pseudokód

```
Zapnutí programu
Prováděj pořád do kola:
        Odečti hodnotu ze ultrazvukového senzoru a ulož ji do proměnné vzdálenost
        Když vzdálenost < 20:
                Zapni levý motor
                Počkej na otočeni o 90 stupňu
                Vyjeď po kruhové dráze
                Jeť tak dlouho dokavaď opět nenarazíš na černou čáru
        Jinak:
                Odečti honotu ze senzoru světla a ulož ji do proměné světlo
                Když svetlo < 50:
                        Spusti pravý motor
                        Vypni levý motor
                Jinak:
                        Spusti levý motor
                        Vypni pravý motor
```

3.5.4.2. NXT-G

Ukázka naprogramování jednoduchého kódu pro jízdu robota po černé čáře a vyhýbání se překážkám. Rovněž je zde ukázáno nastavení obou bloků **Switch** a bloku **Move** pro objíždění překážky. Robot díky tomuto bloku a jeho nastavení jede po kruhové trajektorii. Dále je zde ukázáno nastavení bloku **Wait**, na kterém se program zastaví v okamžiku, kdy robot objíždí překážku a čeká, až narazí na černou čáru.





První **Switch** je v módu *Sensor*, kdy pomocí ultrazvukového senzoru měří vzdálenost překážky před robotem. Pokud je překážka před robotem blíže než 20 centimetrů, aktivuje se spodní větev kódu a robot začne překážku objíždět. V horní větvi kódu je pak naprogramován klasický zig-zag regulátor, tak jak je popsán v předchozí úloze s tím rozdílem, že tentokrát sleduje černou čáru z leva.

3.5.4.3. NXC

Následující kód ukazuje, jak lze programově vyřešit tento úkol v jazyce NXC. Kód jako celek komentován již nebude, protože je popsaný formou poznámek přímo v kódu a také protože všechny jeho stěžejní části již byly popsané, s jedinou výjimkou, kterou si popíšeme pod tímto kódem.

```
1 task main(){ //zacatek programu
2 SetSensorLowspeed(IN_4); //inicializace ultrazvukoveho senzoru senzoru
3 SetSensorLight(IN_3); //inicializace senzoru svetla senzoru
4
5 int distance; //inicialzice promenne distance
6 int light; //inicializace promenne target
7
8     while(true){ //nekonecna smycka
9                 distance = SensorUS(IN_4); //nacteni vzdalenosti z ultrazvukoveho senzoru a ulozeni do promenne distance
10                if(distance < 20){ //pokud dorazim k prekazce spustim objizdeni
11                            Off(OUT_AC); //vypnu oba motory
12                            Wait(100); //pockam
13                            OnFwd(OUT_C, 75); //robot se bude otacet
14                            Wait(700); //pockam
15                            Off(OUT_AC); //vypnu oba motory
16                            Wait(100); //pockam
17                            OnFwdSync(OUT_AC, 30, -10); //jedu objema motory s turn -10, cimz robot opisuje kruznici
18                            while(Sensor(IN_3) > 50){ //cekam dokavad opet nenarazim na cernou caru
19                                   Wait(10); //pockam
20                            Wait(100); //pockam
21                            }
22                }else{//zde je kod zig zag
23                            light = Sensor(IN_3); #nacteni urovne svetla ze svetelneho senzoru a ulozeni
24                            if(light < 50){ #pokud je uroven svetla nizsi nez 50
25                                     OnFwd(OUT_A, 50); #zapnu motor na portu A
26                                     OnFwd(OUT_C, 0); #vypnu motor na portu C
27                            }else{ //jinak
28                                   OnFwd(OUT_C, 50); //zapnu motor na portu C
29                                   OnFwd(OUT_A, 0); //vypnu motor na portu A
30                            }
31                }//konec kodu zig zag
32                Wait(5); //pockam
33      } //konec nekonecne smycky
34 } //konec programu
```

Neokomentovaná část kódu, se nachází mezi řádky 10 a 20. Tato část určuje, jak robot objíždí překážky. Nejprve robot vypne oba motory a počká 100 ms. Dále se pomocí příkazu OnFwd() otočí, následně se opět vypnou na malý okamžik oba motory. Poté se pomocí příkazu OnFwdSync() rozjede dopředu. Ovšem díky tomu, že je třetí parametr nastavený na -10, nepojede rovně dopředu, ale bude vykonávat pohyb po kružnici. Tato kružnice je umístěná tak, aby robot podle ní mohl objet překážku, tak jak je požadováno v zadání. Dále je pak pomocí smyčky while() zajištěno, aby robot tento pohyb po kružnici vykonával tak dlouho, dokud nenarazí na černou čáru. Až na tuto čáru narazí, napojí se na ni a dále po ní pokračuje.

3.5.4.4. PyNXC

Kód v PyNXC je velice podobný kódu v NXC, jen je napsaný v syntaxi Pythonu. Kód opět nebudu nějak komentovat, protože jde již o probranou problematiku a další komentář se nachází přímo v kódu.

```
def main():
    SetSensorLowspeed(IN_4) #inicializace ultrazvukoveho senzoru senzoru
    SetSensorLight(IN_3) #inicializace senzoru svetla senzoru

    while(True): #nekonecna smycka
        distance = SensorUS(IN_4) #nacteni vzdalenosti z ultrazvukoveho senzoru a ulozeni do promenne distance
        if(distance < 20): #pokud dorazim k prekazce spustim objizdeni
            Off(OUT_AC) #vypnu oba motory
            Wait(100) #pockam
            OnFwd(OUT_C, 75) #robot se bude otacet
            Wait(700) #pockam na otoceni
            Off(OUT_AC) #vypnu oba motory
            Wait(100) #pockam
            OnFwdSync(OUT_AC, 30, -10) #jedu objema motory s turn -10, cimz robot opisuje kruznici
            while(Sensor(IN_3) > 50): #cekam dokavad opet nenarazim na cernou caru
                Wait(10) #pockam
            Wait(100) #pockam
        else: #zde je kod zig zag
            light = Sensor(IN_3) #nacteni urovne svetla ze svetelneho senzoru a ulozeni
            if(light < 50): #pokud je uroven svetla nizsi nez 50
                OnFwd(OUT_A, 50) #zapnu motor na portu A
                OnFwd(OUT_C, 0) #vypnu motor na portu C
            else: #jinak
                OnFwd(OUT_C, 50) #zapnu motor na portu C
                OnFwd(OUT_A, 0) #vypnu motor na portu A
        Wait(5) #pockam
```

Tento kód se úspěšně podařilo přeložit do jazyka NXC a následně nahrát do NXT kostky. Po spuštění programu robot sledoval černou čáru a objížděl překážku, proto hodnotím prostředí PyNXC jako vhodné pro tento úkol.

3.5.5. Obecná doporučení k naprogramování PID regulátoru

Jelikož zig-zag regulátor není nejlepší z hlediska jeho vlastností a jeho použití na závěrečné dráze nejspíše nepovede k bodovému zisku, je zde potřeba místo tohoto regulátoru použít PID nebo alespoň PD regulátor.

Nejprve bych upozornil studenty, že konstanty určené v předešlé úloze nebudou na takto upravený regulátor sedět, ale bude je potřeba znovu odladit. Nelekejte se, pokud po zadání konstant z předchozí úlohy se robot nepohybuje, či jen se točí dokola. Dále je potřeba upravit opětovné napojení na černou čáru po objetí překážky. Toto v případě regulátoru cik-cak nebylo potřeba, protože si to regulátor vyřešil sám tím, jak robot zatáčí prudce doprava, když nevidí černou čáru. V podstatě stačí zkopírovat toto chování za stávající část objíždění překážky.

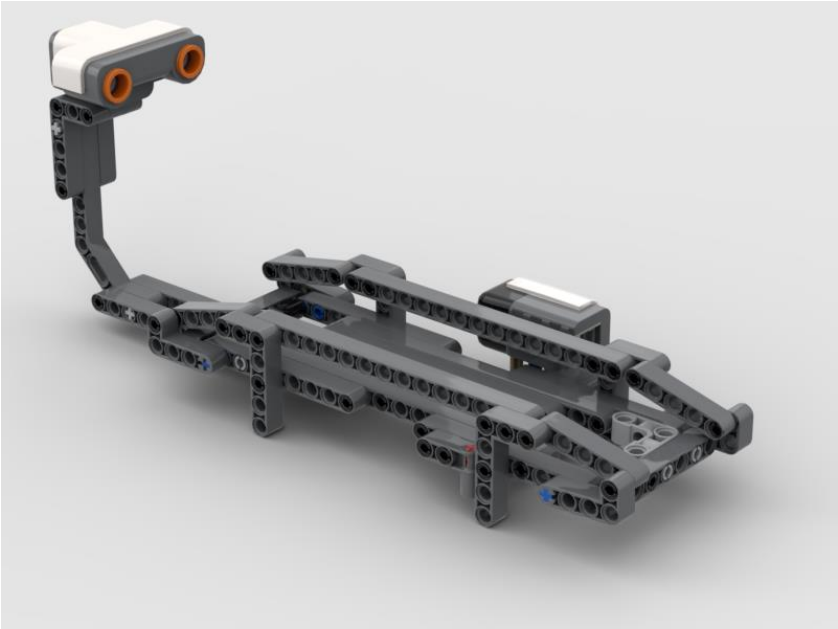
3.5.6. Závěr

V tomto úkolu jsme měli sledovat černou čáru a vyhýbat se překážkám, které jsou na ní umístěné. Samotný koncept sledování černé čáry již byl hotový z úkolu 3. Naším hlavním úkolem bylo vymyslet a napsat koncept samotného objíždění překážky. Překážku jsme se rozhodli objíždět po kruhové dráze, protože tak je zaručeno, že na černou čáru znovu narazíme. Tuto skutečnost znázorňuje Obr. 53 a Obr. 54. Kvůli přehlednosti kódu pro vysvětlování jsem se rozhodl, že zde uvedu regulátor cik-cak. Nicméně při řešení této úlohy vzniklo naprogramování i PID regulátoru, které je k nalezení v kapitole 4.4.1. Největší problém při této úloze je detekce překážky pomocí ultrazvukového senzoru. Problém nastává, když je překážka detekovaná později, než je předpokládáno. Podle kódu objížděcí manévr kolem

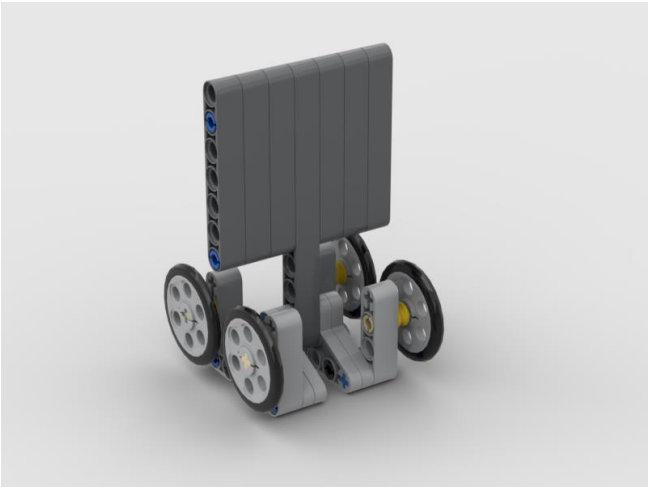
překážky začíná 20 centimetrů před ní. Praxe a následná měření však ukázali, že se překážka detekuje, až 10 centimetrů před překážkou. Toto číslo bylo stanoveno na základě průměrování 5 měření. Je tedy vidět, že senzor, pomocí kterého byl robot testovaný, nedosahuje tabulkové přesnosti, kterou uvádí společnost Lego. Tato tabulková přesnost by měla být $\pm$ 3 centimetry. Dále bych rád zdůraznil, že při programování PD/PID regulátoru je potřeba dané konstanty znovu odladit. Autora sborníku zaujalo, jak se konstanty určené zde a určené v případě pouhého sledování černé čáry mezi sebou liší.

3.6. Úkol 5: Robot číšník

V tomto úkolu bude naším cílem projet předem neznámou dráhu tak, aby nám vozík nesjel z plošiny, která je na robotu umístěná vodorovně ve směru jeho jízdy. Foto této plošiny je na Obr. 56. Foto vozítka je pak na Obr. 57.



Obr. 56 Plošina (rampa) pro stabilizaci vozíku



Obr. 57 Vozík, který balancuje na plošině

3.6.1. Zadání

Cílem tohoto úkolu je bezpečně převést po hrací ploše vozík, který se nachází na rampě. Tato rampa je libovolným způsobem připevněná k robotovi.

Na zadané hrací ploše, bude vytyčena dráha, kterou musí robot projet, aniž by mu vozík spadl z plošiny. Návod na stavbu plošiny s vozíkem pro stavebnice lego je dostupný zde. Rampa se může na robotovi libovolně naklánět. Je zakázáno, aby se vozík dotkl v průběhu jízdy jakékoliv jiné části robota než rampy. Takový dotyk povede k diskvalifikaci.

Soutěžní dráha je přímá, s robotem tedy není potřeba zatáčet, ale obsahuje stoupání a klesání o maximálním sklonu 16°. Foto možné podoby soutěžní dráhy je na Obr. 58.

Obr. 58 Foto sloutěžní plochy

3.6.2. Rozbor problému

Rampu musíme během jízdy naklánět tak, aby byla rovnoběžná s povrchem Země. Tímto nakláněním docílíme toho, že vozík umístěný na rampě z rampy nesjede ani za předpokladu, že budeme s robotem překonávat stoupání nebo klesání. Bez naklánění rampy by jistě vozík z rampy sjel, nebo se dotkl jiné části robota, což by vedlo k diskvalifikaci robota.

Soutěžní dráha je přímá. Není tím pádem potřeba s robotem umět zatáčet. V tomto případě bychom mohli použít na pohyb po dráze pouze jeden motor. Bohužel se v průběhu stavby robota ukázalo, že jeden motor je nedostatečný a robot pomocí něj nevyjede stoupání.

Jedním z možných řešení, jak dosáhnout správného průjezdu dráhou, je použití kyvadla, na které by se rampa umístila. Toto řešení ovšem není dostatečně reprezentativní. Proto si zde ukážeme nastavování náklonu rampy pomocí PD regulátoru. Jako třetí možnost regulace se jevila vícestavová regulace. Ovšem tuto regulaci se nepodařilo úspěšně implementovat do kostky NXC.

V následujícím odstavci shrnuji PD regulátor. Rampu umístíme na servomotor a pomocí servomotoru a zpětné vazby od gyroskopického senzoru budeme řídit jeho natočení. Pomocí tohoto natočení se nakloní i rampa, tak aby nedocházelo ke sjetí vozítka z rampy. V původním plánku na stavbu rampy je znázorněný i ultrazvukový senzor, který je ovšem značně nespolehlivý. Ultrazvukový senzor zanášel do regulace příliš nejistot a šumu, proto jsem ho nemohl použít pro regulaci. Příkladem budiž situace, kdy

jsem položil vozík na rampu a ultrazvukový senzor měřil vzdálenost mezi ním a vozíkem 13 cm. Poté jsem vozík posunul o 2 cm a ultrazvukový senzor stále měřil vzdálenost 13 cm.

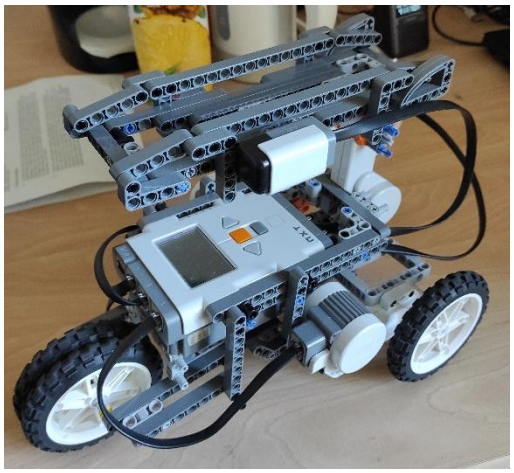
Ultrazvukový senzor jsem se v průběhu pokusů pokoušel nahradit senzorem světla v režimu měření odraženého světla. Bohužel toto nahrazení nepřineslo kýžený výsledek, jelikož je světelný senzor v sadě Lego Mindstorms NXT výrazně závislí na okolním světle. Pokusy o dopsání filtrace odraženého světla od okolního skončily neúspěchem.

3.6.3. Konstrukce

Soutěžní robot je zkonstruovaný tak, aby byl co možná nestabilnější a plošina s vozíkem podléhala při jízdě co možná nejmenšímu kmitání způsobenému touto jízdou. Vozík je tedy konstruovaný, tak aby měl těžiště blízko Zemi a měl velký rozvor kol.

Protože se bude pohybovat pouze dopředu, jsou použity dva motory, které je ovšem potřeba synchronizovat. Stačilo by sice použití jednoho motoru, ale jak již bylo uvedeno dříve, robot pomocí jednoho motoru nedokázal překonávat stoupání na dráze.

Jelikož robot musí umět naklánět plošinu, třetí motor ze sady je pevně spojený s robotem a zároveň je na něm připevněná rampa. Foto soutěžního robota je na Obr. 59.



Obr. 59 Fotografie soutěžního robota

Levý motor pro pohyb robota je připojený na port A, pravý motor je pak připojený na port B. Středový motor pro ovládání náklonu plošiny je připevněný na port C. Gyroskopický senzor, který je jediný použitelný senzor pro stabilizaci vozíku, je připojený na port 2. Dotykový senzor je připojený na port 1.

Ke stavbě robota vznikl návod, který je k nalezení v digitální příloze této práce. Celý robot je zde rozdělen do tří částí, které se musí mezi sebou propojit. První částí je podvozek, druhou je kostka NXT se středovým motorem a třetí je pak samotná plošina. Jako největší nevýhodu tohoto designu vozítka vidím obtížný přístup k baterce a jejímu portu pro nabíjení.

3.6.4. Programování PD regulátoru

Toto naprogramování PD regulátoru do NXT kostky bude popsáno nejprve v pseudokódu, kde budou vysvětleny jednotlivé části programu, které nám zaručí, že se program bude chovat předvídatelným způsobem. Následně bude ukázáno naprogramování v prostředí NXC. Jako poslední bude popsán kód

napsaný v PyNXC, který je otestovaný a odzkoušený a kód se chová podobně jako v případě kódu napsaného v NXC.

3.6.4.1. Pseudokód

```
Zapnuti programu
Nastaveni konstant

Inicializace gyroskopickeho senzoru
Inicializace dotykoveho senzoru

Umisteni rampy, tak aby se samovolne nepohybovala

Nastaveni pohybu robata smerem dopreda
Spocitani offsetu gyroskopickeho senzoru
Vypnuti motoru

Nastaveni rampy, tak aby byla vodorvna s povrchem Zeme a dal se na ni umistit vozik

Pockani na zmackuti dotykoveho senzoru a jeho opetovne pusteni

Nastaveni pohybu robota smerem vpred

uhel = 0
Opakuj v nekonecne smycce:
        nacti hodnotu z gyroskopickeho senzoru
        uhel = uhel + casSmycky*rychlost //vypocitej uhel natoceni plosinky
        regulacniOdchylka = uhel - zadanyUhel //spocitej regulacni odchylku
        derivative = regulacniOdchylka - last_error //spocitej derivacni error
        akcniZasah = K_p*egulacniOdchzlka+K_d*derivative //spocitej akcni zasah

        omez akcni zasah

        aplikuj akcni zasah na motor a tim natoc rampu

        last_error = regulacni odchylka
        pockej 20 ms
```

Po inicializaci senzorů přichází část, která se stará o měření offsetu. Nejprve se umístí rampa tak, aby byla opřená o kostku NXT a tím se co nejméně samovolně pohybovala. Následně se celý robot umístí na šikmou plošinu a pustí se pohyb robota dopředu. Teprve v tomto okamžiku se začíná měřit offset gyroskopického senzoru. Tyto úkony se před samotným měřením offsetu provádí proto, aby změřený offset byl co nejpřesnější.

Následně se plošina umístí do vodorovné polohy. V tomto okamžiku robota umístíme do startovního čtverce a na rampu položíme vozík. Po přípravných krocích se stisknutím a opětovným uvolněním tlačítka spustí pohyb robota a spustí se regulace natočení plošiny vzhledem ke kostce NXT. Toto natočení řídí PD regulátor, který je podobný regulátoru v 3.4.7. Změna zde spočívá v omezení akčního zásahu. Toto omezeni se provádí v případě, když velikost akčního zásahu vystoupí nad určitou hranici, pomocí podmínek se sníží velikost akčního zásahu. Toto snížení se provádí z důvodu, aby pohyb rampy nebyl „trhaný“ a těmito záseky neposouval vozík mimo plošinu.

3.6.4.2. NXC

Následující kód je popsaný formou komentářů a jeho funkce odpovídá chování kódu popsaného v pseudokódu v předchozí kapitole. Z těchto důvodů nebude kód komentován, ale zaměřím se na problémy způsobené s programováním robota. Tyto problémy jsou rozebrané převážně v kapitole 3.6.6. Kód je rozdělený na jednotlivé části, z nichž každá vykonává jeden konkrétní úkol. Jednotlivé řádky kódu jdou po sobě, tak jak jdou po sobě řádky napsané v tomto sborníku.

Definice proměnných:

```
1 //konstanty PD
2 float K_p, K_d;
3 float target_value, limitations;
4
5 //pomocne promenne pro vypocet PD
6 float current_error, turn, offset, last_error, derivate;
7
8 //pomocne promenne pro urceni uhlu
9 float comulative, angle, speed;
10
11 //zbyle pomocne promenne
12 float flag;
13 long tickLast, tickNow;
```

Nastavení konstant:

```
16 task main(){
17      K_p = 7; //nastaveni velikosti proporcionalni konstanty
18      K_d = 5; //nastaveni velikosti derivacni konstatnty
19      limitations = 55; //velikost omezeni akcniho zasahu
20      target_value = 0; //hodnota, kterou chceme sledovat
```

Inicializace senzorů:

```
22      SetSensorTouch(IN_1); //inicializace dotykoveho senzoru
23      SetSensorHTGyro(IN_2); //inicializace gyroskopickeho senzoru
```

Znehybnění plošiny a její opření o kostku NXT:

```
25      OnFwd(OUT_C, 20); //znehybneni rampy
26      Wait(2000); //cekani na znehybneni rampy
27      Off(OUT_C); //vypnuti pohybu rampy
```

Výpočet offsetu gyroskopického senzoru:

```
29      comulative = 0; //nastaneni comulative na nulu
30
31      Wait(5000); //cekani z duvodu ustaleni senzoru
32      OnRevSync(OUT_AB, 40, 0); //spusteni jizdy robota
33      Wait(2000); //cekani z duvodu ustaleni senzoru
34      for(int i = 0; i <30; i++ ){ // smycka se 30 opakovanimi
35              comulative = comulative + SensorHTGyro(IN_2); //nacteni dat a jejich pricteni ke komulative
36              Wait(100); //cekani
37      }
38      Off(OUT_AB); //vypnuti jizdy robota
39      offset = comulative/30; //vypocet offsetu
40      PlaySound(SOUND_LOW_BEEP); //oznameni o ukoncneni mereni offsetu
```

Výpočet offsetu se provádí při pohybu robota po nakloněné rovině, aby se offset změřený co nejvíce přiblížil skutečnému offsetu senzoru.

Umístění plošiny do vodorovné polohy vzhledem k povrchu Země.

```
42      RotateMotorEx(OUT_C, 30, -45, 0, false, true); //ustaleni rampy do vodorovne polohy
43      Wait(10); //cekani
44      OnFwd(OUT_C, -10); //drzeni rampy ve vodorovne poloze
45      Wait(50); //cekani
```

V této fázi kódu umístíme robota do startovního čtverečku a na plošinu mu umístíme vozík.

Čekání na zmáčknutí a opětovné uvolnění tlačítka na dotykovém senzoru, následné rozjetí robota společně s vynulováním proměnných.

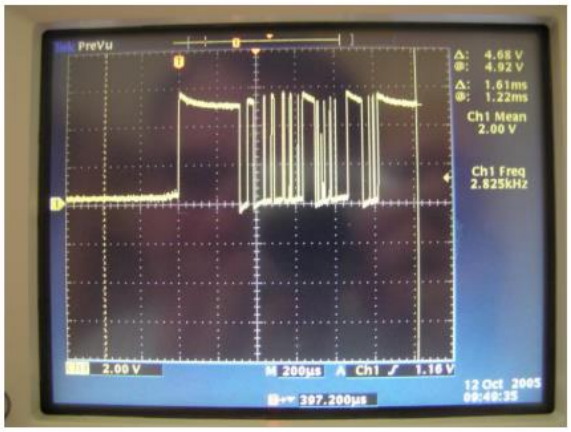
```
47      until(Sensor(IN_1) == 1); //cekani na zmacknuti dotykoveho senzoru
48      until(Sensor(IN_1) == 0); //cekani na uvolneni dotykoveho senzoru
49      OnRevSync(OUT_AB, 40, 0); //spusteni pohybu vpred
50
51      angle = 0; //nastaveni uhlu na nulu
52      flag = 0; //nastaveni vlajky na nulu
53      tickLast = CurrentTick(); //urceni aktualniho casu kostky
```

Smyčka PD regulátoru:

```
55      while(true){ //zacatek nekonecne smycky
56                  tickNow = CurrentTick(); //urceni aktualniho casu kostky
57                  speed = SensorHTGyro(IN_2); //odecteni hodntoty z gyroskopickeho senzoru
58                  angle = angle + abs(tickNow-tickLast)*(speed-offset)/1000; //vypocet uhlu
59                  current_error = angle - target_value; //regulacni odchylka
60                  derivate = current_error - last_error; //vypocet derivative
61                  turn = K_p*current_error + K_d*derivate; //vypocet akcniho zasahu
62
63                  if(turn > limitations){ //nastaveni omezeni
64                          turn = limitations; //maximalni akci nazasah
65                  }
66
67                  if(turn < -limitations){ //nastaveni omezeni
68                          turn = -limitations; //maximalni akci nazasah
69                  }
70
71                  OnFwd(OUT_C, turn); //nastaveni rychlosti motoru C a rampy
72
73                  if(Sensor(IN_1)==1 && flag > 10){ //pokud je tlacitko zmacknuto
74                       break; //zastaveni programu
75                  }
76
77                  last_error = current_error; //nastaveni last_error pro D regulator
78                  flag++; //zvyseni vlajky
79                  Wait(20); //cekani
80                  tickLast = tickNow; //pro nasledne urceni casu trvani smycky
81      }//konec nekonecne smycky
```

Na řádcích 63–69 je vidět ve dvou podmínkách omezení akčního zásahu. Na řádku 72 je pak vidět podmínka po jejímž splnění se program ukončí.

Proměnná flag byla zavedena, protože docházelo k okamžitému zastavení běhu programu. K tomuto jevu docházelo, protože tlačítka jsou nedokonalá zařízení a při rozepnutí tlačítka může dojít k jeho opětovnému krátkému stisknutí, které kostka NXC zaznamenala. Ilustrace tohoto problému je na Obr. 60.



Obr. 60 Rezonance tlačítka [23]

3.6.4.3. PyNXC

Kód vznikl jako přepis kódu NXC z předchozí kapitoly. Obdobně je kód rozdělný do samostatných sekcí, které řeší jednotlivé úkoly, potřebné k vykonání úkolu robot číšník.

Nastavení jednotlivých konstant:

```
def main():
    K_p = 7 #nastaveni velikosti proporcionalni konstanty
    K_d = 5 #nastaveni velikosti derivacni konstatnty
    target_value = 0  #hodnota, kterou chceme sledovat
    limitation = 55 #velikost omezeni akcniho zasahu
```

Inicializace senzorů:

```
    SetSensorTouch(IN_1) #inicializace dotykoveho senzoru
    SetSensorHTGyro(IN_2) #inicializace gyroskopickeho senzoru
```

Znehybnění rampy z důvodu přesnějšího měření gyroskopického senzoru:

```
    OnFwd(OUT_C, 20) #znehybneni rampy
    Wait(2000) #cekani na znehybneni rampy
    Off(OUT_C) #vypnuti pohybu rampy
```

Měření offsetu gyroskopického senzoru, který se provádí za jízdy robota po nakloněné rovině:

```
    comulative = 0 #nastaneni comulative na nulu

    Wait(5000) #cekani z duvodu ustaleni senzoru
    OnRevSync(OUT_AB, 35, 0) #spusteni jizdy robota
    Wait(2000) #cekani z duvodu ustaleni senzoru
    for i in range(0,30): #smycka se 30 opakovanimi
        comulative = comulative + SensorHTGyro(IN_2) #nacteni dat a jejijich  pricteni ke komulative
        Wait(100) #cekani
    Off(OUT_AB) #vypnuti jizdy robota
    offset = comulative/30 #vypocet offsetu
    PlaySound(SOUND_LOW_BEEP) #oznameni o ukoncneni mereni offsetu
```

Umístění rampy do výchozí polohy:

```
    RotateMotorEx(OUT_C, 50, -60, 0, false, true)  #ustaleni rampy do vodorovne polohy
    Wait(10) #cekani
    OnFwd(OUT_C, -10) #drzeni rampy ve vodorovne poloze
    Wait(50) #cekani
```

V tomto okamžiku se předpokládá umístění robota do startovního čtverečku a umístění vozíku na plošinu robota.

Naprogramované čekání na stisknutí a opětovné uvolnění tlačítka na dotykovém senzoru a následné spuštění pohybu robota:

```
    until(Sensor(IN_1) == 1) #cekani na zmacknuti dotykoveho senzoru
    until(Sensor(IN_1) == 0) #cekani na uvolneni dotykoveho senzoru
    OnRevSync(OUT_AB, 35, 0) #spusteni pohybu vpred
```

Nastavení proměnných na nulu:

```
    angle = 0 #nastaveni uhlu na nulu
    flag = 0 #nastaveni vlajky na nulu
    tickLast = CurrentTick() #urceni aktualniho casu kostky
```

Smyčka PD regulátoru:

```
39  while(True): #zacatek nekonecne smycky
40      tickNow = CurrentTick() #urceni aktualniho casu kostky
41      speed = SensorHTGyro(IN_2) #odecteni hodntoty z gyroskopickeho senzoru
42      angle = angle + abs(tickNow - tickLast) * (speed - offset) / 1000 #vypocet uhlu
43      current_error = angle - target_value #regulacni odchylka
44      derivate = current_error - last_error #vypocet derivative
45      turn = K_p * current_error + K_d * derivate #vypocet akcniho zasahu
46
47      if (turn > limitation): #nastaveni omezeni
48          turn = limitation #maximalni akci nazasah
49
50      if (turn < -limitation): #nastaveni omezeni
51          turn = -limitation #maximalni akci nazasah
52
53      OnFwd(OUT_C, turn) #nastaveni rychlosti motoru C
54
55      if (Sensor(IN_1) == 1 and flag > 10): #pokud je tlacitko zmacknuto
56          break #zastaveni programu
57
58      last_error = current_error #nastaveni last_error pro D regulator
59      flag = flag + 1 #zvyseni vlajek
60      Wait(20) #cekani
61      tickLast = tickNow #pro nasledne urceni casu trvani smycky
```

Na řádcích 47–51 je vidět ve dvou podmínkách omezení akčního zásahu. Na řádku 55 je pak vidět podmínka, po jejímž splnění se program ukončí.

Program je otestovaný na stejné dráze jako program napsaný v NXC a na této dráze se chová obdobně. Program se chová stanoveným způsobem. Z těchto důvodů prohlašuji programovací prostředí PyNXC, jako vhodné k naprogramovaní PD regulátoru. Tuto skutečnost potvrzuje i kapitola 3.4.7.4.

3.6.5. Nastavení konstant PD regulátoru

Autor dosáhl nejlepších výsledků při nastavení konstant, které jsou znázorněny v Tabulka 11.

Konstanta	Hodnota
K_p	7
K_d	5
Omezeni (limitations)	55

Tabulka 11: Nastavení konstant

Na začátku zpracování tohoto úkolu byla myšlenka, že bude stačit pro regulování pohybu plošinky vytvořit P regulátor. Tato myšlenka se ukázala jako nemožná, jelikož při nastavení proporcionální konstanty na 11, plošina nestíhala regulovat regulační odchylku, což se projevovalo zejména při jízdě robota směrem dolů po šikmé ploše. Při nastavení této konstanty na 12 se plošina naopak nepříjemně rozkmitávala. Toto rozkmitávání se podařilo eliminovat pomocí přidání derivační složky regulátoru.

Nastavení konstant PD regulátoru probíhalo pomocí Zkouškové a chybové metody. Jiný typ nastavení konstant vzhledem k zaměření této bakalářské práce nepřicházel v úvahu, jelikož musel být tak snadný, aby ho zvládli provést studenti v prvním semestru studia.

3.6.6. Závěr PD regulátoru

Nastavení úhlu otočení hřídele, tak aby byla rampa ve vodorovné poloze vzhledem k Zemi, probíhalo empirickým zjišťováním. Motory u NXT totiž mají problém s přesním nastavením úhlu natočení hřídele. Jistou nedokonalost při tom sehrál i fakt, že servomotoru chvíli trvá, než zabrzdí příkazem OnFwd(OUT_C, -10) a plošina se za tu dobu mírně propadne. Navíc se ukázalo jako vhodné plošinu více natočit, než je nezbytné k udržení vozíku, protože při přechodu k regulaci polohy plošiny pomocí regulátoru dochází k mírnému propadu plošiny. Bohužel se v průběhu testování ukázalo, že úhel natočení plošinky závisí i na stavu nabytí akumulátoru.

Po prvotních neúspěších s regulací plošiny pomocí PD jsem se rozhodl vyzkoušet více stavový regulátor. Tento regulátor se ale bohužel nepodařilo úspěšně implementovat v jazyce NXC. Po tomto neúspěchu jsem se vrátil k PD regulaci. Přepracoval jsem konstrukci robota, tak abych umístil jeho těžiště blíže k povrchu Země a robot se stabilizoval, čímž se zlepšila přesnost gyroskopického senzoru. Původní neúspěšná konstrukce robota vymodelovaná v Lego Digital Designer je k vidění na Obr. 61.

Obr. 61 Původní robot k řešení úlohy robot číšník

Jedním z problémů, proč se nedaří dosáhnout lepší regulace a tím rychlejší jízdy robota, je nedokonalý gyroskopický senzor, pomocí něhož měříme úhel natočení plošiny. Přestože se reguluje chyba offsetu na začátku běhu programu, tato regulace není dokonalá. Změřený úhel natočení plošiny pomocí gyroskopického senzoru, přestože se gyroskopický senzor nepohybuje, se neustále zvětšuje. Toto načítání hodnot způsobí, že na konci pohybu je plošina až o 10° natočená jinak, než byla na začátku pohybu. Autor pozoroval natočení plošiny a téměř po celou dobu pohybu robota se nacházela podle gyroskopického senzoru v úrovni -2° až + 2° oproti počáteční situaci, a přesto z ní vozík sjel, protože reálné natočení plošiny bylo větší. Toto nedokonalost se podařila částečně odstranit pomalejší jízdou robota a částečně stabilnějším umístěním rampy na podvozek robota v druhém konstrukčním návrhu, který je popsán v kapitole 3.6.3.

Druhý problém je nemožnost nastavení zrychlení při pohybu motoru. Motor tedy zrychlí téměř okamžitě na maximální rychlost a pohyb plošiny je trhaný. Díky tomuto trhání se vozík na plošině pohybuje a dochází k jeho sjetí. Tomuto problému částečně pomohlo zabránit omezení akčního zásahu, avšak takto skutečnost znemožňuje zrychlit pohyb robota po dráze.

Přes tyto zmíněné problémy se podařilo robotovi převést bezpečně vozík přes celou soutěžní dráhu. Toto převezení se na následném testování ukázalo jako stabilní, jelikož robot bezpečně převezl vozík z jedné strany soutěžní dráhy na druhou stranu pětkrát z pěti soutěžních pokusů.

3.6.7. Vícestavová regulace

Jako jeden ze způsobů regulace natočení rampy jsem se rozhodl vyzkoušet vícestavovou regulaci. Tuto regulaci se ovšem nepodařilo napsat v žádném programovacím jazyce. Jako největší problém se ukázal již zmíněný gyroskopický senzor. Úhel, naměřený pomocí tohoto senzoru, měl sloužit pro rozdělní jednotlivých stupňů regulace. Myšlenka tohoto regulátoru byla jednoduchá. Pomocí gyroskopického senzoru měřit úhel natočení robota a tím určit, zda jede nahoru nebo dolů po nakloněné rovině nebo zda jede vodorovně s povrchem Země. Podle těchto skutečností jsem následně chtěl upravovat natočení rampy.

Původně jsem měl napsaný 5stupňový regulátor popsaný v Tabulka 12. Ale toto rozdělení se ukázalo, jako příliš nerobustní vzhledem k přesnosti měření úhlu natočení robota.

Natočení robota	Natočení plošiny
< -10°	-15°
$< -10°, -5°)$	-8°
$< -5°, 5° >$	0°
$(5°, 10° >$	8°
> 10°	15°

Tabulka 12: 5stupňový regulátor

Z tohoto důvodu jsem rozhodnul použít 3stupňový regulátor popsaný v Tabulka 13. Problém tohoto regulátoru byl, že nestíhal regulovat polohu plošiny, tak aby vozík nesjel z plošiny.

Natočení robota	Natočení plošiny
< -8°	-15°
$< -8°, 8° >$	0°
> 8°	15°

Tabulka 13: 3stupňový regulátor

Jako poslední jsem se rozhodl, že zkusím napsat chování plošiny v jednotlivých stavech regulátoru z Tabulka 13, za tímto účelem jsem snížil hranici natočení robota, při kterém se rampa nacházela ve vodorovné pozici vzhledem k povrchu Země. Tento způsob regulace natočení plošiny se ukázal jako nejspolehlivější. Robot pomocí něho dokázal projet první stoupání, ovšem při opětovném návratu na vodorovnou plochu již vozík z rampy sjel. Bylo to způsobeno částečně nestíháním správného polohování plošiny, částečně pozdním spuštěním polohování plošiny způsobeným nepřesným měřením úhlu. Bohužel změny hranic pro regulaci nevedli ke kýženému výsledku.

3.6.8. Závěr

V tomto úkolu bylo za úkol projet po dráze, která obsahovala stoupání i klesání, s vozíkem na rampě tak, aby tento z rampy nesjel. Rampa i vozík byly sestaveny podle zadání. Takto sestavenou rampu bylo nejprve třeba umístit na sestaveného robota a následně s ním projet vytvořenou dráhu. V průběhu řešení jsem vystřídal několik návrhů robotů a umístění rampy, ale nakonec se ukázalo, že nejlepších výsledků dosahuji s robotem, který má těžiště blízko země. Při této konfiguraci těžiště se robot a tím pádem i gyroskopický senzor nejméně otřásal a dosahoval větší přesnosti než v případě, kdy měl robot těžiště umístěné dále od země.

Obecně největší problém v tomto úkolu je nedokonalý gyroskopický senzor a určení stupně jeho natočení. Pomocí tohoto senzoru jsem určoval natočení rampy a následně toto natočení pomocí připojeného servomotoru reguloval.

Postupně jsem vyzkoušel P regulátor, který se ale ukázal jako nevhodný, jelikož nestíhal regulovat náhlé změny natočení rampy. Jako další regulátor jsem vyzkoušel PD regulátor, který jsem i popsal jako jediné funkční řešení této úkoly. Jako poslední regulátor jsem vyzkoušel vícestavový regulátor, který bohužel nefungoval a nestíhal regulovat natočení plošiny, tak aby z ní vozík nesjel.

Kód PD regulátoru napsaného v jazyce NXC jsem přepsal do jazyka PyNXC. Po odzkoušení takto přeloženého kódu na dráze jsem nenašel problém, který by znemožnil projetí dráhy a kód se choval dle očekávání, která jsem získal díky funkčnosti kódu z NXC. Z tohoto důvodu hodnotím jazyk PyNXC jako vhodný k řešení této úlohy.

4. Problémy, se kterými se autor práce setkal při řešení úloh ze sborníku úkolů

Během řešení úkolů ze Sborníku jsem narazil na řadu problémů. Některé problémy jsem se rozhodl v této kapitole zveřejnit a popsat.

4.1. Nemožnost napsání všech kódů v NXT-G

Jedním z největších problémů, na který jsem během řešení této práce narazil, byla obtížnost programování v prostředí NXT-G. Toto prostředí je vhodné pro výuku a základní pochopení programování, především na základních školách. Ovšem pro tvorbu větších projektů je nevhodné a ty se v něm stávají nepřehledné. Proto jsem se snažil veškeré projekty psané v tomto jazyku co nejvíce zjednodušit. Neprogramoval jsem do nich tedy části, které nejsou nezbytné pro splnění úlohy. Příkladem může být anti-windup u projektu se sledováním černé čáry.

Jiné projekty v tomto prostředí ani nemohli vzniknout, například prohledávání bludiště pomocí DFS. Jedním z důvodů, proč jsem rozhodl tento projekt v tomto prostředí nenapsat je, že se zde nedají vytvářet pole, která jsou pro tento projekt nezbytná.

Původně jsem si myslel, že alespoň některé nedokonalosti jazyka NXT-G odstraní jeho novější verze EV3-G. Bohužel tato teorie se nepotkala s praxí. S EV3-G sice přišla pokročilejší možnost matematiky, ale většina z nových operací se nedá na kostku NXT převést.

4.2. Problémy spojené s nedokonalostí ultrazvukového senzoru

Asi největší problém, na který jsem v průběhu tvorby této práce, byl nedokonalý ultrazvukový senzor a měření vzdáleností. Podle výrobce společnosti Lego má tento senzor měřit až do vzdálenosti 2,5 metru. [24] V praxi se mi ovšem pomocí ultrazvukového senzoru nepovedlo naměřit více než 170 centimetrů. Ilustrace měření je vidět na Obr. 62 a Obr. 63.

Obr. 62 Ilustrace nemožnosti naměřit vzdálenost pomocí ultrazvukového senzoru

Obr. 63 Největšá naměřená vzdálenost pomocí tohoto senzoru

Hodnoty 162 centimetrů jsem dosáhl za předpokladu, že byl robot absolutně v klidu. Jak se s ním hnulo, ať už dopředu, dozadu, nebo do strany na displeji se objevily otazníky, které indikují, že robot nemohl vzdálenost naměřit. Obdobných problémů jsem dosahoval s robotem, když jsem se snažil měřit vzdálenost objektů za jízdy. Rovněž jsem testoval přesnost tohoto senzoru. Pokud se robot nepohyboval, dosahoval jsem tabulkové přesnosti $\pm$ 3 centimetry. [24] Problém však nastal při pohybu robota, měření jsem prováděl již v úkolu objíždění překážky, kdy se robot zastavoval ve vzdálenosti 10 centimetrů od předmětu místo naprogramovatelných 20 centimetrů. Jsem si vědom, že zde vznikne drobné přejetí z důvodu, že robot nezastaví v okamžiku, kdy se na senzoru naměří 20 centimetrů, a motory ještě nějakou dobu běží a uvádí robota do pohybu. Avšak nemyslím si, že by tato skutečnost způsobila rozdíl 7 centimetrů.

Původně jsem chtěl úlohu objíždění napsat tak, že by si sám robot kontroloval, zda danou překážku již objel a může se vrátit na soutěžní dráhu. Bohužel se tato možnost neukázala jako spolehlivá, jelikož ultrazvukový senzor neměří vzdálenost dostatečně věrohodně. Při pohybu robota a současném měření dochází ve většině případů k naměření hodnoty vzdálenosti nekonečno. Tuto skutečnost jsem se snažil napravit i pomocí klouzavého průměru, ovšem to vzhledem k množství chybějících dat nevedlo k úspěšnému měření.

4.3. Nastavení konstant u PID regulátoru

Jako jeden z dalších problémů se ukázalo nastavování konstant u PID a PD regulátorů. Konstanty byly nastaveny v úloze na sledování černé čáry. Toto nastavení bylo provedeno pomocí metody Ziegler-Nichols. Samotné nastavování konstant dle této metody je časově náročné, ovšem časově náročnější se ukázalo empirické odladění těchto konstant, aby bylo možno na dráze dosáhnout co nejlepšího výsledku. Metoda Ziegler-Nichols totiž nedává přesné výsledky, ale dává pouze odhad velikosti těchto konstant [21]. V této práci jsem byl odkázán pouze na tzv. empirické metody. [25] Použití jiných, než empirický metod nastavování konstant regulátorů nepřipadala v úvahu vzhledem ke složitosti těchto metod a zaměření této práce.

Co mě jako autora této práce překvapilo, byla nutnost nového odladění konstant regulátoru u úlohy objíždění překážky. Konstanty regulátoru pro sledování černé čáry byly nastaveny již v úloze na

sledování černé čáry. K tomuto původnímu regulátoru přibyla druhá větev kódu, která měla za úkol hlídat vzdálenost od překážky a případně tuto překážku objet. Předpokládal jsem, že bude třeba konstanty tako napsaného regulátoru poopravit oproti předchozímu úkolu, aby bylo možno na dráze dosáhnout lepšího výsledku. Co jsem ale nepředpokládal, že bude potřeba toto odvození provést od začátku, tedy včetně nového změření parametrů pro výpočet konstant regulátorů dle metody Ziegler-Nichols. Překvapilo mě, jak se od sebe jednotlivé konstanty pro regulátor z úlohy objíždění překážky a regulátor z úlohy sledování černé čáry liší, někdy až 7násobně. Tato změna nastala z důvodu, jelikož nová smyčka PID regulátoru trvala déle než smyčka u PID regulátoru, který pouze sledoval čáru. K tomuto výraznému prodloužení došlo v důsledku měření hodnoty pomocí ultrazvukového senzoru. Tato procedura trvá 15 ms a výrazně tak prodlužuje trvání smyčky.

4.4. Složitost kódu u PID regulátoru při objíždění překážky

Kvůli složitosti a délce kódu, který vznikl pro úkol objíždění překážky, jsem se rozhodl tento kód upravit a popsat tento problém nikoliv na PID regulátoru, ale na cik-cak regulátoru. Touto úpravou jsem dosáhl srozumitelnějšího a kratšího kódu, který by měl stejně dobře demonstrovat náš problém. Pro studenty by pak již neměl být problém nahradit v kódu cik-cak regulátor za PD případně PID regulátor.

4.4.1. Kód napsaný v jazyce NXC určený pro úkol objíždění překážky společně s PID regulátorem

```
1 //konstanty PID
2 float K_p, K_i, K_d;
3
4 //konstanty
5 int target_value, speed_offset;
6
7 //pomocne promene
8 float integral, last_error, derivate, light_value, current_error, turn;
9 int speed_A, speed_C;
10
11 int distance;
12
13 //hlavni cast
14 task main(){
15     K_p = 1; //nastaveni velikosti proporcionalni konstanty
16     K_i = 0; //nastaveni velikosti integralni konstanty
17     K_d = 0; //nastaveni velikosti derivacni konstanty
18
19     target_value = 47; //hodnota, kterou chceme sledovat
20
21     speed_offset = 25; //offset rychlosti
22
23     integral = 0; //kumulativni error pro potřeby I regulátoru
24
25     distance = 30;
26
27     SetSensorLight (IN_3); //inicializace senzoru světla na poru 3
28     SetSensorLowspeed(IN_4);
29
```

```
30      while(true){ //zacatek nekonecne smycky
31                   distance = SensorUS(IN_4);
32                   if(distance < 20){
33                   Off(OUT_AC); //vypnu oba motory
34                   Wait(100); //pockam
35                          OnFwd(OUT_C, 75);
36                          Wait(700);
37                          OnFwdSync(OUT_AC, 30, -10);
38                          while(Sensor(IN_3) > 50){
39                                                  Wait(10);
40                          }
41                          while(Sensor(IN_3) < 50){
42                                                  Wait(5);
43                          }
44                          Off(OUT_AC); //vypnu oba motory
45                          Wait(100); //pockam
46                          OnFwd(OUT_C, 25);
47                          while(Sensor(IN_3) > 50){
48                                                  Wait(10);
49                          }
50                   }else{
51                  light_value = Sensor(IN_3); //odecteni hodnoty ze svetelneho senzoru
52                  current_error = light_value - target_value; //regulacni odchylka
53                  integral = integral + current_error; //vypocet integral
54                  derivate = current_error - last_error; //vypocet derivative
55                  turn = K_p*current_error + K_i*integral + K_d*derivate; //vypocet akcniho zasahu
56
57                  turn = -turn;
58
```

```
59                  speed_C = speed_offset - turn; //spocteni rychlosti motoru C
60                  speed_A = speed_offset + turn; //spocteni rychlosti motoru A
61
62                  //anti windup
63                  if(speed_C > 100){ //když by se motor C měl otáčet dopředu rychleji než může
64                              speed_C = 100; //nastaveni maximální rychlosti na motru
65                              integral = 0; //nulovani integral
66                  }
67
68                  if(speed_A > 100){ //když by se motor A měl otáčet dopředu rychleji než může
69                              speed_A = 100; //nastaveni maximální rychlosti na motru
70                              integral = 0; //nulovani integral
71                  }
72
73                  if(speed_C < -100){ //když by se motor C měl otáčet dozadu rychleji než může
74                              speed_C = -100; //nastaveni maximální rychlosti na motru
75                              integral = 0; //nulovani integral
76                  }
77
78                  if(speed_A < -100){ //když by se motor A měl otáčet dozadu rychleji než může
79                              speed_A = -100; //nastaveni maximální rychlosti na motru
80                              integral = 0; //nulovani integral
81                  }
82                  //konec anit-windup
83
84                  OnFwd(OUT_C, speed_C); //nastaveni rychlosti motoru C
85                  OnFwd(OUT_A, speed_A); //nastaveni rychlosti motoru A
86                  last_error = current_error; //ulozeni soucasne regulacni odchylky pro D regulato
87                  }
88                  Wait(5); //pockej
89      } //konec nekonecne smycky
90 } //konec programu
```

4.5. Nemožnost vytváření 2D polí v NXC

Dalším z problémů, na který jsem narazil při tvorbě této bakalářské práce, byla absence 2D polí v programovacím jazyce NXC. O možnosti použití 2D polí v tomto jazyce informuje mimo jiné i oficiální stránka NXC [19], nebo například [26], ale i stránky některých univerzit. 2D pole se sice v tomto jazyce dají inicializovat a bez problému se zkompilují a nahrají do NXT kostky, ale nedá se přistupovat k jednotlivým prvkům v tomto poli. Tento problém se dá částečně obejít pomocí následujícího způsobu deklarace

int array[][] = {{1, 2, 3}, {4, 5, 6}};

, který je ovšem nevhodný pro větší pole.

4.6. Závěr této kapitoly

Jak tomu bývá zvykem, nic se neobjede bez problémů, proto jsem řadu problémů, na které jsem narazil, popsal během této kapitoly. Tímto popisem jsem chtěl pomoci budoucím studentům, aby neopakovali stejné chyby, které jsem udělal, nebo aby se na ně naopak lépe připravili.

5. Změření času nejlepších průjezdů jednotlivých drah

Tato kapitola slouží jako podpůrný materiál pro tvorbu bodového systému pro předmět roboti.

Pro měření nerychlejšího času projetí stanovených drah s vyznačenou černou čárou jsem si zvolil robota, který je popsaný v kapitole 3.6.3. A za programovací jazyk jsem vybral NXC. Z hlediska časové náročnosti na odlaďování regulátorů jsem zvolil PD regulátor. PD regulátor je rovněž doporučovaný regulátor pro sledování černé čáry.

Jako metodu pro odladění jednotlivých konstant jsem Zvolil zkouškovou a chybovou metodu. Pomocí této metody jsem došel k následujícím hodnotám nastavení konstant.

$$K_p = 1{,}5, \tag{19}$$

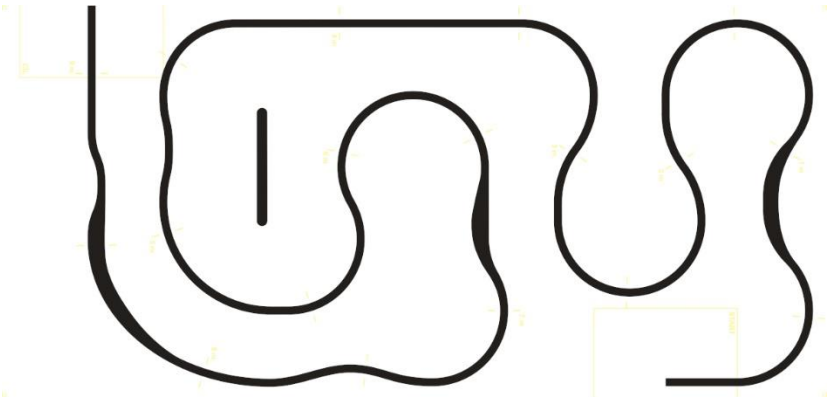
$$K_d = 12, \tag{20}$$

$$speed_{offset} = 60. \tag{21}$$

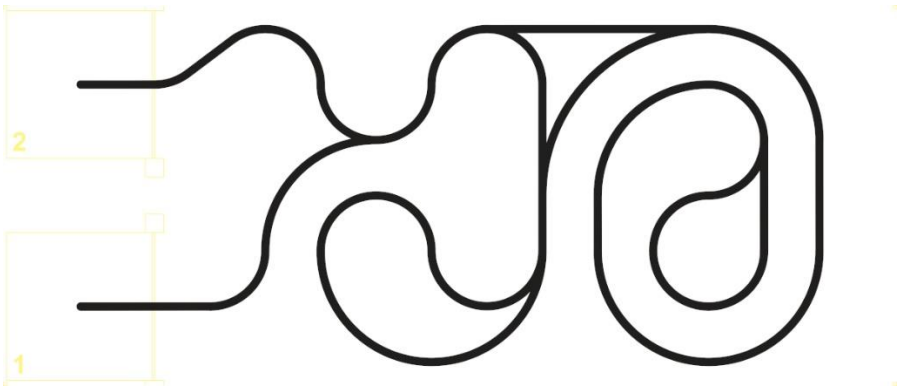
Rovněž je třeba podotknout, že oproti původnímu úkolu došlo ke změně target_value na 55. Jinak kód zůstal stejný, jako je popsaný v úkolu 0. Toto nastavení konstant se ukázalo jako nejlepší nastavení konstant pro všechny tři typy měřených čar.

Pokusy na jednotlivých čarách ukázali, že zvyšováním offsetu rychlosti se již rychlejšího průběhu nedá dosáhnout. Hlavní důvod, proč tento offset nejde zvýšit, je ten, že motory naráží na oblast saturace především při zatáčkách s nízkými poloměry zakřivení.

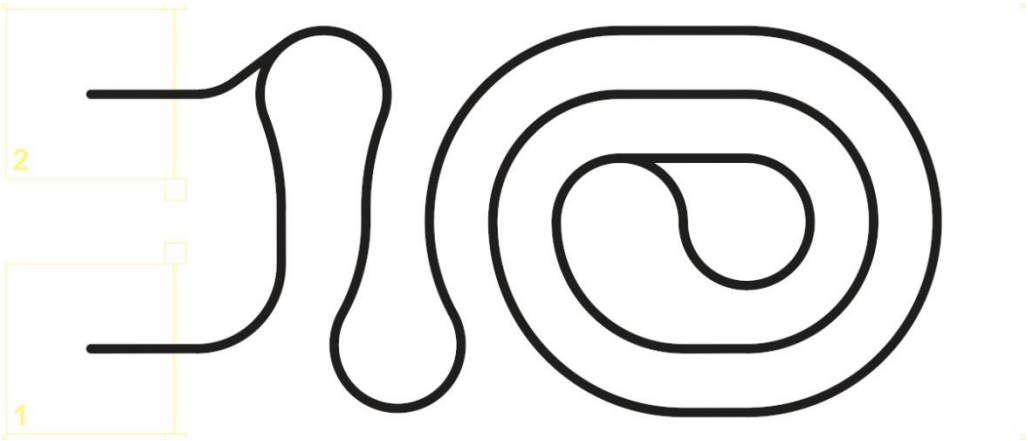
Přehled jednotlivých typů drah je k vidění na Obr. 64, Obr. 65 a Obr. 66.



Obr. 64 Černá čára I.



Obr. 65 Černá čára II.



Obr. 66 Černá čára III.

Přehled nejlepších časů dosažených pomocí PD regulátorů je k nalezení v Tabulka 14.

Typ čáry	**Nejlepší dosažený čas**
Černá čára I.	27,34 sekund
Černá čára II.	43,47 sekund
Černá čára III.	60,54 sekund

Tabulka 14: Přehled dosažených časů

Následně jsem vyzkoušel odladit PID regulátor. Ukázalo se, že odladěný PID regulátor zadané trasy projede v průměru o zhruba 1 sekundu rychleji. Toto rychlejší projetí je dáno plynulejší jízdou robota především na rovných úsecích dráhy.

6. Úprava a popis jazyka PyNXC

Programovat kostku NXT nelze v Pythonu žádným oficiálním softwarem, který by vyvíjela společnost Lego. Nicméně podobně, jako NXC, tedy do jisté míry fanouškovský projekt, existuje i PyNXC. PyNXC si bere příklad v jazyce Python, ale je značně ovlivněný jazykem NXC. Tento jazyk svou syntaxí odpovídá Pythonu až na drobné výjimky, které v následující kapitole popíšu. Většinu funkcí pak přebírá po jazyce NXC. Přebírá například příkazy, pomocí kterých se ovládají motory, nebo se načítají hodnoty ze senzorů.

Jazyk PyNXC vlastně není programovací jazyk sám o sobě. Jedná se spíš o kompilátor, který převádí kód z jazyka Pythonu do jazyka NXC. Jak jeho autor sám uvádí, nejde pouze o kompilátor mezi jazykem Python a jazykem C, jak by se mohlo na první pohled zdát. Ale hardware a firmware kostky NXT tento

úkol činí obtížnějším. [14] Současné verze, která je volně ke stažení z Google Code Archive (https://code.google.com/archive/p/pynxc/source/default/source) obsahuje několik chyb a nedokonalostí, které znemožňují použití tohoto jazyka pro širší veřejnost. Proto bych tento kompilátor rád poupravil, aby práce s ním byla přehlednější a uživatelsky přívětivější než současná verze. Dále bych rád opravil alespoň některé chyby, které znemožňují s tímto jazykem vytvářet komplexnější projekty.

6.1. Jiné možnosti programování NXT kostky v jazyce Python

Mimo PyNXC existuje i projekt NXT-Python, který se bohužel ukázal v průběhu testování jako nefunkční. Druhá možnost je podle wikipedie použití Ironpythonu v prostředí Microsoft Robotics Developer Studio. Bohužel tato informace postrádá na této stránce citaci a autorovi této práce se nepodařilo dohledat jiný zdroj, který by tuto skutečnost potvrdil. [13]

6.2. Oprava některých chyb

V této kapitole se podíváme na opravu největších chyb a nedokonalostí v jazyce PyNXC, které znemožňují jeho použití pro komplikovanější projekty.

6.2.1. Oprava vkládání chybných definic na začátek překládaného souboru

První chybu, kterou se mi podařilo opravit, je vkládání definic na začátek přeloženého souboru. Bohužel se zde vkládali definice a makra k usnadnění programování jednotlivých senzorů, které ale neměli vliv na samotnou funkci přeloženého kódu. Problém zde byl, že kompilátor pro jazyk NXC tyto definice nemohl rozpoznat a při nahrávání programu do kostky NXT hlásil chybu.

Oprava této chyby byla jednoduchá. Po projití kódu jsem si všiml, že se definice vkládají do nově vytvořeného souboru ze souboru defs.h. Původně jsem se snažil chyby v těchto definicích opravit, ale následně jsem se rozhodl, že je na začátek přeloženého kódu nebudu vkládat. Tyto definice nemají vliv na běh programu v kostce NXT.

6.2.2. Oprava špatné inicializace proměnných a funkcí

V původní verzi PyNXC nefungoval systém deklarace proměnných při převodu do jazyka NXC. Tuto nedokonalost jsem opravil přepsáním stromu pro rozhodování, o jaký typ proměnných se jedná, a jejich novou deklaraci v jazyce NXC. Bohužel se mi nepodařilo implementovat deklaraci pro textové řetězce a mutexi.

Obdobný problém pak nastává v případě deklarace jednotlivých funkcí. V jazyce NXC musí hlavička obsahovat návratový typ proměnné. Původně se všechny funkce, které navraceli určitou hodnotu, deklarovali jako integer. Přesnější deklarace jednotlivých funkcí by vyžadovala zásah do rozhodovacího stromu, ke kterému bohužel nemám přístup. Z toho důvodu, jsem se rozhodl, že všechny funkce se budou deklarovat jako float. Tato změna by neměla mít vliv na chování programů a při testování úkolů ze sborníku jsem nenarazil na problém spojený s touto novou deklarací.

6.2.3. Oprava chování smyčky for

Původní překlad for smyčky z PyNXT do jazyka NXC nefungoval. Zde byl problém s deklarací proměnné pro počítání počtu opakovaní smyčky. Původně se smyčka for i in range(0, 20), překládala jako for(i=0;i<20;i++). Tato část kódu, ale hlásila chybu, jelikož interpret NXC nevěděl, jakého je proměnná i typu. Tuto skutečnost, jsem opravil, tak že se smyčka for nyní překládá následovně for(int

i=0;i<20;i++). Interpretr NXC nyní již nehlásí chybu, protože ví, o jaký typ proměnné se jedná a jak s ním zacházet.

6.3. Dopsání uživatelského rozhraní Sekretar pro práci s prostředím PyNXC

Jelikož původní dva způsoby překladu souborů napsaných v PyNXC do jazyka NXC jsou značně neintuitivní, rozhodl jsem se dopsat grafické uživatelské rozhraní, které by zjednodušilo práci s tímto prostředím. Při psaní tohoto prostředí jsem chtěl být věrný původní vizi autora tohoto kompilátoru Brian Blais [14].

Jedním z mých požadavků bylo, aby prostředí Sekretar bylo spustitelné z klasického Pythovského IDLE pro verzi 2.7.x bez potřeby doinstalování knihoven třetích stran. Tento požadavek se mi nepodařilo splnit, jelikož jsem potřeboval způsob, jak stisknout klávesy na klávesnici pomocí Pythonu. Toho stisknutí kláves jsem dosáhl pomocí knihovny pyautogui. Druhý požadavek byl, aby bylo prostředí co nejvíce intuitivní a uživatelsky co nejpřívětivější.

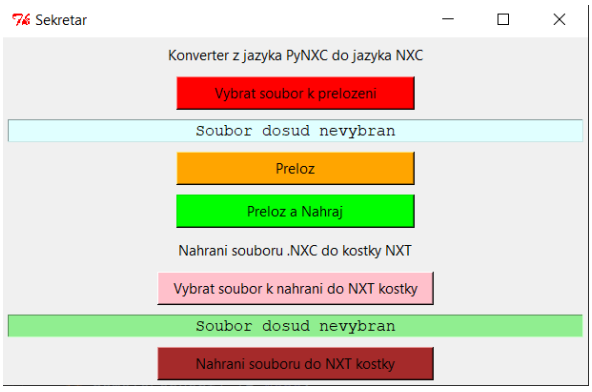
Z důvodu nemožnosti doinstalování knihoven jsem neměl jinou možnost než použít knihovnu Tkinter, která je již v základní instalaci jazyka Python 2.7.x. Tato knihovna není vhodná pro tvorbu graficky vysoce atraktivních prostředí, ale pro účely jednouchého widgetu je více než vhodný. Pomocí této knihovny jsem vytvořil samotné okno widgetu Sekretar. Naprogramoval do něj tlačítka, kterým jsem přiřadil funkce k překladu a nahrání souborů. Zároveň jsem zde implementoval dvě textové pole, do kterých se zapisují cesty k vybraným souborům.

Knihovna Tkinter za mě mimo jiné obstarala funkci vyber soubor. Takto jsem tedy měl jednoduchou možnost, jak přistupovat ke kterémukoliv souboru v počítači, respektive jak získat adresu jeho umístění. V kódu pro prostředí PyNXC se nachází již předpřipravená funkce python_to_nxc() pro překlad souborů .py do souborů .nxc. Tuto funkci jsem pouze vizuálně upravil a použil pro Sekretar.

Největší nepříjemnost nastala s nahráváním souborů .nxc do kostky NXT. Jelikož jsem chtěl být věrný původní vizi autora, a navíc jsem neobjevil jinou možnost, jak nahrát soubory do NXT kostky než pomocí prostředí BricxCC, musel jsem zautomatizovat tento proces.

Po otevření souboru .nxc v prostředí BricxCC je potřeba spojit počítač s kostkou NXT a následně zmáčknout tlačítko pro nahrání souboru do NXT kostky a zavřít prostředí BricxCC. Tohoto jsem dosáhl pomocí postupného stisknutí několika klávesových zkratek na klávesnici pomocí Pythonu a jeho knihovny pyautogui.

Grafický vzhled Sekretare jsem zvolil barevný a lineární, aby se v něm bylo co možná nejjednodušší vyznat. Foto tohoto grafického rozhraní je na Obr. 67.



Obr. 67 Ilustrace grafického prostředí Sekretar

Popis jednotlivých tlačítek a textových boxů je k nalezení v kapitole 2.3.3.

Jako největší nedostatek zhodnocuji nemožnost kontroly, zda se soubor nahrál na kostku NXT. Druhým nedostatkem je pak spouštění grafického prostředí BricxCC. Bohužel se nepodařilo nalézt funkční knihovnu pro Python 2.7.x, která by spouštěla BricxCC pouze na pozadí počítače. Přes tyto dva nedostatky si myslím, že widget Sekretar zjednoduší práci s programovacím prostředím PyNXC, jelikož se s ním soubory snáze překládají a nahrávají do kostky NXT, než v případě původních způsobů překladu. Tento widget mimo jiné upozorní na případné syntaktické chyby v souboru .py, které znemožňují jeho správné přeložení. Tuto funkci nemají všechny původní způsoby překladu.

6.4. Závěr této kapitoly

V této kapitole jsem nastínil možnosti programování kostky NXT pomocí Pythonu. Jako nejvíce propracované a zároveň funkční řešení jsem si k bližšímu prozkoumání vybral PyNXC, které i tak obsahovalo řadu chyb. Jelikož se dá kód podle licence libovolně upravovat, rozhodl jsem se jeho největší chyby opravit. Zároveň jsem k tomuto kódu dopsal uživatelské rozhraní, které zjednodušilo proces nahrávání jednotlivých programů .py do NXT kostky. Přes veškerou snahu se mi, ale bohužel do tohoto prostředí nepodařilo zcela implementovat pole. Tuto skutečnost hodnotím, jako největší nedostatek toho programovacího prostředí

Závěr bakalářské práce

V této bakalářské práci jsme měli možnost se seznámit se stavebnicí Lego Mindstorms NXT a dále jsem zde navrhl a otestoval několik úkolů pro předmět ROBOTI vyučovaný na Českém vysokém učení technickém. Tento předmět je povinný pro bakalářský studijní program Kybernetika a robotika na fakultě elektrotechnické.

Během tvorby této práce jsem vycházel ze tří programovacích jazyků NXT-G, NXC a PyNXC. Právě začlenění PyNXC do výuky by mohlo vést k lepšímu porozumění jednotlivých problémů a úkolů, se kterými se studenti setkávají v průběhu tohoto předmětu. PyNXC vychází z programovacího jazyka Python, který se vyučuje v paralelně běžícím předmětu Algoritmy a programování. Z tohoto důvodu se domnívám, že je pro studenty vhodnější než jazyk NXC, který vychází z jazyka C. Netřeba zmiňovat, že jazyk Python je jazyk vysoké úrovně a odpadají zde problémy s deklarací proměnných. Středobodem této práce je tvorba sborníku úkolů, který dává jednotlivé úkoly do širší souvislosti.

Nejprve byla v úvodní části práce představena stručná historie výukové sady Lego Mindstorms a její využití ve světě. Zde musel být pouze zkonstatován již známý fakt, že dochází k problému s nevhodností Lego Mindstorms Prime Spike pro výuku na vysokých školách. Bohužel poslední vhodná verze byla Lego Mindstorms EV3, která se však již nevyrábí. Proto řada vysokých škol po celém světě řeší problém s nedostatkem těchto stavebnic.

V následující části jsem představil jednotlivé senzory a aktuátory. Jediný senzor, kterému jsem věnoval větší pozornost, byl gyroskopický senzor. Tento senzor je zde popsán ve 3 programovacích jazycích a je zde popsána regulace jeho offsetu i způsob měření úhlu natočení tohoto senzoru vůči jeho poloze na začátku měření. Zbylé senzory a aktuátory jsou zde pouze obecně popsány a je zde popsána pouze práce s těmito senzory v programovacím prostředí PyNXC.

Součástí práce je Sborník úkolů, který představil pět úkolů, které vytváří pomyslný průřez studiem. Za první úkol bylo zvoleno Sumo, které má studenty seznámit s programovací sadou NXT a jednotlivými programovacími prostředími, včetně práce s několika senzory. Druhý úkol je Bludiště, který má studenty seznámit se základy orientace v prostoru. Třetí úkol je Sledování černé čáry, které seznamuje studenty s problematikou regulátorů. Předposledním úkolem je sledování černé čáry s objížděním

překážek, které dává základní přehled o přepínání mezi různými způsoby regulací. Poslední úkol je Robo číšník. V tomto úkolu má student za cíl převézt vozítko na plošině po nerovné dráze. Musí tedy vymyslet způsob uchycení a natáčená rampy. Tato úloha předvádí složitější způsob regulace.

Úlohy ze sborníku jsou vysvětleny pomocí pseudokódů a následně jsou naprogramovány v NXT-G, NXC a PyNXC. Jistou výjimku tvoří úloha Bludiště, kde se ukázkový kód v jazyce NXC pro algoritmus DFS nepodařilo převézt do PyNXC a NXT-G. Problém těchto prostředí je v tom, že neumí zacházet s poli. U úlohy Robo číšník jsem taktéž vynechal naprogramování v NXT-G, jelikož program je velice složitý a nepřehledný. K vybraným úlohám vznikl i manuál pro stavbu robotů. Tyto manuály jsou k nalezení v digitální příloze. Videa s řešením jednotlivých úkolů lze najít na webové stránce https://www.youtube.com/playlist?list=PLFHduE4trgzjOVngcCrtRrZY9En8s_sWl ({5}).

Po otestování většiny programů napsaných v PYNXC, jsem dospěl k závěru, že se všechny kódy chovají stanoveným způsobem. Proto hodnotím jazyk PyNXC, jako vhodný pro úlohy, které neuvažují práci s poli. Jednotlivé úkoly na regulace jsem odlaďoval pomocí empirických metod. Použil jsem především metodu Ziegler-Nichols nebo Zkouškovou a Chybovou metodu.

Při řešení úkolů ze sborníku jsem narazil na řadu problémů. Tyto problémy jsem se rozhodl popsat v jedné z kapitol, aby se jim případní budoucí studenti mohli vyhnout. Jeden z největších problémů byl nedokonalý ultrazvukový senzor pro měření vzdáleností. Další problém je měření úhlu natočení pomocí gyroskopického senzoru, kdy se nepodařilo offset zcela vykompenzovat a senzor tak načítal hodnoty i když se nepohyboval.

Během tvorby této práce bylo sledování černé čáry otestováno na několika dráhách a výsledné časy průjezdu těchto drah pomocí PD regulátoru jsou zaznamenány v samostatné kapitole. Tyto časy budou sloužit jako podklad pro tvorbu bodového systému předmětu roboti.

Poslední část této práce je věnovaná programování stavebnice Lego Mindstorms NXT pomocí Pythonu. Z možností, které jsem nalezl, jsem vybral prostředí PyNXC, protože bylo jako jediné alespoň částečně funkční. Během práce jsem opravil některé jeho nedostatky a zároveň jsem k němu dopsal grafické uživatelské rozhraní. Toto prostředí slouží k překladu souborů .py do .nxc a nahrávání souborů do kostky NXT. Pomocí tohoto prostředí a opravy některých chyb jsem z tohoto prostředí vytvořil prostředí, které se dle mého názoru hodí k výuce více než NXC. Především z důvodu jednoduší práce s tímto prostředím.

Citace

[1] The Brothers Brick, „The Brothers Brick," The Brothers Brick, LLC, [Online]. Available: https://www.brothers-brick.com/2020/01/14/a-history-of-lego-education-part-1-strong-foundations-feature/. [Přístup získán 25 2 2022].

[2] „LEGO company Historie," LEGO, [Online]. Available: https://www.lego.com/cs-cz/aboutus/lego-group/the-lego-group-history. [Přístup získán 25 2 2022].

[3] The Brothers Brick, LLC, „The Brothers Brick," The Brothers Brick, LLC, [Online]. Available: https://www.brothers-brick.com/2020/01/31/a-history-of-lego-education-part-2-path-to-mindstorms-feature/. [Přístup získán 25 2 2022].

[4] LEGO Group, „LEGO," LEGO Group, 12 6 2020. [Online]. Available: https://www.lego.com/en-us/aboutus/news/2020/june/lego-mindstorms-robot-inventor/. [Přístup získán 25 2 2022].

[5] T. Malik, „space.com," Future US, Inc., 14 1 2020. [Online]. Available: https://www.space.com/lego-education-spike-prime-stem-robotics.html. [Přístup získán 25 3 2022].

[6] B. J. Baťko, LEGO MINDSTORMS NXT - VYUŽITÍ PROGRAMOVACÍCH, Plzeň: ZÁPADOČESKÁ UNIVERZITA V PLZNI, 2014.

[7] Temple University, „Temple University," Temple University, [Online]. Available: https://noncredit.temple.edu/search/publicCourseSearchDetails.do?method=load&courseId=90991. [Přístup získán 28 2 2022].

[8] Univerzita Palackého v Olomouc, „Univerzita Palackého v Olomouc," Univerzita Palackého v Olomouc, 28 8 2022. [Online]. Available: https://stag.upol.cz/ects/predmet/KTE/ROB@?lang=cs&rocnik=1. [Přístup získán 28 2 2022].

[9] Technická univerzita v Liberci, „Technická univerzita v Liberci," Technická univerzita v Liberci, [Online]. Available: https://stansevedcem.tul.cz/nabidka-aktivit-pro-mladez/celorocni-technicke-kurzy/celorocni-technicke-kurzy. [Přístup získán 28 2 2022].

[10] Massachusetts Institute of Technology, „Massachusetts Institute of Technology," Massachusetts Institute of Technology, [Online]. Available: https://appinventor.mit.edu/explore/content/legomindstorms.html. [Přístup získán 28 2 2022].

[11] LEGO, „First Lego League Challenge," Lucasfilm Ltd., [Online]. Available: https://www.firstlegoleague.org/about#. [Přístup získán 28 2 2022].

[12] LEGO, „FIRST Tech Challenge," FIRST, [Online]. Available: https://www.firstinspires.org/robotics/ftc. [Přístup získán 28 2 2022].

[13] „Wikipedie," [Online]. Available: https://en.wikipedia.org/wiki/Lego_Mindstorms_NXT. [Přístup získán 15 2 2022].

[14] B. S. Blais, „digitalcommons bryant,“ [Online]. Available: https://digitalcommons.bryant.edu/cgi/viewcontent.cgi?article=1017&context=sci_jou. [Přístup získán 23 1 2022].

[15] „Google Code Archiv,“ [Online]. Available: https://code.google.com/archive/p/pynxc/. [Přístup získán 31 1 2022].

[16] Electrical4U, „Electrical4U,“ Electrical4U, 28 10 2020. [Online]. Available: https://www.electrical4u.com/what-is-servo-motor/. [Přístup získán 13 2 2022].

[17] P. PhDr. Tomáš Jakeš, „LEGO MINDSTORMS NXT,“ ZČU, 17 2 2013. [Online]. Available: https://www.lego.zcu.cz/web/moduly-systemu-lego-nxt. [Přístup získán 31 1 2022].

[18] SPŠ, SOŠ a SOU, Nové Město nad Metují, „RoboWiki,“ Moderní výuka s podporou ESF, 6 8 2013. [Online]. Available: http://robowiki.spsnome.cz/Programovani/NXC-Senzory-USonic. [Přístup získán 11 2 2022].

[19] bricxcc.sourceforge.net, „bricxcc.sourceforge.net,“ bricxcc.sourceforge.net, [Online]. Available: http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/group___low_speed_module_functions_ga7543e9f7d63dda6389db68271134f62e.html. [Přístup získán 21 3 2022].

[20] Mathworks, „Mathworks,“ The MathWorks, Inc., [Online]. Available: https://es.mathworks.com/help/supportpkg/legomindstormsev3/ref/nxtgyrosensor.html. [Přístup získán 14 3 2022].

[21] J. Sluka, „inpharmix,“ inpharmix, [Online]. Available: https://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html. [Přístup získán 20 1 2022].

[22] jf-parede.pt, „jf-parede,“ jf-parede.pt, [Online]. Available: https://cs.jf-parede.pt/what-is-pid-controller. [Přístup získán 19 2 2022].

[23] P. Ing. Richard Šusta, „https://moodle.fel.cvut.cz,“ 12 11 2020. [Online]. Available: https://moodle.fel.cvut.cz/pluginfile.php/261897/mod_resource/content/3/LSP2020pr08_FSM2_ControlUnits_V11.pdf. [Přístup získán 02 04 2022].

[24] Lego Grup, „generationrobots,“ [Online]. Available: https://www.generationrobots.com/media/Lego-Mindstorms-NXT-Education-Kit.pdf. [Přístup získán 3 3 2022].

[25] VSCHT, „VSCHT,“ [Online]. Available: https://web.vscht.cz/~cermanj/file/TR/Kapitola%2011-Metody%20návrhu%20regulačního%20obvodu.pdf. [Přístup získán 4 3 2022].

[26] J. Hansen, „NXC Version 1.2.1 r5,“ http://bricxcc.sourceforge.net, 19 2 2013. [Online]. Available: http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html. [Přístup získán 17 2 2022].

[27] The Brothers Brick, LLC, „ The Brothers Brick,“ The Brothers Brick, LLC, [Online]. Available: https://www.brothers-brick.com/2020/02/03/a-history-of-lego-education-part-3-mindstorms-over-matter-feature/. [Přístup získán 25 2 2022].

[28] D. Benedettelli, „Programming,“ 7 6 2007. [Online]. Available: http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_tutorial.pdf. [Přístup získán 10 9 2021].

[29] ROBOTSQUARE, „ROBOTSQUARE,“ ROBOTSQUARE, 2016. [Online]. Available: http://robotsquare.com/2013/07/16/ev3-nxt-compatibility/. [Přístup získán 1 3 2022].

[30] LEGO Group, „LEGO Group,“ LEGO Group, [Online]. Available: https://www.lego.com/cs-cz/themes/mindstorms/downloads. [Přístup získán 2 3 2022].

[31] S. S. A. S. Cathy Sarisky, „EV3 Lessons.com,“ 5 2017. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiduc2ntaf2AhU6IMUKHSAgDOoQFnoECBgQAQ&url=https%3A%2F%2Fev3lessons.com%2Fen%2FProgrammingLessons%2Fbeginner%2FLightNXT.pptx&usg=AOvVaw3ACQArBmA7MZRi9o9_N119. [Přístup získán 12 2 2022].

[32] P. PhDr. Tomáš Jakeš, „Robotické vzdělání LEGO mindstorms,“ ZČU, 17 2 2013. [Online]. Available: https://www.lego.zcu.cz/web/moduly-systemu-lego-nxt. [Přístup získán 19 3 2022].

[33] bricxcc, „Bricx Command Center 3.3,“ bricxcc, [Online]. Available: http://bricxcc.sourceforge.net. [Přístup získán 10 4 2022].

Odkazy

{1} https://modernroboticsinc.com/download/hitechnic-gyro-sensor-block-for-Mindstorms-nxt-software/

{2} https://moodle.fel.cvut.cz/course/view.php?id=6490

{3} https://robosoutez.fel.cvut.cz/zadani-soutezni-ulohy-robo-cisnik-0

{4} https://moodle.fel.cvut.cz/pluginfile.php/320552/mod_resource/content/1/tribot.pdf

{5} https://www.youtube.com/playlist?list=PLFHduE4trgzjOVngcCrtRrZY9En8s_sWl

Seznam obrázků

Obr. 1 Otáčení robota o určitý úhel doprava 19
Obr. 2 ilustrace trajketorie robota, při otočení o 90° doprava 19
Obr. 3 Řídící jednotka (kostka) NXT 20
Obr. 4 Pohled na propojovací kabely 20
Obr. 5 Detailní pohled na konektor 20
Obr. 6 Ukázka kódu v NXT-G 21
Obr. 7 Ukázka kódu v jazyce NXC 22
Obr. 8 Ukázka kódu v jazyce PyNXC 22

Obr. 9 Uživatelské rozhraní ... 23
Obr. 10 Foto ukázkového robota z hora ... 24
Obr. 11 Foto ukázkového robota zepředu ... 24
Obr. 12 Servomotor ... 25
Obr. 13 LCD ... 27
Obr. 14 Senzor zvuku ... 30
Obr. 15 Dotykový senzor ... 31
Obr. 16 Senzor světla ... 32
Obr. 17 Popis senzoru světla ... 32
Obr. 18 Ilustrace odrazu světla ... 32
Obr. 19 Ilustrace funkce senzoru ... 34
Obr. 20 Ultrazvukový senzor ... 34
Obr. 21 Gyroskopický senzor ... 36
Obr. 22 Znázorněný měřené osy [20] ... 36
Obr. 23 Hrací plocha pro sumo zápasy ... 44
Obr. 24 Kostka představující soupeřova robota ... 45
Obr. 25 Foto sumo robota I. ... 46
Obr. 26 Foto sumo robota II. ... 46
Obr. 27 Místa, ze kterých robot prohledává hrací plochu za použití NXT-G ... 48
Obr. 28 Místa, ze kterých robot prohledává hrací plochu za použití NXC ... 49
Obr. 29 Možné umístění startu a cíle v bludišti ... 55
Obr. 30 Fotografie možného rozmístění bludiště ... 55
Obr. 31 Konstrukce robota na průjezd bludištěm pohled I. ... 56
Obr. 32 kostrukce robota na průjezd bludištěm pohled II. ... 56
Obr. 33 Ilustrace bludiště společně se stěnou, která se sleduje ... 57
Obr. 34 Orientace robota v bludišti ... 60
Obr. 35 Mapa měst ... 67
Obr. 36 První krok ... 68
Obr. 37 Druhý krok ... 68
Obr. 38 Ilustrace zacykleni ... 68
Obr. 39 Finální graf DFS ... 68
Obr. 40 Čtverečky, na které může robot vjet ... 74
Obr. 41 Možné uspořádání soutěžní dráhy ... 78
Obr. 42 Sledovač čáry pohled I. ... 79
Obr. 43 Sledovač čáry pohled II. ... 79
Obr. 44 Zig Zag regulátor ... 79
Obr. 45 Graf teploty ... 80
Obr. 46 Pohyb robota při zig zag regulátoru ... 81
Obr. 47 Příklad regulace teploty pomocí PID ... 83
Obr. 48 PID regulátor ... 84
Obr. 49 blokové schéma PID regulátoru ... 85
Obr. 50 Proporcionální regulátor a jeho časový vývoj v okolí target value ... 85
Obr. 51 PD regulátor a jeho časový vývoj v okolí target value ... 86
Obr. 52 Vysvětleni pojmů ... 95
Obr. 53 Objíždění překážky I. ... 100
Obr. 54 Objíždění překážky II. ... 100
Obr. 55 Sledovač černé čáry společně s ultrazvukovým senzorem ... 100
Obr. 56 Plošina (rampa) pro stabilizaci vozíku ... 105

Obr. 57 Vozík, který balancuje na plošině 105
Obr. 58 Foto sloutěžní plochy 106
Obr. 59 Fotografie soutěžního robota 107
Obr. 60 Rezonance tlačítka [23] 110
Obr. 61 Původní robot k řešení úlohy robot číšník 113
Obr. 62 Ilustrace nemožnosti naměřit vzdálenost pomocí ultrazvukového senzoru 116
Obr. 63 Největšá naměřená vzdálenost pomocí tohoto senzoru 116
Obr. 64 Černá čára I. 119
Obr. 65 Černá čára II. 120
Obr. 66 Černá čára III. 120
Obr. 67 Ilustrace grafického prostředí Sekretar 122

Seznam Tabulek

Tabulka 1: Makra pro jednotlivé zvuky 28
Tabulka 2: Naměřené hodnoty jednotlivých barev 47
Tabulka 3: Legenda měst 67
Tabulka 4: Význam jednotlivých konstant 94
Tabulka 5: Rovnice výpočtu jednotlivých konstant PID regulátoru 96
Tabulka 6: Výhody a nevýhody jednosenzorového zig-zag regulátoru 97
Tabulka 7: Výhody a nevýhody dvousenzorového regulátoru 97
Tabulka 8: Výhody a nevýhody proporcionálního regulátoru 98
Tabulka 9: Výhody a nevýhody PD regulátoru 98
Tabulka 10: Výhody a nevýhody PID regulátoru 98
Tabulka 11: Nastavení konstant 112
Tabulka 12: 5stupňový regulátor 114
Tabulka 13: 3stupňový regulátor 114
Tabulka 14: Přehled dosažených časů 120

Digitální přílohy

1. bludiste_navod.pdf
2. robo_cisnik_navod.pdf
3. cara_navod.pdf
4. cara_prekazka_navod.pdf
5. prekazka_navod.pdf
6. PyNXC.zip