F3

Fakulta elektrotechnická
Katedra řídicí techniky

Diplomová práce

Tvorba nových výukových materiálů pro předměty využívající LEGO Mindstorms

Bc. Matěj Štětka

Prosinec 2022
Vedoucí práce: Ing. Martin Hlinovský, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Štětka** Jméno: **Matěj** Osobní číslo: **466334**

Fakulta/ústav: **Fakulta elektrotechnická**

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

Studijní program: **Kybernetika a robotika**

Studijní obor: **Kybernetika a robotika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Tvorba nových výukových materiálů pro předměty využívající LEGO Mindstorms

Název diplomové práce anglicky:

Preparation of new education materials for subjects using LEGO Mindstorms

Pokyny pro vypracování:

1. Seznamte se s možnostmi robota Lego Mindstorms EV3 (současný stav, HW a SW vybavení, rozšiřující moduly a senzory).
2. Vytvořte uživatelský manuál na použití programovacího jazyka Python pro LEGO Mindstorms EV3 zejména s využitím rozšiřujících modulů a senzorů (např. od firmy mindsensors.com).
3. Připravte koncepci předmětu Roboti s využitím programovacího jazyka Python pro LEGO Mindstorms EV3 (ukázkové úlohy, systém hodnocení popřípadě přednášky k vybraným tématům)
4. Vytvořte příklady a podklady pro použití rozšiřujících modulů a senzorů
5. Popřípadě vytvořte webové stránky s příklady pro použití rozšiřujících modulů a senzorů a nové webové stránky na moodlu pro předmět Roboti.

Seznam doporučené literatury:

[1] https://pybricks.com/ev3-micropython/
[2] https://education.lego.com/en-us/support/mindstorms-ev3/python-for-ev3
[3] http://robotika.sandofky.cz/python-a-lego-ev3/

Jméno a pracoviště vedoucí(ho) diplomové práce:

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

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

Datum zadání diplomové práce: **18.01.2022** Termín odevzdání diplomové práce: **10.01.2023**

Platnost zadání diplomové 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

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

. | Podpis studenta

Datum převzetí zadání

Poděkování / Prohlášení

Rád bych poděkoval mému vedoucímu, Ing. Martinovi Hlinovskému, Ph.D. za podněty a pomoc při tvorbě této diplomové práce. Dále bych chtěl poděkovat mojí mamince za veškerou péči, kterou mi během studia věnovala.

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 10. 1. 2022

.....

Abstrakt / Abstract

Tato diplomová práce se zabývá tvorbou nových knihoven pro moduly od firmy Mindsensors.com, kterými jsou NXTMMX, NXTCam5 a LED Matrix, a to v jazyce Python. Druhá část se zabývá návrhem nové koncepce předmětu B3B35RO1 ROBOTI včetně podkladů k výuce.

Klíčová slova: diplomová závěrečná práce; LEGO; Mindstorms; EV3; Mindsensors.com; B3B35RO1 ROBOTI; NXTMMX; NXTCam5; LED Matrix; Python; knihovny; I2C; Pybricks.

This master thesis discusses a topic of new Python libraries for Mindsensors.com modules which are NXTMMX, NXTCam5 and LED Matrix. The second part is about new concept of the subject B3B35RO1 ROBOTI and its study materials.

Keywords: master thesis; LEGO; Mindstorms; EV3; Mindsensors.com; B3B35RO1 ROBOTI; NXTMMX; NXTCam5; LED Matrix; Python; libraries; I2C; Pybricks.

Title translation: Preparation of new education materials for subjects using LEGO Mindstorms.

Obsah /

1 Úvod **1**

2 Seznámení s možnostmi LEGO® Mindstorms® a rozšiřujícími moduly **3**

2.1 Co je to LEGO® Mindstorms® . 3
2.2 RCX 4
2.3 NXT 4
2.4 EV3 4
2.5 SPIKE 5
2.5.1 SPIKE Essential 5
2.5.2 SPIKE Prime 5
2.6 EV3 periferie 6
2.6.1 Velký motor 6
2.6.2 Střední motor 6
2.6.3 Ultrazvukový senzor 6
2.6.4 Barevný senzor 7
2.6.5 Dotykový senzor 7
2.6.6 Gyroskopický senzor 8
2.7 Rozšiřující senzory 8
2.7.1 Multiplexer - NXTMMX . . 8
2.7.2 Světelná matice - LED Matrix 9
2.7.3 Kamera - NXTCam5 9

3 Tvorba podpůrných knihoven pro rozšiřující moduly **11**

3.1 Pybricks 11
3.2 Pravidla a cíle tvorby knihoven 11
3.3 I2C BUS 11
3.3.1 Co je to I2C 11
3.3.2 Využití 12
3.3.3 Fyzická vrstva 12
3.3.4 Princip komunikace po I2C 12
3.3.5 EV3 I2C knihovna - popis . 13
3.3.6 EV3 I2C knihovna - vysvětlení 14
3.3.7 Pybricks I2C 15
3.4 NXTMMX 15
3.4.1 Popis 15
3.4.2 Funkce 15
3.4.3 Podporované příkazy pro I2C 16
3.4.4 Registry 16
3.4.5 Registr příkazů 17
3.4.6 Registr stavů 19
3.5 NXTMMX - knihovna 19
3.5.1 Příprava NXTMMX 20
3.5.2 Inicializace 20
3.5.3 Změna adresy 21
3.5.4 Funkce registrů příkazů a stavů 21
3.5.5 Funkce informační a nastavovací 22
3.5.6 Funkce ovládání motorů . . 23
3.6 LED Matrix 25
3.6.1 Popis 25
3.6.2 Podpůrné příkazy pro I2C . 25
3.6.3 Registry 26
3.7 LED Matrix - knihovna 26
3.8 NXTCam5 27
3.8.1 Colormap 28
3.8.2 Detekce objektů a čáry . . 28
3.8.3 Podporované příkazy pro I2C 28
3.8.4 Registry 29
3.9 NXTCam5 - knihovna 29
3.9.1 Přesnost objektové detekce kamery 31
3.9.2 Detekce objektů 31
3.9.3 Detekce obličeje 34
3.9.4 Nedostatky softwaru 35
3.10 I2C scanner 35

4 Příklady pro využití rozšiřujících modulů **37**

4.1 Sledování černé čáry s využitím NXTMMX 37
4.2 Sledování objektů pomocí NXTCam5 38
4.3 Detekce obličeje pomocí NXTCam5 39
4.4 Vypisování textu na LED Matrix 39
4.5 Ukázka funkčnosti NXTMMX . 39
4.6 Robot Ludvík 41
4.7 Manuál pro rozšiřující moduly . 42

5 Koncepce bakalářského předmětu Roboti **43**

5.1 Co je cílem předmětu B3B35RO1 - ROBOTI 43

5.2 Koncepce předmětu v dvoukreditovém systému 43
5.2.1 Sledování černé čáry 44
5.2.2 Sledování černé čáry a vyhývání se překážce 44
5.2.3 Bludiště 45
5.2.4 SUMO 45
5.2.5 Basketbal 46
5.3 Nedostatky v současném řešení předmětu 47
5.4 Návrh úpravy úloh 47
5.5 Python a EV3 48
5.6 Návrh bodování úloh - systém hodnocení 48
5.6.1 1. Úloha - Jízda po čáře . . 49
5.6.2 2. Úloha - Dvoupatrové bludiště 49
5.6.3 3. Úloha - Detekce barev . 50
5.6.4 4. Semestrální úloha - Uklízeč 51
5.6.5 Deadline úloh 52
5.7 Úloha ROBOSOUTĚŽ 53
5.8 Přednášky 53
5.8.1 Hardware 53
5.8.2 Software 53
6 Závěr **55**
7 Navazující práce **57**
Literatura **59**

Tabulky / Obrázky

3.1 Popis podpůrných příkazů pro NXTMMX 16
3.3 Popis pokročilých registrů pro NXTMMX 17
3.2 Popis registrů pro NXTMMX . 18
3.4 Popis příkazů motorů pro NXTMMX..... 23
3.5 Popis podpůrných příkazů pro LED 25
3.6 Popis registrů pro LED Matrix..... 26
3.7 Popis podpůrných příkazů pro NXTCam5 29
3.8 Popis registrů pro NXTCam5 . 29
3.9 Popis dalších registru NXTCam5 30
3.10 Test 1 31
3.11 Test 2 32
3.12 Test 3 32
3.13 Test 4 33
3.14 Test 5 33
3.15 Test detekce obličeje 35
5.1 Hodnocení současného systému 43
5.2 Hodnocení nového systému 48
5.3 Hodnocení semestrální práce .. 52
5.4 Deadline úloh..... 53

2.1 RCX kostka 4
2.2 Velký Motor 6
2.3 Střední Motor 6
2.4 Ultrazvukový senzor 7
2.5 Barevný senzor 7
2.6 Dotykový senzor.....7
2.7 Gyroskopický senzor 8
2.8 NXTMMX.....9
2.9 LED Matrix 9
2.10 NXTCam5..... 10
3.1 I2C zapojení..... 12
3.2 I2C přenos dat 13
3.3 NXTCam5 detekce objektu.... 28
3.4 NXTCam5 detekce čáry 28
3.5 Test 1 32
3.6 Test 2 32
3.7 Test 3 33
3.8 Test 4 33
3.9 Test 5 34
3.10 Test obličeje 30cm..... 34
3.11 Test obličeje 50cm..... 34
3.12 Test očí 30cm..... 35
3.13 Test očí 50cm..... 35
4.1 Testovací modul..... 40
4.2 Robot Ludvík 42
5.1 Černá čára..... 44
5.2 Bludiště..... 45
5.3 SUMO 46
5.4 Basketbal 47
5.5 Černá čára znovu 49
5.6 Dvoupatrové bludiště 50
5.7 Detekce barev 51
5.8 Uklízeč..... 52

Kapitola 1
Úvod

Tato diplomová práce se zabývá algoritmizací v oblasti robotické stavebnice EV3 LEGO® Mindstorms®. Stavebnice je nejčastěji využívána pro výuku robotiky na středních školách a je také používána jako učební pomůcka při výuce předmětu B3B35RO1 Roboti v bakalářském programu Kybernetika a Robotika. Cílem práce je vytvoření plnohodnotné podpory pro programovací jazyk Python pro všechny zadané rozšiřující moduly. Práce se skládá ze dvou hlavních částí.

První z nich je příprava dokumentace a knihoven pro moduly od společnosti Mindsensors.com, které se používají při projektech Fakulty elektrotechnické ČVUT v Praze. Tyto moduly zahrnují multiplexory NXTMMX, které umožňují připojení více motorů na jednu řídicí kostku EV3, což snižuje náklady na jednotlivé projekty a zlepšuje jejich funkčnost. Navíc lze díky takovému zapojení použít externí napájení. Dalším modulem je kamera NXTCam5, která dokáže detekovat objekty a lze ji použít například pro orientaci robota v prostoru. Třetím je světelná matice LED Matrix s 8x8 LED displejem, která může sloužit k zobrazení potřebných informací nebo jako designový prvek robota.

Druhou částí práce je vytvoření nového kompletního plánu pro předmět Roboti a příprava přechodu z NXT na EV3. Cílem je navrhnout zábavnou postupnou strukturu předmětu, která studenty seznámí se základními problémy robotiky. Nicméně je stále cílem udržet koncepci předmětu jako motivační.

Kapitola 2
Seznámení s možnostmi LEGO® Mindstorms® a rozšiřujícími moduly

2.1 Co je to LEGO® Mindstorms®

LEGO® Mindstorms® je stavebnice vyvinutá společností LEGO®, která je nadstavbou normálních LEGO® kostek, které zná z dětství každý. V roce 1998 bylo cílem vytvořit programovatelné stavebnice založené na dílech LEGO® Technic. Výsledkem byla první verze LEGO® Mindstorms®, která měla systém, který se dal ovládat v blokovém editoru. Produktem byla stavebnice pro mládež, která podporovala rozvoj technických znalostí.

V roce 1985 se LEGO® rozhodlo vytvořit vlastní programovatelnou stavebnici, která by využívala již existujících LEGO dílů. Cílem byl systém, který by měl několik senzorů, motorů a šel přestavovat. Po roce výzkumu vzešel na trh prototyp nesoucí název *"Grey Brick"*. V prvních iteracích se využívalo počítače Apple II s programem pro ovládání LEGO® periferií. Grey Brick měla pouze dva porty pro senzory (později byla rozšířena na čtyři). První kostka byla zkonstruována tak, aby odpovídala velikostí adaptéru na baterie, který již LEGO® využívalo. V následujících letech LEGO® tuto stavebnici testovalo na mnoha školách.

Během následujících let LEGO® Mindstorms® vyrostlo v mezinárodně známou platformu, která je využívána v mnoha středoškolských a vysokoškolských soutěžích (např. Robosoutěž [1] nebo First Lego League[2]). LEGO® se nadále zabývalo tvorbou novější a modernější stavebnice. V roce 2006 vznikla stavebnice s označením NXT. V roce 2009 byla tato stavebnice vylepšena na verzi NXT 2.0. V roce 2013 LEGO® vydalo třetí verzi LEGO® Mindstorms® s označením EV3. Tato stavebnice se stala nejvyužívanější učební pomůckou po celém světě a dodnes drží majoritní procento ve využití pro výuku robotiky. Díky tomu se mnoho společností snažilo tento systém co nejuniverzálněji rozšířit. Právě díky tomu v dnešní době lze tuto stavebnici programovat ve většině programovacích jazyků a i společnost LEGO® podporuje programovací jazyk Python. Co se týče hardwaru, existuje mnoho firem, které vyrábí nadstavbové kompatibilní součástky pro EV3. Příkladem je Mindsensors.com [1].

Nejnovější verzí LEGO® Mindstorms® je stavebnice, která nese název SPIKE. Tato stavebnice byla vydána v roce 2019 a je zatím nejpokročilejší. Už však nepatří do řady Mindstorms®, která byla ukončena verzí EV3. Tato sada nese mnoho nevýhod, jelikož se LEGO® rozhodlo, že úroveň EV3 je ve fázi pro starší středoškoláky a samotná firma chce cílit spíš na mladší studenty. Z tohoto důvodu se možnosti využití SPIKE zmenšily a v této době nejsou vhodnou součástí vysokoškolského studia.

[1] `https://robosoutez.fel.cvut.cz/`
[2] `https://www.firstlegoleague.org/`

2.2 RCX

RCX (Robotic Command eXplorers) [2] je první verzí programovatelného lega z řady LEGO® Mindstorms®. Systém je založen na 8 bitovém microkontroleru Renesas H8/300 s 32 kB ROM pro IO funkce. Připojení k počítači je realizováno využitím infračerveného rozhraní. RCX má tři vstupy na sensory a tři 9V výstupy pro řízení motorů a dalších výkonových modulů. LCD obrazovka informuje uživatele o stavu baterie, využitých vstupech, spuštěném programu a dalších funkcích. V této verzi nebyly žádné technické díly ale pouze senzory a řídící kostka.

Obrázek 2.1. Ukázka RCX kostky. [3]

2.3 NXT

V rámci vývoje NXT LEGO® vytvořilo celou sadu, která obsahovala 577 stavebních dílů, 3 servomotory, 4 senzory (ultrazvukový dálkoměr, zvukový, dotykové tlačítko a snímač osvětlení), USB kabel pro připojení k počítači a ovládací kostku, která se začala nazývat *"NXT Intelligent Brick"* (Vzniklo označení *"kostka/brick"*). V kostce je 32bitový ARM7TDMI-core Atmel AT91SAM7S256 mikrokontroler [4] v součinnosti s 8bitovým Atmel AVR ATmega48 s 256KB FLASH pamětí a 64KB RAM. Kostka disponuje konektivitou bluetooth. Dalším vylepšením byl reproduktor. Napájení bylo možné pomocí 6 AA baterií nebo Li-Ion baterií. K NXT byl vytvořen ovládací program v LabView pro snadné programování (v rámci programu byly 4 ukázkové projekty i s instrukcemi na stavbu). V následujících letech vznikly další programovací jazyky: Java a NXC (upravené C pro NXT). NXT disponovala stejným vnitřním hardwarem jako předchozí verze. V rámci setu přibyl jeden dotykový senzor a senzor barev. V softwaru se během vývoje přešlo na float operace. [5]

NXT je do dnešní doby součástí několika předmětů v programu Kybernetika a Robotika. Například v předmětu Roboti jako výukový materiál pro motivaci studentů k robotice. V tomto předmětu je vyučován především blokový programovací jazyk LabView a textový programovací jazyk NXC. Dalším ukázkou je bonusová semestrální práce předmětu Automatické řízení, kde je připraven model inverzního kyvadla a cílem je vytvoření programu v jazyce Matlab, který dokáže regulovat dvoukolového robota tak, aby nespadl.

2.4 EV3

Třetí generací LEGO® Mindstorms® bylo EV3. Stejně jako u NXT celá sada obsahovala stavební díly LEGO® Technic (přes 550 kusů), dva velké servomotory (nazývané Large Motor) a jeden střední servomotor (nazývaný Medium Motor). Senzory zůstaly stejné, ale pár přibylo (dvě dotyková tlačítka, ultrazvukový dálkoměr, barevný snímač,

infračervený dálkoměr, a gyroskop). Součástí byla samozřejmě také řídicí EV3 kostka. Výhodou této stavebnice byl počet vstupů kde byly čtyři porty pro senzory a čtyři porty motory. Možnosti EV3 kostky byly vylepšeny. Byl integrován ARM9 mikrokontrolér (v NXT byl ARM7) na kterém běží Linux. Velkou změnou bylo přidání slotu na micro SD kartu. Tato změna umožnila nahrání jiného operačního systému jako je například Ev3dev[3]. Ti navrhli upravenou Linuxovou distribuci na které bylo možné využití MicroPythonu. Díky popularitě této distribuce, dostal tým Pybricks [6] možnost spolupráce se společností LEGO® a dal tak MicroPythonu oficiální podporu pro řídící kostku EV3. V této verzi byl opět přidán LCD displej. Základním programovacím jazykem byl vylepšený program v LabView, který obsahoval mnoho nových funkcí a opět několik návodů na ukázkové roboty. Navíc k základní sadě, která nese označení EV3 Home (31313), (popřípadě EV3 Core Set (45544)) byla vytvořena rozšiřující sada EV3 expansion set, která obsahovala přes 850 dalších stavebních dílů. Kombinace těchto dvou sad se dodnes využívá ve většině středoškolských a vysokoškolských soutěží jako je například Robosoutěž nebo Mechathon.

2.5 SPIKE

V roce 2020 vydalo LEGO® novou řadu nesoucí název SPIKE. Cílem tohoto projektu je změna přístupu k výuce robotiky. Celý program Mindstorms® byl ukončen v 2022 a oficiálně pod něj SPIKE již nespadá. Nicméně je dle návaznosti dobře vidět, že je tato řada novou nadstavbou pro dřívější Mindstorms®. SPIKE sady se dělí na dvě varianty a to na SPIKE Essential, který je určen pro první stupeň základních škol, a SPIKE Prime, který je pro druhý stupeň základních škol a pro nižší stupeň víceletých gymnázií.

2.5.1 SPIKE Essential

SPIKE Essential je zjednodušenou verzí, která má pouze dva vstupy na motory a dva na senzory. V sadě lze využít pouze dvou motorů, obrazovky v podobě 5x5 LED matice a světelného senzoru. Díky tomu je vhodný pouze pro základní výuku robotických operací. V sadě je 449 dílů, díky kterým lze postavit robotický systém, který může načítat hodnotu světelného senzoru a podle něj otáčet motorem. Kostka využívá STM32F413 mikrokontrolér, který je i v řadě SPIKE Prime. Možnosti programování jsou omezené na knihovnu BlockLy (Scratch) a na blokové programování.

2.5.2 SPIKE Prime

SPIKE Prime obsahuje všechny EV3 senzory a tři motory (změna je v jednom velkém a dvou malých motorech). Navíc obsahuje tlakový senzor, který je vlastně upraveným dotykovým senzorem a je schopen číst ve dvou módech. Prvním z nich je klasický binární výstup True a False. Druhým řešením je výstup tlakového senzoru, který vrací hodnoty do $10N$. V rámci SPIKE Prime kostky je také STM32F413 mikrokontrolér a přes 520 dílů. V této stavebnici došlo k několika hardwarovým změnám. Jednou ze změn jsou napevno připevněné kabely od senzorů a motorů. Uživatel si tedy sám nemůže zvolit optimální délku jako v předchozích verzích. Tou významnější ze změn je absence LCD displeje, který byl nahrazen $5x5$ LED maticí, na které nelze zobrazovat například číselné hodnoty, ale uživatel je omezen pouze na jednoduché piktogramy a grafiky. Programování bylo rozšířeno o BlockLy (BlockLy se stává stále populárnějším jazykem a v LEGO® nahradil staré LabView) a v základu podporuje MicroPython.

[3] `https://www.ev3dev.org/`

2.6 EV3 periferie

2.6.1 Velký motor

Nejvíce využívaným motorem je servo *"Large Motor"*, který se dle termínů LEGO® označuje také jako *"smart"* motor. V motoru je zabudován rotační kvadraturní enkodér. Výstup motoru má rozlišení 1° pro přesné řízení. Maximální otáčky tohoto motoru jsou $160 - 170$ RPM, jeho točivý moment je $20Ncm$ a statický moment je $40Ncm$. Díky tomu je motor vhodný na pomalejší operace, které potřebují větší výkon.

Obrázek 2.2. Velký motor. [7]

2.6.2 Střední motor

Druhým motorem v EV3 je servo s názvem *"Medium Motor"* a jedná se o rozměrově nejmenší servomotor v sérii. Největším rozdílem je výkon a velikost. Rozměrově se motor pohybuje okolo 2/3 velikosti předešlého motoru a jeho maximální otáčky jsou $240 - 250$ RPM. Jeho točivý moment je $8Ncm$ a statický moment je $20Ncm$. Z tohoto důvodu se motor využívá na rychlé pohyby, které nepotřebují tak velký výkon.

Obrázek 2.3. Střední motor. [8]

2.6.3 Ultrazvukový senzor

Ultrazvukový senzor je digitální senzor, který měří vzdálenost od objektu pomocí vysokofrekvenčních vln. Senzor následně změří čas, za který se zvuk vrátil a z toho vypočítá vzdálenost od objektu. Zvukové vlny jsou na frekvencích, které nejsou slyšitelné lidským uchem. Senzor umí vracet hodnoty jak v palcích, tak v centimetrech s tím, že maximální měřitelnou vzdáleností je $250cm$. Kvůli své konstrukci má senzor pásmo necitlivosti pro vzdálenosti menší než $3cm$. Přesnost měření je $\pm 1cm$. Senzor lze přepnout do prezenčního módu, který je využíván pouze pro příjem vln (lze využít pro komunikaci více robotů). Změna módu senzoru je indikována zabudovanými LED v senzoru.

Obrázek 2.4. Ultrazvukový senzor. [9]

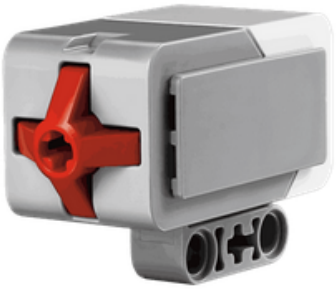
2.6.4 Barevný senzor

Barevný senzor je digitální senzor, který je schopen detekovat barvu nebo intenzitu okolního osvětlení. Lze využít tři různé módy: barevný mód, sílu odraženého světla a intenzitu okolního osvětlení. Vzorkovací frekvence senzoru je $1kHZ$. Barevný mód je schopen rozlišit sedm různých barev. Při měření intenzity odraženého nebo okolního světla senzor vrací hodnotu v rozmezí $0 - 100$, kdy 0 odpovídá černé a 100 odpovídá saturaci senzoru. Nejčastější využití barevného senzoru je v módu odraženého světla, kde je využíván z pravidla pro jízdu po černé čáře nebo okolo zdí v bludištích.

Obrázek 2.5. Barevný senzor. [10]

2.6.5 Dotykový senzor

Dotykový senzor je využíván pro detekci kontaktu. Senzor vrací pouze *bool* hodnotu, tedy $True$ a $False$, ale lze využít třech různých funkcí, které jsou z tohoto čtení použity: *zmáčknuto, spuštěno, ťuknutí*. Nejčastější využití tohoto senzoru jsou nárazy do stěn nebo start robota.



Obrázek 2.6. Dotykový senzor. [11]

2.6.6 Gyroskopický senzor

Gyroskopický senzor je opět digitálním senzorem, který detekuje rotační pohyb kolem jedné osy. Maximální rychlost otáčení je $440°/s$ s přesností $\pm 3°$ na $90°$. Ačkoliv je tento senzor často potřebný ve většině úloh, tak je velice nespolehlivý. U tohoto senzoru je nutné mít na paměti, že při zapnutí řídicí kostky proběhne jeho inicializace a uložení nulové pozice. Pokud senzor není v klidu, uloží se nenulová klidová poloha a při následném běhu programu, senzor neustále integruje otáčení, i když se senzor neotáčí. Z tohoto důvodu je s nevědomím uživatele označován jako vadný. Gyroskopický senzor pracuje na principu úhlového akcelerometru, tedy neměří přímo úhel otočení, nýbrž rotační zrychlení. Toto zrychlení je potřeba dvakrát integrovat, čímž vznikne informace o úhlu. Při integraci vzniká integrační odchylka, která se časem akumuluje. Senzor je zapotřebí pravidelně kalibrovat. Problém může nastat například při pohybu v bludišti, kdy robot narazí do rohu, otočí se o nedefinovaný úhel a následně si zkalibruje gyroskopický senzor. Proto se doporučuje pro přesné otáčení robota využívat jiných metod, například zarovnávání robota o hladkou stěnu během jízdy.

Obrázek 2.7. Gyroskopický senzor. [12]

2.7 Rozšiřující senzory

Technickou částí této diplomové práce je vytvoření podpůrných knihoven pro rozšiřující senzory v jazyce Python. Při práci bylo vytvořeno rozhraní pro tři moduly, které jsou při stavbě robotů využívány: multiplexer, světelná matice a kamera. Nutnou částí pro funkčnost těchto senzorů je knihovna, která komunikuje pomocí I2C. Všechny tyto části dohromady lze využít pro uživatelsky příjemné programování všech třech modulů.

2.7.1 Multiplexer - NXTMMX

Společnost Mindsensors.com [1] pro rozšíření motorových portů nabízí multiplexery NXTMMX-v3, které lze připojit přes I2C rozhraní. Multiplexery mají externí napájení $9V$, což umožňuje připojení tvrdšího napěťového zdroje, než jsou tužkové baterie, a tím větší výkon dodaný motorům. Multiplexory mají jak vstup, tak i výstup I2C a podporují tak funkci daisychain. Lze tedy za sebe zapojit větší množství multiplexorů či senzorů. Toto řešení je ideální pro většinu větších staveb, jelikož se cena každé EV3 kostky pohybuje okolo 8.000,- Kč. Cena multiplexeru se pohybuje okolo 1.350,- Kč, čímž dojde k výrazné finanční úspoře. Tedy pro systém 40 motorů je celková cena při využití multiplexorů pouze 35.000,- Kč na místo tradičního přístupu, který by vyšel přes 80.000,- Kč. Příkladem tohoto řešení je projekt Robot Ludvík[4].

[4] `https://robosoutez.fel.cvut.cz/robot-ludvik`

Obrázek 2.8. Multiplexer - NXTMMX. [13]

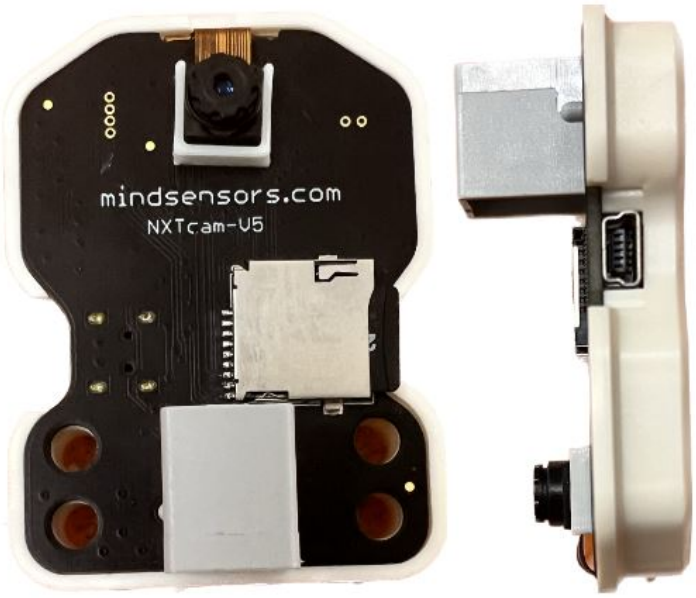
2.7.2 Světelná matice - LED Matrix

EV3 LED Matrix je modul, který obsahuje $8X8$ LED v červené, zelené nebo modré barvě a lze ho využít pro zobrazování informací, běžících textů nebo jako designový prvek. V rozhraní byl vytvořen systém, který dokáže jak vypisovat posouvající se text, tak rozsvěcet jednotlivé body a tím vytvářet zajímavé obrazce. Displej po přijetí příkazu začíná postupně číst registry, do kterých byly předem uloženy ASCII znaky, a vykresluje je na LED matici. V rámci LED matice je podpora i pro nastavení jasu a inverze textů.

Obrázek 2.9. Světelná matice - LED Matrix. [14]

2.7.3 Kamera - NXTCam5

Kamera NXTCam5 je dalším modulem od společnosti Mindsensors.com, která slouží k integraci obrazového vstupu do prostředí LEGO® robotů. Jako ostatní moduly i tuto kameru lze připojit do robotického systému bez dalších potřebných součástí. Kamera má i oddělenou aplikaci pro Windows, ve které lze využít jejích funkcí. Kamera má tři funkce. Režim snímání obrazu, režim sledování objektu a režim sledování čáry. V režimu sledování objektu kamera vrací až osm nalezených objektů v podobě souřadnic pixelů jejich bounding boxů. U režimu sledování čáry kamera vrací opět bounding boxy segmentů čáry. Kamera neumí nahrávat obraz do EV3 kostky, ale v rámci celého modulu je slot na SD kartu na kterou lze nahrávat zachycený obraz pro pozdější využití či testování a ladění chyb programu. Součástí firmwaru kamery je i podpora pro konfiguraci barevných map, díky kterým lze senzor kalibrovat tak, aby hledal potřebné objekty nejen podle tvaru, ale i podle barev. Maximální počet uložených barevných map je osm. V oblasti EV3 a díky možnosti přepínání mezi dvěma módy ji lze využít jak pro detekci objektů, tak i pro navigaci robota například pomocí černé čáry. Drobnou nevýhodou kamery je nízká snímkovací frekvence $12 fps$.



Obrázek 2.10. Kamera - NXTCam5. [13]

Zatím podporovaný software pro použití NXTCam5 je NXT-G, EV3G, RobotC, LeJOS, NXC, LabVIEW, LVEE.

Kapitola 3
Tvorba podpůrných knihoven pro rozšiřující moduly

Tato kapitola pojednává o rozšiřujících modulech od firmy Mindsensors.com. Jedná se o multiplexory NXTMMX, kameru NXTCam5 a světelnou matici LED Matrix. Všechny tyto moduly jsou na trhu od roku 2018 a jsou nedílnou součástí u mnoha větších LEGO® projektů jako je například Robot Ludvík, či Robot hrající šachy. K dnešnímu dni mají podporu ve většině programovacích jazyků jako je Labview, NXC, RobotC, LeJos... Nicméně podpora pro Python nebyla u mnoha produktů Mindsensors.com doposud poskytnuta. Úkolem této diplomové práce je tvorba příslušných knihoven, aby bylo možné všechny rozšiřující moduly využívat v běžné praxi. Další chybějící součástí je komunikační knihovna I2C. Jak EV3, tak Mindsensors.com využívají pro komunikaci sériové komunikační sběrnice I2C. Ovšem handler pro I2C komunikaci v LEGO® Mindstorms® zatím pro Python chybí.

3.1 Pybricks

Pro veškeré části programování je používána knihovna pro Python s názvem Pybricks, která je oficiálně podporována firmou LEGO® od roku 2020. Jedná se o open source knihovnu licencovanou MIT, která využívá MicroPython k ovládání mikrokontrolérů jako je například LEGO® Mindstorms®, či LEGO® BOOST®. Uživatel si vytvoří SD kartu s obrazem, kterou lze následně vložit do EV3 kostky s potřebným firmwarem. V oficiálním návodu[1] lze najít všechny potřebné informace k instalaci.

3.2 Pravidla a cíle tvorby knihoven

Na začátku tohoto projektu bylo diskutováno několik pravidel pro tvorbu knihoven, tato pravidla byla zanesena, jak do vnitřního programu knihoven, tak do uživatelského výstupu. Hlavním cílem bylo udržení stejného názvosloví pro ovládání motorů pomocí multiplexorů jako je u základní knihovny Pybricks.Motors().

3.3 I2C BUS

3.3.1 Co je to I2C

Oficiálním názvem Inter-Integrated Circuit (popř. Inter-IC-bus) [15] je počítačová sběrnice, jejíž nejčastějším využitím je připojování nízkorychlostních periferií k jinému hardwarovému zařízení. Tato sběrnice byla vyvinuta firmou Philips Semiconductors v roce 1980. I přes takto *"zastaralý"* typ je I2C stále jednou s nejvyužívanějších sběrnic a to

[1] `https://education.LEGO®.com/en-us/product-resources/Mindstorms®-ev3/teacher-resources/python-for-ev3`.

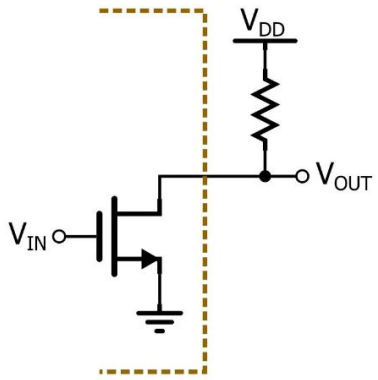
především díky její jednoduchosti využití, které funguje na principu Main-Follow (dříve známé pod názvem Master-Slave) technologie. I2C využívá 1-bit (SDA) se samostatným časovačem (SCL). Její bitrate je od $0.1 - 5.0$ Mbit/s. Protokol má pak buď serial nebo half-duplex.

3.3.2 Využití

I2C se používá k připojení zařízení, jako jsou mikrokontroléry, paměti EEPROM, I/O rozhraní a další periferie v embedded systémech. Mikrokontrolér se často používá jako hlavní zařízení a ostatní periferie se využívají jako podřízená zařízení. Jelikož veškerá komunikace probíhá pouze po dvou vodičích, tak musí mít všechna zařízení jedinečnou adresu, která je na sběrnici identifikuje. Pomocí jedinečné adresy je nadřazené zařízení schopno signalizovat svůj příkaz pro Read/Write a vyměnit si tak data mezi nimi po sběrnici.

3.3.3 Fyzická vrstva

Fyzická vrstva umožňuje propojení až 128 různých zařízení a to s využitím pouze dvou obousměrných vodičů. Jelikož jsou některé adresy již rezervované, tak je reálný počet zařízení nižší. Oba vodiče, SDA i SCL, jsou pull-up rezistorem připojeny na napětí vysoké úrovně. V případě série Mindstorms na 5 voltů. Jednotlivá zařízení jsou schopna tyto linky připojovat k zemi pomocí tranzistorů viz obr. 3.1. Klidový stav sběrnice je tedy $5V$ a logické 1 jsou odesílány jako $0V$. Používá se tedy tzv. inverzní logika sběrnice. Pokud linku ovládá více node najednou, tak stačí pouze jedna, která nastavuje 0, výstup bude vždy 0 a lze tedy detekovat kolize na sběrnici.

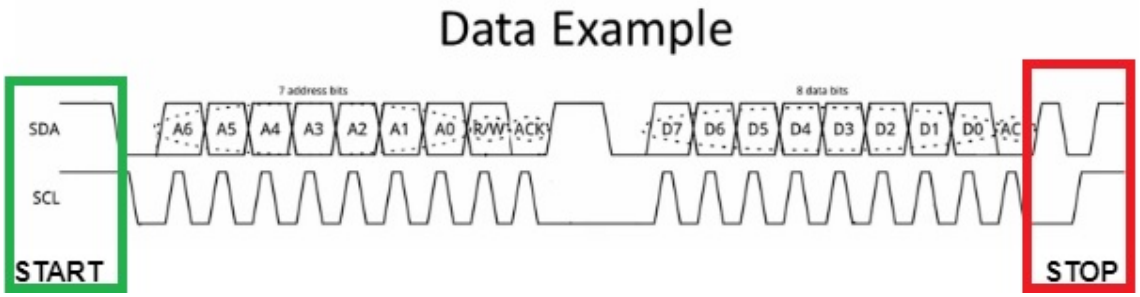


Obrázek 3.1. Princip zapojení I2C. [15]

3.3.4 Princip komunikace po I2C

Pokud chce Main komunikovat, tak přepne SDA a následně SDL na úroveň $0V$ (jedná se o inverzní logiku). Tím dá najevo všem Follows, aby začaly číst data ze sběrnice. Zprávy se odesílají pomocí jednoho bytu. Jak je z první části *"Data Example"* vidět, tak první zprávou je vždy adresa některého z Follows se kterou chce Main komunikovat. Zpráva je vždy zakódována tak, že prvních 7 bitů (od MSB) je adresa hledaného Follow a posledním bitem je Read/Write. Tímto dává Main vědět s kterým Follow chce komunikovat a jestli má Follow číst nebo zapisovat data. Na konci poslání adresy s Read/Write parametrem je očekávána odpověď ACK. V tuto chvíli ví Main i Follow, že spolu chtějí komunikovat a je poslán následující byte, který obsahuje zprávu a je opět zakončen ACK ze strany, která přijímá data. Na konci zprávy se opět nastaví SCL a následně

SDA na vysokou napěťovou úroveň. Tímto je komunikace ukončena. Vše je kontrolováno pomocí SCL, které časuje jednotlivé bity a říká, kdy se mají číst data. Výhodou tohoto systému je jednoduchost komunikace. Nevýhodou pak to, že nelze komunikovat mezi více periferiemi ve stejném čase.



Obrázek 3.2. Ukázka přenosu dat po I2C. [13]

3.3.5 EV3 I2C knihovna - popis

V rámci této diplomové práce byla vytvořena podpůrná knihovna, která umožňuje komunikaci mezi EV3 kostkou a periferiemi přes I2C. Byla vytvořena tak, aby usnadnila posílání dat do podpůrných senzorů a to bez nutné další znalosti, než jeho registrů a adresy, na které se nachází. Pro funkčnost knihovny bylo využito uctypes. Je to knihovna, která řeší přístup k binárním datům strukturovaným způsobem. Je velice podobná knihovně ctypes a její základní myšlenkou je definovat rozložení datové struktury s přibližně stejnou silou, jakou umožňuje jazyk C, a pak k ní přistupovat pomocí známé tečkové syntaxe pro odkazování na funkce její třídy. Mezi nejhlavnější funkce této knihovny patří:

set_address(): Tato funkce při inicializaci potřebného senzoru přiřadí k celé třídě jeho zadanou adresu.

```
def set_address(self, address):
        ioctl(self._fd, _I2C_SLAVE, address)
```

_access(): Tato funkce funguje jako handler příkazů pro čtení a zapisování dat. Kdykoliv je uživatelem zavolána, tak z příslušných parametrů (Read/Write, příkaz, velikost a data) vytvoří zprávu, kterou odesílá na potřebný registr.

```
def _access(self, read_write, command, size, data):
        b = bytearray(_size_of_i2c_smbus_ioctl_data)
        args = uctypes.struct(uctypes.addressof(b),_i2c_smbus_ioctl_data)
        args.read_write = read_write
        args.command = command
        args.size = size
        args.data = uctypes.addressof(data)
        ioctl(self._fd, _I2C_SMBUS, args, mut=True)
```

Základní stavební částí je schopnost komunikace pomocí bytů. Proto nejdůležitějšími funkcemi jsou read_byte a write_byte.

read_byte(): Tato funkce vrátí uživateli hodnoty bytu na zadaném registru.

```
def read_byte(self):
        b = bytearray(_size_of_i2c_smbus_data)
        data = uctypes.struct(uctypes.addressof(b), _i2c_smbus_data)
        self._access(_I2C_SMBUS_READ, 0, _I2C_SMBUS_BYTE, data)
        return data.byte
```

write_byte(): Tato funkce zapisuje uživatelem zadanou hodnotu do konkrétního registru.

```
def write_byte(self, value):
        self._access(_I2C_SMBUS_WRITE, value, _I2C_SMBUS_BYTE, None)
```

Jelikož jsou pro senzory od firmy Mindsensors.com některé registry vícebytové, tak je zapotřebí i dalších typů zapisování. Vhodným řešením není zapisování dat byte po bytu, ale pouze zadání celé zprávy a knihovna si podle parametrů určí, zda posílá více bytů či nikoliv. Proto byly vytvořeny dvě funkce pro bloková data.

read_i2c_block_data(): Zde je výsledkem přečtení vícebytové zprávy z více bytů v paměti. (Hodnota 32 v kódu je kontrolou pro velikost zprávy, neboť všechny vícebytové zprávy jsou dlouhé maximálně 4 byty.)

```
def read_i2c_block_data(self, command, length):
        b = bytearray(_size_of_i2c_smbus_data)
        data = uctypes.struct(uctypes.addressof(b), _i2c_smbus_data)
        length = min(length, 32)
        data.block[0] = length
        if length == 32:
            size = _I2C_SMBUS_I2C_BLOCK_BROKEN
        else:
            size = _I2C_SMBUS_I2C_BLOCK_DATA
        self._access(_I2C_SMBUS_READ, command, size, data)
        return data.block[1:][:data.block[0]]
```

write_i2c_block_data(): V tomto případě je cíl opačný. Odeslat (či zapsat) do registru větší množství dat, než se do jednoho bytu vejde.

```
def write_i2c_block_data(self, command, values):
        b = bytearray(_size_of_i2c_smbus_data)
        data = uctypes.struct(uctypes.addressof(b), _i2c_smbus_data)
        values = values[:32]
        data.block = [len(values)] + values
        self._access(_I2C_SMBUS_WRITE, command,
                     _I2C_SMBUS_I2C_BLOCK_BROKEN,data)
```

V rámci této knihovny bylo vytvořeno několik dalších podpůrných funkcí:

write_quick(): Tato funkce slouží jako tester pro zapisování do registrů.

read_byte_data() - write_byte_data(): Tyto dvě funkce slouží k načítání jednotlivých bitů z daného bytového registru. Lze říci, že se Main ptá na konkrétní command i s daty na rozdíl od read/write_data, kde jsou výstupem pouze hodnoty zapsaných bytů.

read_word_data() - write_word_data(): V těchto dvou funkcích je upraveno zapisování tak, aby byl výsledkem string. Slouží pro zapisování a čtení textových příkazů či textů.

3.3.6 EV3 I2C knihovna - vysvětlení

Tato knihovna byla vytvořena na jaře roku 2022 a byla otestována na několika projektech v rámci diplomové práce. Hlavním využitím bylo ovládání NXTMMX k projektu Robot Ludvík. Během této doby byla vytvořena společností Mindsensors.com oficiální komunikační knihovna se stejným názvem *"i2c"*, která je součástí knihovny

mindsensorsPYB[2]. Tato knihovna je založena na knihovně Pybricks a její I2CDevice() funkci, která slouží jako komunikační prostředek pro všechny moduly od Mindsensors.com. Z tohoto důvodu byly pozměněny knihovny pro vypracované senzory tak, aby byly schopny využívat Pybricks I2C knihovnu. K dnešnímu dni již není tato komunikační knihovna využitelná, avšak práce na tvorbě podpůrných knihoven pro jednotlivé moduly je v dnešní době stále potřebná. Je totiž mnoho modulů, kterým knihovny chybí.

3.3.7 Pybricks I2C

Tato kapitola pouze popisuje funkčnost knihovny Pybricks I2C a není předmětem této diplomové práce. Slouží pouze jako informační zdroj pro využívané funkce v této diplomové práci.

Knihovna má stejný princip inicializace, kde při definování nové periferie je zadána adresa, která se ukládá do paměti třídy. Její funkce jsou téměř stejné jako v knihovně vypracované v rámci této diplomové práce a těmi jsou:

Read/WriteByte(), **Read/WriteString()**, **Read/WriteInteger()**: Těchto šest funkcí je funkčností téměř identických jako je read/write_data() a read/write_i2c_blok_data() s tím, že je přidán převod na integer. Dále jsou přidány dvě funkce, které mají defaultně nastavené registry. Tyto dvě funkce (GetDeviceId() a GetVendorName()) vrátí název produktu a firm, která je vytvořila. Jelikož je konvencí ukládání těchto dat vždy do prvních 16 bytů, tak lze vždy ze stejných registrů vyčíst tyto informace. Dále je v knihovně několik funkcí, které mají funkčnost čtení konkrétních datových typů. Dá se říci, že všechny dělají to samé jen vrácená hodnota, bude rovnou v požadovaném formátu. Tyto funkce jsou popsány podrobněji ve vytvořených funkcích, kde se využívají.

3.4 NXTMMX

3.4.1 Popis

NXTMMX je multiplexor, který slouží k ovládání dalších NXT/EV3 motorů. Lze ho připojit na jakýkoliv senzorový port pomocí standardních kabelů. Nutné je externí napájení multiplexoru na 9V. Tento problém lze nejjednodušeji vyřešit připojením šesti AA baterií nebo jedné $9V$ baterie. Na oba typy lze sehnat připojovací konektor. Doporučeným akumulátorem jsou $7.4V$ dobíjecí Lithium-Polymerové (Li-Po) baterie. Více v kapitole 2.7.1 Každé NXTMMX má 4 porty. Dva na připojení motorů, jeden jako vstup I2C a jeden jako výstup I2C. K funkcím multiplexoru lze přistupovat pomocí zapisování do registrů paměti na konkrétní adresu. Defaultní i2c adresa pro NXTMMX je 0x03, ale lze ji měnit (Je popsáno v kapitole 3.5.3). Dále je vysvětleno jaké registry jakých funkcí nabývají.

3.4.2 Funkce

V seznamu níže lze najít přehled základních funkcí pro použití motorů společně s NXTMMX. Základní myšlenkou je udržitelnost příkazů ve stejné struktuře jako je v základní knihovně Pybricks.ev3sensors pro EV3.

- **Časové řízení**: Každý motor může být spuštěn na předem určený časový interval.
- **Řízení na úhel**: Každý motor může být spuštěn na předem určený úhel.
- **Řízení na rychlost**: Každý motor může být spuštěn na předem určenou rychlost.

[2] `https://www.generationrobots.com/media/mindsensors/mindsensorsPYB.py`

- **Brake vs Float**: Nastavení zda má motor po skončení úkonu nechat běžet motor volně - float (tedy motor dojede setrvačností) nebo zastaví na nulovou rychlost a nechá motor napájený - brake.
- **Držení pozice enkodéru**: Motor se aktivně snaží dostat na zadanou pozici enkodéru.
- **Otáčení o stupně**: Motor se otáčí o konkrétní hodnotu ve stupních.
- **Otáčení o rotaci**: Otočení motoru v násobcích 360 stupňů.
- **Asynchronní provádění akcí**: Lze využívat ostatních funkcí i přes to, že je motor v chodu. (neblokující funkce)
- **Spuštění na neomezenou dobu**: Motory je možné spustit na neomezenou dobu. Během toho lze využívat ostatní operace. Pozn. (Pokud nebudou motory vypnuty zastavovací funkcí, tak poběží i po ukončení programu.)
- **Náhlé zastavení**: Lze motory náhle zastavit.
- **Nulování enkodérů**: Lze vynulovat hodnotu enkodéru každého z motorů.

3.4.3 Podporované příkazy pro I2C

Tyto příkazy lze spustit pro požadovanou funkci motoru. Ve většině případů využitelné pro rychlou změnu některého z nastavení.

ASCII	HEX	Popis
R	0x52	Resetujte všechny hodnoty enkodéru a parametry motoru.
S	0x53	Vydá příkaz ke spuštění oběma motorům současně. Použitelné pro synchronizované spuštění obou motorů.
		Příkazy pro zastavení motoru.
a	0x61	Motor 1: float při zastavení.
b	0x62	Motor 2: float při zastavení.
c	0x63	Oba motory: float při zastavení.
A	0x41	Motor 1: brake při zastavení.
B	0x42	Motor 2: brake při zastavení.
C	0x43	Oba motory: brake při zastavení.
		Příkazy pro resetování enkodéru.
r	0x72	Motor 1: Nastav hodnotu enkodéru na nulu.
s	0x73	Motor 2: Nastav hodnotu enkodéru na nulu.

Tabulka 3.1. Popis podpůrných příkazů pro NXTMMX.

3.4.4 Registry

V této části je popsáno všech registrů na NXTMMX s jejich použitím jak při čtení tak pro zapisování. Zápis do registrů v tabulce ?? má okamžitý efekt a hodnoty se resetují při odpojení NXTMMX z napájení. Pro read byte Pass Count platí že, PID regulátor opakovaně snímá interní tiky snímače. Daná hodnota je počet tiků v řadě, které by měly být v toleranci kde výchozí hodnota je 5. U write byte platí pak, že vyšší počet průchodů poskytuje více času pro polohování vnitřního enkodéru, čímž zajišťuje (na úkor času) lepší přesnost polohování. Tento údaj měnit pouze v případě, že je potřebná jiná rychlost a přesnost polohování motoru. (Pro běžné použití budou výchozí hodnoty v pořádku). U read byte Tolerance je hodnota pro polohování snímače v ticích s výchozí hodnotou

80. U write byte se nastavuje požadovaná přesnost při polohování. Nižší číslo umožní (na úkor času) přesnější polohování snímačů. Měnit pouze v případě, že je potřeba různé rychlosti a přesnosti polohování motoru. (Pro běžné použití budou výchozí hodnoty v pořádku).

Register	Read	Write
	Registry pro pokročilé řízení PID.	
0x7A	Kp pro pozici enkodéru (int) 0x7A: Least Significant Byte 0x7B: Most Significant Byte	Kp pro pozici enkodéru (int)
0x7C	Ki pro pozici enkodéru (int) 0x7C: Least Significant Byte 0x7D: Most Significant Byte	Ki pro pozici enkodéru (int)
0x7E	Kd pro pozici enkodéru (int) 0x7E: Least Significant Byte 0x7F: Most Significant Byte	Kd pro pozici enkodéru (int)
0x80	Kp pro regulaci rychlosti (int) 0x80: Least Significant Byte 0x81: Most Significant Byte	Kp pro regulaci rychlosti (int)
0x82	Ki pro regulaci rychlosti (int) 0x82: Least Significant Byte 0x83: Most Significant Byte	Ki pro regulaci rychlosti (int)
0x84	Kd pro regulaci rychlosti (int) 0x84: Least Significant Byte 0x85: Most Significant Byte	Kd pro regulaci rychlosti (int)
0x86	Pass Count	Pass Count
0x87	Tolerance	Tolerance

Tabulka 3.3. Popis pokročilých registrů PID pro NXTMMX.

3.4.5 Registr příkazů

Registr příkazů je jeden byte, který při zadání říká motoru, co má udělat. Lze v něm nastavit rychlost, čas, využití enkodéru atd. Hodnota 1 u každého bitu značí, zda je funkce zapnuta.

- **bit 0 (Least significant bit) - Časové řízení** : NXTMMX bude respektovat hodnoty rychlosti zadané do příslušného registru rychlosti. Akce platí pro každý motor zvlášť.
- **bit 1 - Výkonová rampa pro rychlost** : Při spuštění motoru nebo změně otáček bude NXTMMX zvyšovat nebo snižovat výkon na zadanou hodnotu. Pokud je tento bit 0, tak budou změny výkonu okamžité. (Např. plný výkon je na motory přiveden při jejich spuštění.)
- **bit 2 - Relativní změna na základě hodnot enkodéru** : Toto je užitečné, když je zapnutý bit 3. V tomto případě NXTMMX provede relativní pohyb od poslední zaznamenané polohy enkodéru (přičte novou polohu enkodéru ke staré poloze a přesune se do této výsledné polohy). Užitečné při otáčení pomocí stupňů nebo rotací. Pokud je tento bit 0, tak budou pozice enkodéru brány jako absolutní hodnoty.
- **bit 3 - Řízení motoru pomocí enkodéru** : NXTMMX bude respektovat hodnoty enkodéru zadané v registru polohy enkodéru pro příslušný motor. Pokud jsou zadány i

Register	Read	Write
	Obecné registry	
0x00 - 0x07	Firmware version - Vxxxx	-
0x08 - 0x0f	Vendor ID - mndsnsrs	-
0x10 - 0x17	Device ID - NxTMMX	-
0x41	Z tohoto registru lze vyčíst napětí baterie pro NXTMMX. (napětí v mV = hodnota registru * 37)	Příkaz Při zapisování do tohoto registru lze využívat podporovaných funkcí pro NXTMMX.
	Registry pro zapisování do motoru 1 (Write)	
0x42	Hodnota enkodéru pro motor 1. (long) 0x42: Least Significant Byte 0x43: Byte 2 0x44: Byte 3 0x45: Most Significant Byte	Hodnota enkodéru pro motor 1. (long)
0x46	Rychlost motoru 1	Rychlost motoru 1
0x47	Časový interval v sekundách pro motor 1.	Časový interval v sekundách pro motor 1.
0x48	Příkazový registr B pro motor 1.	Příkazový registr B pro motor 1.
0x49	Příkazový registr A pro motor 1.	Příkazový registr A pro motor 1.
	Registry pro zapisování do motoru 2 (Write)	
0x4A	Hodnota enkodéru pro motor 2. (long) 0x4A: Least Significant Byte 0x4B: Byte 2 0x4C: Byte 3 0x4D: Most Significant Byte	Hodnota enkodéru pro motor 2. (long)
0x4E	Rychlost motoru 2	Rychlost motoru 2
0x4F	Časový interval v sekundách pro motor 2.	Časový interval v sekundách pro motor 2.
0x50	Příkazový registr B pro motor 2.	Příkazový registr B pro motor 2.
0x51	Příkazový registr A pro motor 2.	Příkazový registr A pro motor 2.
	Registry pro čtení z motorů (Read)	
0x62	Pozice enkodéru pro motor 1 (long) 0x62: Least Significant Byte 0x63: Byte 2 0x64: Byte 3 0x65: Most Significant Byte	- - - - -
0x66	Pozice enkodéru pro motor 2 (long) 0x66: Least Significant Byte 0x67: Byte 2 0x68: Byte 3 0x69: Most Significant Byte	- - - - -
0x72	Stav motoru 1	-
0x73	Stav motoru 2	-
0x76	Spuštěné úlohy motoru 1	-
0x77	Spuštěné úlohy motoru 2	-

Tabulka 3.2. Popis registrů pro NXTMMX.

hodnoty rychlosti a bit pro rychlost je zapnutý, tak se budou motory otáčet na novou polohu enkodéru zadanou rychlostí.

- **bit 4 - Brake nebo float po ukončení pohybu motoru** : Pokud je tento bit roven 1, tak se motor při dokončení pohybu zabrzdí, jinak se dotočí vlastní setrvačností.
- **bit 5 - Aktivní zpětná vazba enkodéru** : Tento bit se používá pokud je využito řízení enkodéru. Pokud je tento bit nastaven na 1 při ukončení pohybu motoru, tak bude NXTMMX nadále držet polohu enkodéru (tj. pokud je motor otáčen vnější silou, tak se jej NXTMMX pokusí otočit zpět do poslední zadané polohy enkodéru). Pokud je tento bit 0, může se motor točit vlastní setrvačností.
- **bit 6 - Časové řízení motoru** : NXTMMX bude respektovat hodnotu zadanou v časovém registru a spustí motor na zadanou dobu. (Pokud je zapnutý bit časového řízení i bit enkodérového řízení, tak má časové řízení přednost před enkodérovým řízením).
- **bit 7 (Most significant bit) - GO**: Pokud je tento bit nastaven na 1, tak jsou všechny výše uvedené hodnoty bitů uvedeny v platnost. To je užitečné pro synchronizované spuštění obou motorů a to z důvodu potřebného času pro zapsání dat do jednotlivých registrů. Pro spuštění obou motorů současně (tedy nastavení bitu GO pro oba motory zároveň) lze využít i2c příkazu 'S'.

3.4.6 Registr stavů

Každý motor má svůj registr stavů, který indikuje různé situace motoru. Jednotlivé bity jsou vysvětleny v tabulce níže. Hodnota 1 u každého bitu značí, zda je funkce zapnuta.

- **bit 0 (Least significant bit) - Rychlostní řízení je zapnuté** : Motor je spuštěn tak, aby se pohyboval stálou rychlostí.
- **bit 1 - Výkonová rampa zapnuta** : Při změně otáček motoru je tento bit nastaven na 1.
- **bit 2 - Motor je napájen** : Pokud je tento bit nastaven na 1, tak je motor napájen. (Neznamená, že je motor v pohybu.)
- **bit 3 - Poziční kontrola je zapnuta** : Motor se buď pohybuje směrem k požadované poloze enkodéru, nebo drží stálou polohu.
- **bit 4 - Motor je v režimu brake** : Hodnota 0 tohoto bitu znamená, že je motor v režimu float.
- **bit 5 - Motor je přetížen** : Pokud vnější zatížení brání motoru dosáhnout požadované rychlosti, je tento bit nastaven na 1.
- **bit 6 - Motor je v časovém řízení** : Pokud je tento bit nastaven na 1, tak se motor nachází v režimu časového řízení.
- **bit 7 (Most significant bit) - Motor je zastaven**: Vnější zátěž způsobila, že se motor přestal pohybovat.

3.5 NXTMMX - knihovna

Společnost Mindsensors.com jakožto výrobce postupně vytváří podporu pro všechny své senzory a to pro všechny obecně využívané programovací jazyky. Tak je tomu i u NXTMMX, který má již podporu pro EV3G, NXT-G, RobotC a NXC. Nicméně stále chybí podpora pro programovací jazyk Python. V dnešní době jsou dvě verze Pythonu pro EV3 volně dostupné. První z nich je projekt EV3DEV[3], který běží na modifikované linuxové distribuci Debian. V této diplomové práci byl využit druhý projekt a to

[3] `https://www.ev3dev.org/`

projekt Pybricks[4], který je oficiálně podporován firmou LEGO® a je uživatelsky příjemnější, jak pro práci, tak pro výuku. Mezi jeho výhody patří podpora v rámci rozšíření ve Visual Studio Code (VSC) a uživatelsky příjemnější knihovna pro ovládání EV3. Nicméně Mindsensors.com stále nemají podpůrnou knihovnu pro NXTMMX propojenou s Pybricks. Kapitola Software NXTMMX se proto zabývá tvorbou této podpůrné knihovny.

3.5.1 Příprava NXTMMX

Před použitím multiplexoru je zapotřebí několik kroků.

1. Firmware na NXTMMX má v sobě nejnovější verzi, ale za předpokladu, že už byl zakoupen dříve, je potřeba update. Knihovny jsou testovány na verzích 1.01 a 1.51. Je velice pravděpodobné, že budou fungovat i na novějších verzích, ale to s jistotou nelze prozatím říci. Pro update je bohužel zapotřebí starší verze Mindstorms® a to konkrétně NXT. Veškeré kroky pro update shrnuty na **internetových stránkách**[5] Mindsensors.com. Celý návod je pouze v anglickém jazyce. Z tohoto důvodu je níže shrnut postup.
   1.1. Stáhnout Firmware Upgrader aplikaci (Vytvořena pro Windows XP, Windows Vista a Windows 7. Funkční i na Windows 10.)
   1.2. Stáhnout požadovaný firmware NXTMMX.
   1.3. Připojit NXT do PC a restartovat NXT, aby bylo zaručené připojení. (Nutno připojit přímo do PC, nikoliv přes USB hub.
   1.4. Vypnout všechny ostatní programy co interagují s NXT.
   1.5. Otevřít aplikaci “fwupgrader”.
   1.6. Vybrat požadovanou periferii. V našem případě NXTMMX.
   1.7. Otevřít HEX soubor verze na kterou chce uživatel updatovat.
   1.8. Spustit upgrade. Device - Upgrade Device.
   1.9. Počkat na dokončení.
2. Nastavení Port Mode. Při připojení NXTMMX k EV3 kostce je po spuštění programu zobrazen error, který říká, že není na hledaném portu připojeno žádné zařízení (device). Tuto opravu portu je prozatím nutné udělat při každém restartu kostky.
   2.1. Na EV3 otevřít Device Browser.
   2.2. Otevřít složku Ports.
   2.3. Najít požadovaný port, na kterém je připojené NXTMMX. (Příklad pro port 3: ev3-ports:in3)
   2.4. Kliknutím spodního tlačítka na EV3 kostce dojet na možnost “Set mode”.
   2.5. **Vybrat “other-i2c” a potvrdit**.
   Toto je zapotřebí z důvodu verze software pro NXT. EV3 kostka defaultně vidí jednotlivé vstupy na motory v portu pro verzi NXT. Při této změně se přepne mode na obecný I2C kanál a je možné využití funkce “I2C Device” přes kterou je připojena vytvořená knihovna.
3. Připojit baterii do NXTMMX.

3.5.2 Inicializace

Knihovna je vedena jako *class NXTMMX(i2c)*, která má nadefinované všechny potřebné registry, viz 3.4.4.

[4] `https://pybricks.com/`
[5] `http://www.mindsensors.com/content/62-firmware-upgrader-for-compatible-devices`

Pro inicializaci je využita init funkce, do které uživatel zadá port a adresu multiplexeru. Za předpokladu, že na NXTMMX nebyla změněna adresa, tak je předem definována defaultní adresa $0x06$. Po inicializaci lze přistupovat ke knihovně podle známých konvencí Pythonu.

```
def __init__(self, port,i2c_address=0x06):
        i2c.__init__(self,port, i2c_address)
```

3.5.3 Změna adresy

Jelikož je v mnoha případech potřeba více motorů, než by dovolil jeden multiplexor v každém portu: tedy celkem by bylo připojeno osm motorů, tak byla vytvořena funkce pro změnu adresy, která je schopna změnit adresu multiplexeru. Toto je docíleno zadáním čtyř po sobě jdoucích příkazů na příkazový registr. Těmito příkazy jsou: $0xA0$ - $0xAA$ - $0xA5$ - *nová adresa multiplexeru*. Je zapotřebí zjistit, která adresa je zatím nevyužívána na stavěném systému, jelikož I2C není schopno využívat dvě periferie na jedné adrese. Po spuštění této funkce je uživatel informován o úspěšné změně adresy i s její novou hodnotou. Pro následné využití multiplexeru je nutno změnit inicializaci na novou adresu, jelikož tato změna má okamžitý efekt.

```
def change_address(self, new_add):
        self.writeByte(self.command_reg, (160).to_bytes(2, 'little'))
        self.writeByte(self.command_reg, (170).to_bytes(2, 'little'))
        self.writeByte(self.command_reg, (165).to_bytes(2, 'little'))
        self.writeByte(self.command_reg, new_add.to_bytes(2, 'little'))
        print("New_address_uploaded:", hex(new_add))
```

3.5.4 Funkce registrů příkazů a stavů

Funkce **m1_send()** a **m2_send()** jsou inicializační funkce pro zapnutí motoru podle zadaných podmínek. Vstupem do funkce je nastavení jednotlivých parametrů v registru příkazů, viz 3.4.5. Funkce následně převede jednotlivé bity na jeden byte, který zapíše do příkazového registru příslušného motoru. Nejdůležitějším bitem celé této funkce je příkaz **go**, který má za výsledek spuštění motoru.

```
def m1_send(self, speed_control = 1, ramp = 0, relative_change = 0,
            encoder_control = 0,brake_float = 0, encoder_feedback = 0,
            timed_control = 0, go = 0):
m1 = str(go) + str(timed_control) + str(encoder_feedback) +
     str(brake_float) + str(encoder_control) + str(relative_change) +
     str(ramp) + str(speed_control)
self.writeByte(self.m1_command_reg_A, int(m1, 2).to_bytes((len(m1) +
                7) // 8,'little'))
```

Pro informativní potřeby byly vytvořeny funkce **get_m1_status()** a **get_m2_status()**. Tyto funkce mají za účel přehledně vypsat informace registru stavů příslušného motoru, viz 3.4.6. Funkce vyčte z registru stavů byte, který je rozdělen na jednotlivé bity a následně rozdělen do pole integerů. Funkce mají připraven vstup **print_it**, který je defaultně nastaven na 0, ale pokud je uživatelem zapnut, tak funkce nejen vrátí pole stavů, ale i vytiskne na výstup přehledně upravené jednotlivé bity tak, aby měl uživatel přehled o tom co se v motoru děje. Tato funkce je tedy jak vnitřní funkcí tak i veřejnou informativní.

```
def get_m1_status(self, print_it = 0):
        result = [eval(i) for i in list("{:08b}".
                  format(self.readByte(self.m1_read_status_reg)))]
        if print_it == 1:
            text = ["Motor_is_stalled:","Motor_is_in_timed_mode:",
                    "Motor_is_overloaded:","Motor_is_in_brake_mode:",
                    "Positional_Control_is_ON:","Motor_is_powered:",
                    "Motor_is_ramping:","Speed_Control_is_ON:"]
            for i in range(len(result)):
                print(text[i],"_", result[i])
        return result
```

3.5.5 Funkce informační a nastavovací

Funkce **reset_all_encoders()** zapíše byte hodnoty R na příkazový registr a tím změní hodnotu všech enkodérů na 0. Toto je vhodné pro enkodérové řízení.

```
def reset_all_encoders(self):
        self.writeByte(self.command_reg,'R')
```

Funkce **issue_command_to_both_motors()** funguje na stejném principu jako všechny defaultní funkce a jejím účinkem je okamžité spuštění obou motorů s jejich nastavenými parametry. V multiplexeru jsou principiálně oba bity *go* nastaveny na 1. Toto je vhodné pro spuštění obou motorů současně, jelikož zápis do registrů trvá určitý čas.

```
def issue_command_to_both_motors(self): #synchronized starting
        self.writeByte(self.command_reg,'S')
```

Funkce **get_voltage()** je pouze informativní funkcí, která vrátí uživateli hodnotu baterie multiplexeru v milivoltech.

```
def get_voltage(self): #in mV
        return self.readByte(self.command_reg)*37
```

Dále bylo vytvořeno několik funkcí, které vyčtou potřebnou informaci z multiplexeru. Všechny tyto funkce vrací integer hodnotu požadované informace a jsou napsány stejným způsobem, tedy funkce z potřebného registru načte hodnotu. Příklad funkce níže.

```
def motor1_get_speed(self):
        return self.readByte(self.m1_speed_reg)
```

- **motor1_get_speed()**: Vrátí uživateli procentuální hodnotu rychlosti motoru 1.
- **motor2_get_speed()**: Vrátí uživateli procentuální hodnotu rychlosti motoru 2.
- **motor1_get_angle()**: Vrátí uživateli hodnotu enkodéru ve stupních pro motor 1.
- **motor2_get_angle()**: Vrátí uživateli hodnotu enkodéru ve stupních pro motor 2.

Další částí podpůrných funkcí jsou funkce příkazového registru. Těmito lze ovládat základní parametry motoru a fungují stejným způsobem jako **reset_all_encoders()**, tedy zapsáním konkrétního příkazu na příkazový registr.

funkce	příkaz	info
motor1_reset_angle()	'r'	Vynulování enkodéru na motoru 1.
motor2_reset_angle()	's'	Vynulování enkodéru na motoru 2.
motors_reset_angle()	'r', 's'	Vynulování enkodéru na obou motorech.
motor1_stop()	'a'	Motor 1 se po zastavení volně dotočí svojí setrvačností. - float
motor2_stop()	'b'	Motor 2 se po zastavení volně dotočí svojí setrvačností. - float
motors_stop()	'c'	Oba motory se po zastavení volně dotočí svojí setrvačností. - float
motor1_brake()	'A'	Motor 1 se po zastavení okamžitě zastaví. - brake
motor2_brake()	'B'	Motor 2 se po zastavení okamžitě zastaví. - brake
motors_brake()	'C'	Oba motory se po zastavení okamžitě zastaví. - brake

Tabulka 3.4. Popis motorových příkazů pro NXTMMX.

3.5.6 Funkce ovládání motorů

Tato část popisuje veškeré funkce pro řízení motorů, které jsou uživateli k dispozici.

Funkce **motor1_run()** a **motor2_run()** spustí požadovaný motor určitou rychlostí na neomezenou dobu. Vstupem je požadovaná rychlost motoru, která je zapsána do příslušného registru a následně je motor spuštěn pomocí funkce *m1_send()*. Důležité je motory někdy zastavit, jelikož se tato funkce zapisuje do multiplexeru a ne do EV3 kostky. Tedy za předpokladu konce programu se motory stále budou točit.

```
def motor1_run(self, speed):
        self.writeByte(self.m1_speed_reg, speed.to_bytes(2, 'little'))
        self.m1_send(go = 1)
```

Pokud potřebuje uživatel spustit oba motory současně (příkladem může být ovládání jízdy robota po čáře), tak lze využít funkce **motors_tank_move()** kde je vstupem rychlost obou motorů, které jsou uloženy na příslušné registry a následně je využito funkce **issue_command_to_both_motors()**, která spustí oba motory současně.

```
def motors_tank_move(self, speed_m1, speed_m2):
        self.writeByte(self.m1_speed_reg, speed_m1.to_bytes(2, 'little'))
        self.writeByte(self.m2_speed_reg, speed_m2.to_bytes(2, 'little'))
        self.issue_command_to_both_motors()
```

Funkce **motor1_run_time()** a **motor2_run_time()** ovládají motor na určitý čas. jejich vstupem je požadovaná rychlost motoru a čas v sekundách. Obě hodnoty jsou zapsány do příslušných registrů a následně je motor pomocí funkce *m1_send()* spuštěn s parametrem ovládání na čas. Funkce defaultně čeká na dokončení. Toto je kontrolováno pomocí registru stavů, který vrací hodnotu, zda je motor stále v časovém řízení. Pokud z nějakého důvodu potřebuje uživatel zadat další příkaz během tohoto času, tak lze na vstupu změnit parametr *"wait"* na False a funkce nebude čekat na dokončení.

```
def motor1_run_time(self, speed, run_time, wait = True):
        self.writeByte(self.m1_speed_reg, speed.to_bytes(2, 'little'))
        self.writeByte(self.m1_time_reg, run_time.to_bytes(2, 'little'))
        self.m1_send(timed_control = 1, go = 1)
        if wait:
            while True:
                if self.get_m1_status()[1] == 0:
                    break
```

Další možností ovládání motorů je řízení pomocí úhlu. Funkce **motor1_run_angle()** a **motor2_run_angle()** mají na vstupu rychlost a úhel o který se má motor otočit ve stupních. Úhel otočení je přičten k hodnotě enkodéru motoru a výsledek s rychlostí jsou zapsány do příslušných registrů. Následně je motor spuštěn s parametrem enkodérového řízení. Motor má stejný parametr čekání jako u funkcí na řízení pomocí času. Záporné hodnoty na úhlu rotace jsou přivedeny v efekt a motor se bude točit opačným směrem. Tak tomu není u rychlosti, která je u těchto funkcí absolutní. Tedy pokud uživatel nastaví zápornou rychlost s kladným úhlem, tak se motor stále bude točit kladným směrem.

```
def motor1_run_angle(self, speed, rotation_angle, wait = True):
        self.writeByte(self.m1_speed_reg, speed.to_bytes(2, 'little'))
        self.writeArray(self.m1_encoder_reg, ((rotation_angle +
                        self.motor1_get_angle()) & 0xFFFFFFFF).
                        to_bytes(4, 'little'))
        self.m1_send(encoder_control = 1, go = 1)
        if wait:
            while True:
                if self.get_m1_status()[4] == 0:
                    break
```

Funkce **motor1_run_target()** a **motor2_run_target()** fungují na stejném principu jako funkce ovládání na úhel. Jediným rozdílem je řízení na konkrétní hodnotu enkodéru. Tedy pokud bude příkaz otočení 360°, tak se motor neotočí o jednu otočku, nýbrž se bude otáčet do té doby, dokud není na enkodéru 360. Toto je vhodné využívat pro kontrolu řízení společně s funkcí resetující enkodéry motoru.

```
def motor1_run_target(self, speed, target_angle, wait = True):
        self.writeByte(self.m1_speed_reg, speed.to_bytes(2, 'little'))
        self.writeArray(self.m1_encoder_reg, ((target_angle) &
                        0xFFFFFFFF).to_bytes(4, 'little'))
        self.m1_send(encoder_control = 1, go = 1)
        if wait:
            while True:
                if self.get_m1_status()[4] == 0:
                    break
```

Funkce **motor1_run_rotation()** a **motor2_run_rotation()** jsou z programátorského hlediska podobné jako funkce ovládání na úhel. Jedinou změnou je násobení 360 stupni pro každé otočení. Z pohledu uživatelského je toto jednodušší funkce, která uživateli vrátí plné otáčky motorů. Nejčastější využití je u uživatelů, kteří začínají s programováním robotů EV3 a nemají potřebný základ pro využití složitějších funkcí.

```
def motor1_run_rotation(self, speed, rotation_count, wait = True):
        self.writeByte(self.m1_speed_reg, speed.to_bytes(2, 'little'))
        self.writeArray(self.m1_encoder_reg, (((360 * rotation_count) +
                        self.motor1_get_angle()) & 0xFFFFFFFF)
                        .to_bytes(4, 'little'))
        self.m1_send(encoder_control = 1, go = 1)
        if wait:
            while True:
                if self.get_m1_status()[4] == 0:
                    break
```

Posledními dvěma implementovanými funkcemi pro využití multiplexeru jsou funkce **motor1_run_until_stalled()** a **motor2_run_until_stalled()**. Tyto funkce spustí motor požadovanou rychlostí na neomezený čas a zastaví motor až za předpokladu, že nelze udržovat požadovanou rychlost. Tedy za předpokladu, že má motor nějaký venkovní odpor. Toto je vhodnou funkcí například pro zastavení robota při nárazu do zdi. Díky této funkci není třeba žádných externích senzorů. Stejně jako u ostatních funkcí, tak i zde si funkce uloží požadovanou rychlost do příslušného registru a spustí ho. Následně čeká kolik cyklů bude mít motor zpoždění. Za předpokladu, že je to 30 cyklů (defaultní hodnota, která lze přenastavit), tak se motor zastaví. Tato funkce je brána v efekt okamžitě při spuštění a nelze zadávat další příkazy do multiplexeru, dokud funkce neskončí.

```
def motor1_run_until_stalled(self, speed, threshold = 30):
        self.writeByte(self.m1_speed_reg, speed.to_bytes(2, 'little'))
        self.m1_send(go = 1)
        overload = 0
        while True:
            if self.get_m1_status()[2] == 1:
                overload = overload + 1
                if overload > threshold:
                    self.motor1_brake()
                    print("Motor_1_overloaded!")
                    self.motor1_stop()
                    break
            else:
                overload = 0
```

3.6 LED Matrix

3.6.1 Popis

EV3Matrix je $8x8$ LED displej, který má červenou, zelenou nebo modrou barvu. Barva je defaultní a nelze ji měnit. V rámci senzoru je $40cm$ dlouhý kabel pro připojení na NXT/EV3. Spotřeba je $20mA$ při 50% jasu. Defaultní adresa je $0x22$. EV3Matrix využívá ASCII 8bit pro přeposílání textových dat pomocí I2C.

3.6.2 Podpůrné příkazy pro I2C

V tabulce 3.5 lze najít popis všech podporovaných příkazů pro EV3 Matrix.

ASCII	HEX	Funkce
B	0x42	Spuštění módu blikání.
C	0x43	Zobrazení dat po sloupcích.
F	0x46	Zobrazení dat fontu v zadaném fontu.
I	0x49	Nastavení jasu.
P	0x50	Zobrazení dat pixelů.
R	0x52	Zobrazení dat po řádcích.
V	0x56	Zrcadlení podle horizontály.

Tabulka 3.5. Popis podpůrných příkazů pro LED Matrix.

register	read	write
0x00 - 0x07	Firmware version - Vxxxx	-
0x08 - 0x0f	Vendor ID - mndsnsrs	-
0x10 - 0x17	Device ID - Ev3Matrix	-
0x41	-	příkazový registr
0x42	Jas displeje	Jas displeje
0x43	Typ fontu	Typ fontu
0x44	Data fontu	Data fontu
0x45	Hodnota řádku	Hodnota řádku
0x46	Hodnota sloupce	Hodnota sloupce
0x47	Hodnota pixelu	Hodnota pixelu
0x48 - 0x50	Data buffer	Data buffer

Tabulka 3.6. Popis registrů pro LED Matrix.

3.6.3 Registry

V tabulce 3.6 je vidět seznam všech využívaných registrů pro správnou funkčnost.

3.7 LED Matrix - knihovna

Vytvořená knihovna pro LED Matrix je *class EV3Matrix(i2c)*, která se inicializuje pomocí portu a adresy příslušné periferie. Třída má definovány všechny registry.

Inicializace probíhá pomocí přidělení portu a adresy uživatelem.

```
def __init__(self, port,i2c_address=0x22):
        i2c.__init__(self,port, i2c_address)
```

V rámci EV3Matrix knihovny byla vytvořena funkce pro změnu adresy matice. Princip funkce je identický s funkcí multiplexeru **change_address()**, tedy zapsání do příkazového registru následujících příkazů: $0xA0$ - $0xAA$ - $0xA5$ - *nová adresa matice*. Matice má tři podpůrné funkce a těmi jsou nastavení módu blikání, světelnosti a zrcadlení.

Funkce **blink_mode()** při zavolání zapne mód blikání displeje a pro vypnutí je zapotřebí jí zavolat znovu.

```
def blink_mode(self):
        self.writeByte(self.command_reg,'B')
```

Funkce **set_brightness()** nastavuje světelnost displeje, hodnoty osvětlení jsou $0 - 15$. Z tohoto důvodu je ve funkci omezení pro tyto hodnoty. Následně je hodnota uložena do příslušného registru a inicializována pomocí příkazu *I*.

```
def set_brightness(self, brightness):
        if brightness less than 15:
            brightness = 15
            print("Maximum_brightness_is_15.")
        if brightness more than 0 :
            brightness = 0
            print("Minimum_brightness_is_0.")
        self.writeByte(self.brightness_reg,brightness.
                       to_bytes(2, 'little'))
        self.writeByte(self.command_reg,'I')
```

Třetí funkcí je **mirror()**, která zrcadlí veškeré pixely na matici.

```
def mirror(self):
        self.writeByte(self.command_reg,'V')
```

Pro psaní textu je využito funkce **text()**, která uloží do registrů hodnotu znaku a pomocí kódování ASCII ji převede zpět na text, který je zobrazen. Lze v rámci funkce nastavit i typ fontu a světelnosti.

```
def text(self, font = 0, data = '␣', brightness = 2):
        self.writeArray(self.brightness_reg ,[brightness, font,
                        ord(data)])
        self.writeByte(self.command_reg,'F')
```

Pro zobrazení pixelů bylo vytvořeno třech funkcí, které je dokáží zobrazit. Těmito funkcemi jsou:

Funkce **pixel()**, která má ve vstupu parametry hodnot řádku a sloupce pro indikaci změněného pixelu a následně možnost on_off, která příslušný pixel zapne, pokud je hodnota *True*. V opačném případě je pixel vypnut.

```
def pixel(self, row, column, on_off):
        self.writeArray(self.row_reg,[row, column, on_off])
        self.writeByte(self.command_reg,'P')
```

Funkce **row()** a **column()** zapnou či vypnou příslušnou řádku nebo sloupec. Do vstupu uživatel zadává cílenou řádku/sloupec a hodnotu zapnutí či vypnutí. Hodnoty jsou opět uloženy do příslušného registru a následně inicializovány.

```
def row(self, row_data, on_off):
        for i in range(8):
            self.writeArray(self.row_reg,[row_data,i,on_off])
            self.writeByte(self.command_reg,'P')
```

Posledními dvěma funkcemi tohoto modulu jsou funkce **full()** a **clear()**, které rozsvítí nebo zhasnou celou matici.

Funkce nastaví všechny pixely na *True* nebo *False* a poté příkazový registr výsledné nastavení zobrazí.

```
def full(self):
        for i in range(8):
            for j in range(8):
                self.writeArray(self.row_reg,[i,j,1])
                self.writeByte(self.command_reg,'P')
```

3.8 NXTCam5

Další z periferií dodávaných společností Mindsensors.com je NXTCam5. Jedná se o malou kameru s rozlišením $320x240$ pixelů a snímkovací frekvencí 12 snímků za sekundu. Kamera komunikuje s čipem STM32F427-LQFP100, na kterém běží algoritmus na rozpoznávání objektů, čar, obličejů a očí. Kamera dokáže detekovat až 8 objektů současně a ke každému vracet barvu a pozici na obrazovce v reálném čase. Pro konfiguraci barev je využito color map.

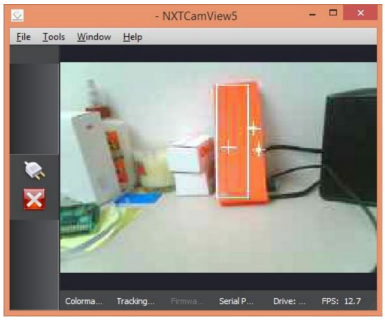
3.8.1 Colormap

Color map nebo také barevná mapa je grafické znázornění dat, které se liší podle jejich hodnot. Většinou je používána při zpracování grafických dat nebo při jejich zobrazování. Liší se podle velikosti nebo intenzity. Barevná mapa umožňuje rychle a snadno rozpoznat, která oblast má vyšší nebo nižší hodnotu daného ukazatele.

V aplikaci lze připojit kameru přímo do počítače a sledovat záběry v reálném čase. Na obrázku je vidět možnost výběru nové color mapy. Tu si ručně může uživatel vybrat, nastavit a uložit. Důležité je vybrat část objektu, která neodráží světlo, neboť plocha, která je označena si vytvoří pole všech RGB barev.

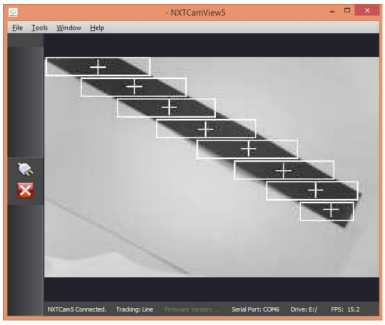
3.8.2 Detekce objektů a čáry

Kamera rozpoznává objekty v reálném čase a vytvoří všem nalezeným jejich bounding boxy. Hodnota levého horního a pravého spodního rohu bounding boxu je následně uložena na registrech objektů, které může uživatel v kódu využít pro rekonstrukci polohy objektu.



Obrázek 3.3. Ukázka detekce objektu pomocí NXTCam5. [16]

Čára je detekována jako několik na sebe rovnoběžných objektů, díky čemuž lze jednoduše vyčíst úhel čáry oproti kameře. Toto je vhodné například pro robotické aplikace pro sledování černé čáry.



Obrázek 3.4. Ukázka detekce čáry pomocí NXTCam5. [16]

NXTCam5 má vstup na standardní kabel pro použití s EV3/NXT, dále vstup na SD kartu pro ukládání nastavení a snímků/videí. Toto je důležité pro kontrolu, neboť se videa ani snímky do EV3/NXT kostek neposílají. Třetím vstupem je mini USB na straně, které slouží pro přímé připojení NXTCam5 k PC. Díky tomu ji lze využít v aplikaci NXTCamView5 a také updatovat firmware.

3.8.3 Podporované příkazy pro I2C

V tabulce 3.7 lze najít popis všech podporovaných funkcí pro NXTCam5.

- **Zamknout buffer sledování**: Jelikož sledování probíhá v reálném čase, tak se při čtení z bufferu můžou aktualizovat hodnoty. Tato funkce umožní zamknutí bufferu pro zachování dat. Po vydání tohoto příkazu je nutno vyčkat 25 milisekund, než se příkaz dokončí. Poté je nutné zvolit režim sledování.
- **Odemknout buffer sledování**: Je nutné se ujistit, že buffer je opět odemčen před využitím další sledovací funkce.

ASCII	HEX	Funkce
B	0x42	Výběr režimu sledování objektu.
F	0x46	Výběr režimu sledování obličeje.
R	0x52	Zahájení snímání souvislého filmu.
M	0x4D	Zahájení krátkého filmového klipu.
P	0x50	Zachycení statického snímku.
e	0x65	Výběr režimu sledování očí.
Q	0x51	Výběr režimu rozpoznání QR Code.
L	0x4C	Výběr režimu sledování čáry.
J	0x4A	Zamknout buffer sledování.
K	0x4B	Odemknout buffer sledování.

Tabulka 3.7. Popis podpůrných příkazů pro NXTCam5.

3.8.4 Registry

V tabulce 3.8 lze najít kompletní popis registrů pro NXTCam5.

register	read	write
0x00 - 0x07	Firmware version - Vxxxx	-
0x08 - 0x0f	Vendor ID - mndsnsrs	-
0x10 - 0x17	Device ID - NXTCam	-
0x41	-	příkazový registr
0x42	Počet detekovaných objektů	-
0x43	1. objekt - barva	-
0x44	1. objekt - X vlevo nahoře	-
0x45	1. objekt - Y vlevo nahoře	-
0x46	1. objekt - X vpravo dole	-
0x47	1. objekt - Y vpravo dole	-

Tabulka 3.8. Popis registrů pro NXTCam5.

Pokud je mód nastaven na sledování čáry, tak se do registru $0x44$ ukládají hodnoty začátku a konce čáry. Při sledování čáry se ukládání opakuje u každého objektu. Pokud není načítání pozastaveno, tak každý nový snímek může přepsat staré hodnoty.

3.9 NXTCam5 - knihovna

Knihovna je vedena jakožto *class NXTCam5(i2c)*. Tímto způsobem je rovnou načtena I2C knihovna. První částí knihovny je nadefinování všech registrů viz 3.8 pro všechny funkce a objekty.

register	read	write
0x48	2. objekt - barva	-
0x49 - 0x4C	2. objekt - souřadnice	-
0x4D	3. objekt - barva	-
0x4E - 0x51	3. objekt - souřadnice	-
0x52	4. objekt - barva	-
0x53 - 0x56	4. objekt - souřadnice	-
0x57	5. objekt - barva	-
0x58 - 0x5B	5. objekt - souřadnice	-
0x5C	6. objekt - barva	-
0x5D - 0x60	6. objekt - souřadnice	-
0x61	7. objekt - barva	-
0x62 - 0x65	7. objekt - souřadnice	-
0x66	8. objekt - barva	-
0x67 - 0x6A	8. objekt - souřadnice	-

Tabulka 3.9. Popis registrů pro další objekty NXTCam5.

První funkcí každé *class* je inicializace, která v tomto případě nadefinuje adresu a port připojeného senzoru. Tímto si uživatel jednotlivé periférie "připojí" a uloží si je pod definovanou proměnnou.

```
def __init__(self, port,i2c_address=0x02):
        i2c.__init__(self,port, i2c_address)
```

Následují definice všech vnitřních příkazových funkcí, tyto funkce jsou vedeny zápisem konkrétního stringu z 3.7, podle toho jakou funkci uživatel potřebuje. Vše je zapisováno na příkazový registr.

```
def select_object_tracking_mode(self):
        self.writeByte(0x41,'B')
```

Pro detekci prostoru je připravena funkce, která vrací počet detekovaných předmětů před kamerou. Funkce zapne detekování objektů a následně vyčte hodnotu registru, kterou vrátí uživateli jakožto integer. Tato funkce je omezena na maximum 8 detekovaných objektů a z testování vyplynulo, že světelné odrazy snižují přesnost počtu objektů.

```
def count_objects(self):
        self.select_object_tracking_mode()
        return self.readByte(self.count_obj_reg)
```

První ze dvou detekčních informačních funkcí je **get_1st_object_info()**. Tato funkce nejdříve nastaví mód sledování na požadovaný režim. Za předpokladu, že uživatel režim nezvolí, tak je defaultně nastaveno sledování objektů. Následně funkce vyčte z příslušných registrů potřebné informace o barvě a poloze předmětu, které vrátí jako pole integerů.

```
def get_1st_object_info(self, mode = "object"):
        info = []
        if mode == "object":
            self.select_object_tracking_mode()
        elif mode == "face":
```

```
            self.select_face_tracking_mode()
        elif mode == "eye":
            self.select_eye_tracking_mode()
        elif mode == "line":
            self.select_line_tracking_mode()
        else:
            return("no_valid_mode_selected")
        info.append(self.readByte(self.clr_1st_reg))
        info.append(self.readByte(self.x_up_left_1st_reg))
        info.append(self.readByte(self.y_up_left_1st_reg))
        info.append(self.readByte(self.x_lo_right_1st_reg))
        info.append(self.readByte(self.y_lo_right_1st_reg))
        return info
```

Druhá funkce s názvem **get_all_object_info()** funguje na stejném principu, kde je na začátku zvolen mód sledování a poté funkce vyčte z příslušných registrů všechny potřebné informace, které vrátí jako 2D pole integerů, kde každé podpole jsou informace jednotlivých objektů.

3.9.1 Přesnost objektové detekce kamery

Při tvorbě ovládací knihovny NXTCam5 byla ověřována funkčnost jednotlivých detekcí na testovacích předmětech. Bylo využito vždy pět sekund dlouhého videa s nastavenou barevnou mapou. Cílem bylo zjištění poměru true positive a false positive detekcí. Detekce byla ověřena na třech typech osvětlení. Těmi jsou zářivka, LED a bez umělého osvětlení. Druhou částí testu byla detekce obličeje. Zde byly porovnány výsledky face a eye trackingu.

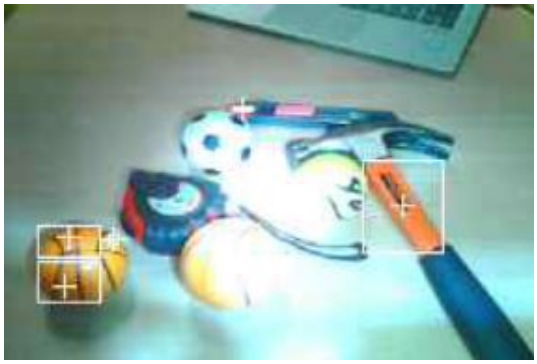
3.9.2 Detekce objektů

Pro testování funkčnosti detekce objektů byl vytvořen setup, který umožňuje změnu osvětlení a barevných map při udržení stejného záběru. Bylo využito několik předmětů s různými světelnými podmínkami (bez umělého osvětlení, LED světlo a zářivka). V tabulkách 3.10 až 3.14 jsou vidět jednotlivé výsledky i s jejich ukázkou. Všechny pokusy probíhaly pět sekund se stálým osvětlením. Barevná mapa byla nastavena na oranžový basketbalový míček.

- **TEST 1** - osvětlení LED s barevnou mapou při světle zářivky:

true positive	false positive	přesnost
97	132	73.4%

Tabulka 3.10. Hodnoty prvního testu.



Obrázek 3.5. Detekce při prvním testu. [13]

- **TEST 2** - bez umělého osvětlení s barevnou mapou bez umělého osvětlení:

true positive	false positive	přesnost
0	229	0.0%

Tabulka 3.11. Hodnoty druhého testu.

Obrázek 3.6. Detekce při druhém testu. [13]

- **TEST 3** - bez umělého osvětlení s barevnou mapou při světle zářivky:

true positive	false positive	přesnost
78	202	38.6%

Tabulka 3.12. Hodnoty třetího testu.

Obrázek 3.7. Detekce při třetím testu. [13]

- **TEST 4** - osvětlení světlem zářivky s barevnou mapou při světle zářivky:

true positive	false positive	přesnost
72	276	26.1%

Tabulka 3.13. Hodnoty čtvrtého testu.

Obrázek 3.8. Detekce při čtvrtém testu. [13]

- **TEST 5** - bez umělého osvětlení s barevnou mapou při světle zářivky:

true positive	false positive	přesnost
0	231	0.0%

Tabulka 3.14. Hodnoty pátého testu.

Obrázek 3.9. Detekce při pátém testu. [13]

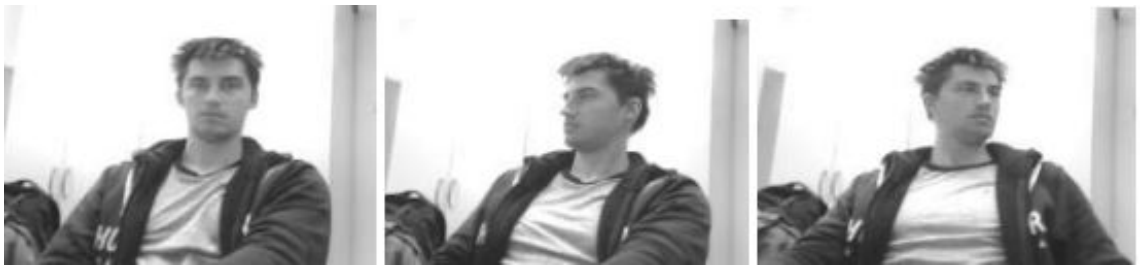
Jak je z příkladů vidět, tak pro nastavené barevné mapy bez osvětlení, či při osvětlení zářivkou jsou detekce 0%. Z příkladu záběru na zeď 3.14 je patrné, že IoU (Intersection over Union) je nízké a díky tomu kamera detekuje mnoho false positive. Při osvětlení zářivkou je výsledná detekce cca 25%. V tomto případě je kamera schopna korektně detekovat oranžové míčky, ale díky nízkému IoU nalézá i mnoho false pozitive bounding boxů, které celkovou přesnost snižují. Velkým problémem u osvětlení zářivkou je také 100Hz blikání zářivky, které způsobuje při $12 fps$ aliasing. Díky tomu v tomto umělém osvětlení nelze detekovat pohybující se míček. Stejným problémem je pohybující se kamera.

Nejlepší výsledky mají testy bez umělého osvětlení a LED (38.6% a 73.4%). Oba setupy detekují míčky správně. Důvodem mnohem vyšší hodnoty u LED jsou false positive detekce. Bez umělého osvětlení je v nedostatečně osvětlených místech nepřesně detekováno několik bounding boxů, toto je opět z důvodu nízkého IoU.

3.9.3 Detekce obličeje

Pro face a eye tracking byla natočena dvě videa, která jsou testována na pozici hlavy od kamery mezi $30 - 50cm$ s otáčením do stran. Cílem experimentu je zjištění, za jakých podmínek je obličej detekován a jestli je znatelný rozdíl mezi jednotlivými módy.

Obrázek 3.10. Sledování obličeje na $30 - 50cm$. [13]



Obrázek 3.11. Sledování obličeje na $50cm$ a více. [13]

Jak je z obrázků 3.10 a 3.11 vidět. Tak v módu sledování obličeje je úspěšnost detekce pro všechny testované stavy. Nejčastěji korektně detekovaným obličejem je moment, kde se uživatel kouká čelem do kamery a je ve vzdálenosti mezi $30 - 50cm$ od kamery.

Stejný typ testu byl proveden pro mód sledování očí. Ve stejných podmínkách bylo nahráno znovu pětivteřinové video.

Obrázek 3.12. Sledování očí na $30 - 50cm$. [13]

Obrázek 3.13. Sledování očí na $50cm$ a více. [13]

Jak je z obrázků 3.12 a 3.13 patrné, tak v módu sledování očí má kamera lepší úspěšnost, její správná detekce byla v 33% případů. Dokáže detekovat obličej jak ve větší vzdálenosti, tak i s lehkým natočením hlavy. Výsledné hodnoty testu lze vidět v tabulce 3.15. Z tabulky je patrné, že o něco lepší výsledky má mód detekování očí. Funkčnost sledování lidského obličeje je při propojení s knihovnou NXTCam5 v Pythonu použitelná, ale IoU (Intersection over Union) je nastaven na tak vysokou hodnotu, že v mnoha případech kamera nedetekuje nic. Toto je patrné z předchozího testu, kde je detekce obličeje nedostatečná pro běžné využití v LEGO robotice.

	true positive	false positive	přesnost
sledování obličeje	12	48	25%
sledování očí	20	40	50%

Tabulka 3.15. Výsledky testů pro módy sledování obličeje a očí.

3.9.4 Nedostatky softwaru

Velkým problémem současného stavu softwaru pro NXTCam5 je, že hodnoty bounding boxů nevrací hodnotu spolehlivosti (confidance rate). Dále je nastavení na detekci osmi objektů fixní a chybí možnost změny IoU.

3.10 I2C scanner

V mnoha případech se při změně i2c adres na různých senzorech stává, že není změněná adresa naznačena a při pozdějším použití nelze daný modul nalézt. Z tohoto důvodu byla

vytvořena funkce **i2cScanner()**, která vrátí název a adresu všech připojených modulů na EV3 kostce. Tato funkce je spíše "quality of life" podporou, ale pro mnoho případů je velice užitečná.

Principem funkce je proskenování všech 128 adres na každém portu. Na každé adrese se otestuje inicializace I2CDevice. Za předpokladu, že je adresa nalezena, tak je přečten název modulu, který je vždy uložen na registrech $0x10-0x17$. Následně je tato informace dekódována a převedena do hexadecimální soustavy. Tato hodnota je vrácena uživateli.

```
def i2cScanner():
    for p in [Port.S1, Port.S2, Port.S3, Port.S4]:
        print("------Testing", p,"------------")
        for i in range(0, 128, 2):
            try:
                test = I2CDevice(p,(i)>>1)
                print(test.read(0x10, 8).decode("utf-8") )
                print("Sensor_on:",hex(i))
            except:
                pass
```

Kapitola 4
Příklady pro využití rozšiřujících modulů

V této kapitole je vytvořena ukázka využití několika funkčních aplikací s využitím vytvořených knihoven. Hlavním zaměřením není složitost řešení, nýbrž praktická ukázka funkčnosti. Všechny tyto příklady se zabývají právě NXTMMX, NXTCam5, LED Matrixem a jejich kombinacemi.

4.1 Sledování černé čáry s využitím NXTMMX

Nejčastěji využívanou metodou na EV3 robotech je sledování černé čáry pomocí PID regulátoru. Pro pohon motorů bylo využito zapojení pomocí jednoho multiplexeru.

V první části ukázkového programu je vytvořena inicializace pro EV3 kostku, NXTMMX, LED Matrix a barevného senzoru. V rámci inicializace jsou vypsány parametry jednotlivých Mindsensors.com modulů, tedy verze firmwaru, název zařízení a pro NXTMMX i stav baterie.

```
ev3 = EV3Brick()
mux = NXTMMX(Port.S3,0x6)
print("NXTMMX_Connected.")
print(mux.GetFirmwareVersion())
print(mux.GetDeviceId())
print(mux.get_voltage())
print("-----------------")
matrix = EV3Matrix(Port.S3,0x22)
print("EV3_Matrix_Connected.")
print(matrix.GetFirmwareVersion())
print(matrix.GetDeviceId())
print("-----------------")
color_sensor = ColorSensor(Port.S2)
```

Následně je spuštěn LED Matrix a zvuk EV3 kostky jakožto indikátor inicializace. Dále jsou přidány potřebné proměnné pro funkčnost kódu.

```
matrix.full()
ev3.speaker.beep()
Kp = 1.2
Ki = 0
Kd = 0.6
I = 0
previous_error = 0
base_power = 40
threshold = 22
matrix.clear()
```

V nekonečném cyklu je ukázkový příklad PID (V tomto případě PD, jelikož je hodnota integrační složky 0) regulátoru s výstupem na motory připojené přes NXTMMX.

```
    while True:
        light_sensor_value = color_sensor.reflection()
        error = light_sensor_value - threshold
        P = error
        I = I + error
        D = error - previous_error
        previous_error = error

        correction = int((P * Kp) + (I * Ki) + (D * Kd))
        left_motor = base_power + correction
        right_motor = base_power - correction

        mux.motor1_run(left_motor)
        mux.motor2_run(right_motor)
```

Tato ukázka je z většiny stejná jako klasický PID regulátor bez využití mutiprexorů, ale je na ní ukázána funkčnost zapínání motorů na neomezenou dobu s indikací pomocí LED Matrixu. Značným problémem je čas, který trvá I2C komunikaci. Z tohoto důvodu je regulátor pomalejší, než při klasickém zapojení přes standardní vstupy motorů. Druhou ukázkou je daisy chain připojení LED Matrixu přes výstupní NXTMMX port.

4.2 Sledování objektů pomocí NXTCam5

V tomto příkladě je ukázána detekce objektů u NXTCam5 a její využití pro sledování pomocí robota se zapojením přes NXTMMX. Cílem je otáčení robota tak, aby pomocí P regulátoru udržoval objekt na středu.

- **Poznámka**: Pro zkrácení ukázek byla vynechána inicializace, která je stejná jako v první ukázce.

Program si přečte parametry nejbližšího objektu před sebou a vypočte horizontální střed předmětu. Tuto hodnotu se snaží regulovat k hodnotě středu záběru. Výsledná hodnota regulace je přenesena na rychlost motorů.

```
    Kp = 0.4
    real_middle = 256 / 2

    while True:
        time.sleep(1)
        info = camera.get_1st_object_info()
        object_middle = (info[1] + info[3]) / 2

        P = object_middle - real_middle

        correction = int((P * Kp))
        speed_left_motor =  correction
        speed_right_motor = - correction

        mux.motor1_run(left_motor)
        mux.motor2_run(right_motor)
```

U NXTCam5 využívá OpenMV Cam software, který je pro NXTCam5 stále ve vývoji. Z tohoto důvodu není detekce předmětů ideální a ukázka funguje jen při specifických podmínkách, viz 3.9.2.

4.3 Detekce obličeje pomocí NXTCam5

Další experimentální ukázkou je detekce obličeje pomocí NXTCam5 s využitím LED Matrix. Cílem tohoto příkladu je seznámení s nastavením jiných sledovacích módů, konkrétně Face Tracking a indikací na LED Matrixu. Stejně jako v předchozím příkladě i zde je funkčnost omezená a vhodná pouze jako ukázka ve zvláštních podmínkách.

Funkce nastaví sledovací mód na detekci obličeje a poté si v nekonečném cyklu načítá počet detekovaných objektů. V případě, že je hodnota větší než 0, tedy, že je nalezen obličej, tak program zobrazí na LED Matrixu písmeno *"X"*, v opačném případě je zobrazeno *"O"*.

```
camera.select_face_tracking_mode()
    while True:
        data = camera.count_objects()
        print(data)
        if data > 0:
            matrix.text(0,'X',1)
        else:
            matrix.text(0,'O',1)
        time.sleep(1)
```

4.4 Vypisování textu na LED Matrix

V tomto triviálním příkladu je ukázáno, jak zapisovat string postupně na LED Matrix. Hodnota uložená v zaslané zprávě, je písmeno po písmenu zobrazována na LED Matrix s funkcí *wait*, která je nastavena tak, aby lidské oko stihlo zaregistrovat jednotlivá písmena.

```
message = "HI_CVUT!"
    for m in message:
        matrix.text(0,m,1)
        wait(500)
```

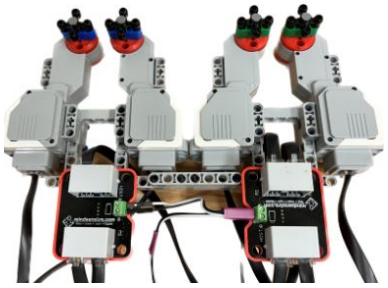
4.5 Ukázka funkčnosti NXTMMX

Tento příklad je ukázkou většiny možných využití multiplexerů. Je na něm předvedeno jak daisy chain zapojení dvou NXTMMX, tak i paralelní spouštění motorů.

První částí pro test je nutné změnit adresu jednoho z NXTMMX na novou. Námi zvolená hodnota je $0x08$ tedy je volána funkce:

```
mux.change_address(0x08)
```

Jakmile je adresa změněna a program vrátí na výstupu úspěšné potvrzení změny, tak lze využít tento multiplexor společně s dalším adresovaným $0x06$. Pro ukázku bylo využito zapojení z obrázku níže. Multiplexery jsou připojeny do portu 3 na EV3 kostce a také na externí baterii.



Obrázek 4.1. Testovací modul. [13]

První částí testovacího programu je inicializace obou NXTMMX na jejich příslušných portech, tedy na $0x06$ a $0x08$ a vypsání jejich parametrů (verze firmware, název zařízení a stav baterie).

```
mux = NXTMMX(Port.S3,0x6)
print(mux.GetFirmwareVersion())
print(mux.GetDeviceId())
print(mux.get_voltage())

mux2 = NXTMMX(Port.S3,0x8)
print(mux.GetFirmwareVersion())
print(mux.GetDeviceId())
print(mux.get_voltage())
```

První testovací funkcí je **motor_run_time()**, která spustí všechny motory rychlostí 50% na pět sekund. Pro první tři motory je přidán *wait* na *False*, který docílí spuštění motorů najednou.

```
print("All motors running on timed.")
mux.motor1_run_time(50, 5, wait = False)
mux.motor2_run_time(50, 5, wait = False)
mux2.motor1_run_time(50, 5, wait = False)
mux2.motor2_run_time(50, 5)
```

Druhým testem je **motor_run_until_stalled()**, která zapne první motor a čeká, dokud je schopen se otáčet. Pokud je zastaven (například uživatelem), tak se zastaví a funkce spustí další motor.

```
print("Run stalled one by one.")
mux.motor1_run_until_stalled(50)
mux.motor2_run_until_stalled(50)
mux2.motor1_run_until_stalled(50)
mux2.motor2_run_until_stalled(50)
```

Třetím testem je *motor_tank_move()*, který na neomezenou dobu pustí motory požadovanou rychlostí současně. Tato funkce je spuštěna na obou multiplexorech a po dvou vteřinách zastavena.

```
print("Tank move for both muxes for 2 seconds.")
mux.motors_tank_move( -100, 100)
mux2.motors_tank_move( 100, -100)
wait(2000)
mux.motors_stop()
mux2.motors_stop()
```

Čtvrtým testem je *motor_run_angle()*, která otočí na každém NXTMMX prvním motorem o 90°.

```
    print("Run angle motor 90 degrees.")
    mux.motor1_run_angle(50, 90)
    mux2.motor1_run_angle(50, -90)
```

Pátým testem je **motor_run_rotation()**. Zde jsou vyresetovány hodnoty enkoderů pro všechny motory z důvodu posledního testu a poté je otočeno motory postupně po jedné až čtyřech otočkách.

```
    print("Reset encoders and run motors for 1,2,3,4 rotations.")
    mux.reset_all_encoders()
    mux2.reset_all_encoders()
    mux.motor1_run_rotation(100, 1)
    mux.motor2_run_rotation(100, 2)
    mux2.motor1_run_rotation(100, 3)
    mux2.motor2_run_rotation(100, 4)
```

Poslední test je **motor_run_target()**, který otočí všechny motory zpět na vyresetovaný nulový úhel z minulého kroku.

```
    print("Encoder target to 0 for all motors.")
    mux.motor1_run_target(50, 0)
    mux.motor2_run_target(50, 0)
    mux2.motor1_run_target(50, 0)
    mux2.motor2_run_target(50, 0)
    print("Testing complete.")
```

V následující ukázce je uživatelský výstup celého testu.

```
b'V1.51\x00\x00\x00'
b'NxTMMX\x00\x00'
7770
b'V1.51\x00\x00\x00'
b'NxTMMX\x00\x00'
7770
All motors running on timed.
Run stalled one by one.
Motor 1 overloaded!
Motor 2 overloaded!
Motor 1 overloaded!
Motor 2 overloaded!
Tank move for both muxes for 2 seconds.
Run angle motor 90 degrees.
Reset encoders and run motors for 1,2,3,4 rotations.
Encoder target to 0 for all motors.
Testing complete.
```

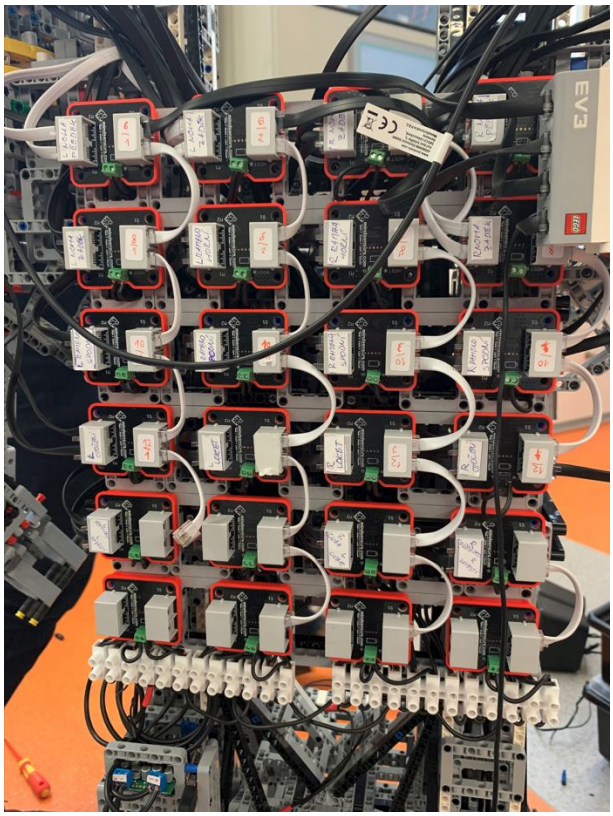
4.6 Robot Ludvík

Tato sekce je popisem reálného využití v rámci projektu Robot Ludvík. Tento 150*cm* vysoký humanoidní robot je studentským projektem Fakulty elektrotechnické a je pravidelně využíván na akcích fakulty jako jsou Dny Otevřených Dveří či GAUDEAMUS.

Jeho stavba proběhla v roce 2019 a v bakalářské práci [17] Bc. Matěje Štětky je popsán program, který robot využívá. Tento program byl napsán v grafickém programovacím prostředí LEGO® Mindstorms Education. Cílem tohoto programu je kompletní ovládání robota pomocí dálkového ovladače. Robot využívá přes čtyřicet motorů naráz, viz 4.2. Z tohoto důvodu bylo využito NXTMMX již při původní stavbě.

Softwarová část robota má závažný problém a to ten, že grafický kód pro robota je nepřehledný a kvůli omezení programu i nepřesný. Je běžnou skutečností, že některý motor není spuštěn a kvůli tomu dochází k poškození a dlouhodobému opotřebování.

Cílem knihovny pro NXTMMX je ovládání Robota Ludvíka pomocí Pythonu. Díky hotové knihovně a připraveným funkcím pro ovládání motorů je toto nyní možné. Bylo vytvořeno rozhraní pro NXTMMX takové, aby signál z dálkového ovládání byl přebírán korektně a spouštěl požadované funkce robustně. V době vzniku této diplomové práce je verze Python pro Robota Ludvíka připravena a plně funkční.



Obrázek 4.2. Zapojení NXTMMX na Robotovi Ludvíkovi. [17]

4.7 Manuál pro rozšiřující moduly

V rámci této diplomové práce byl vytvořen manuál pro podpůrné moduly EV3 od firmy Mindsensors.com s využitím Pybrics. Tento manuál i s knihovnou lze najít na stránce Github[1].

[1] `https://github.com/Audrielcz/Mindsensors_Pybricks`

Kapitola 5
Koncepce bakalářského předmětu Roboti

Druhá část této diplomové práce má za cíl úpravu předmětu bakalářského programu Kybernetika a Robotika: B3B35RO1 - Roboti. Důvodem této změny je přehodnocení kreditového systému pro obor Kybernetika a Robotika kde předmět Roboti má nově čtyři namísto původních dvou kreditů. Díky tomu se z dříve '*motivačního předmětu*' stal předmět kreditovou náročností dvakrát náročnější a přiblížil se složitostí povinně volitelným předmětům, které mají čtyři nebo šest kreditů. V rámci hodinové dotace by nyní měli studenti splnit $4x30$ hodin práce pro získání klasifikovaného zápočtu.

5.1 Co je cílem předmětu B3B35RO1 - ROBOTI

Tento předmět je určen pro nově příchozí studenty, kteří ještě nemusí mít nutný základ programování, robotiky či teorie řízení. Všechny tyto schopnosti se student naučí během bakalářského programu. Myšlenkou předmětu je využití LEGO® Mindstorms® pro sestavení základního robota v tříčlenném týmu, který má splnit zadané úlohy. Tyto úlohy by měly být zábavné a naučné. Předmět je zakončen klasifikovaným zápočtem. Proto musí být laboratorní úlohy navrženy tak, aby byl student nucen se seznámit s veškerým hardwarem (v tomto případě se stavebnicí LEGO® Mindstorms®) a také porozuměl možnostem dostupných programovacích jazyků. V neposlední řadě tento předmět rozvíjí týmové schopnosti, které jsou v programu Kybernetika a Robotika fundamentálním základem práce a úspěchu.

5.2 Koncepce předmětu v dvoukreditovém systému

Pro splnění předmětu bylo nutné získání minimálního počtu 50 bodů ze zadaných úloh, čemuž odpovídá známka E. Studenti mají ovšem možnost splnění více úloh pro dosažení až 149 bodů, které jim umožní získání známky A. Týmy studentů mají možnost práce většinou na pěti různých projektech, z kterých musí získat dostatečný počet bodů. Myšlenkou velikého bodového zisku je volitelný výběr z různých úloh.
V tabulce 5.1 je uvedeno možné bodování dané pětice úloh:

úloha	bodové ohodnocení
Sledování černé čáry	Max 51b v závislosti na dosaženém času
Sledování černé čáry a vyhýbání se překážce	Max 18b v závislosti na dosaženém času
Bludiště	Max 25b
SUMO	Max 20b
Úloha ROBOSOUTĚŽ	Max 30b + 10b za účast v ROBOSOUTĚŽI

Tabulka 5.1. Bodové hodnocení dvoukreditového systému.

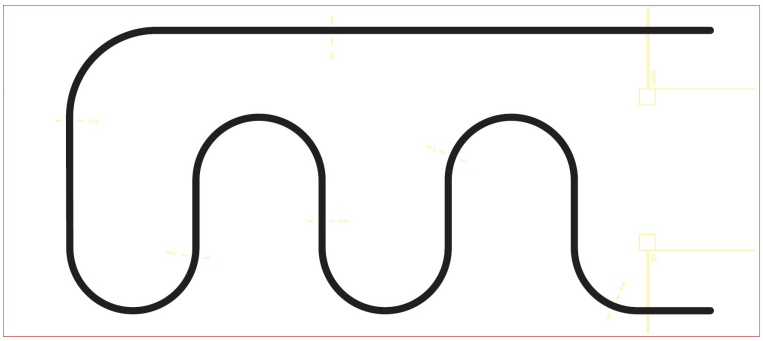
Každá úloha má svá pravidla a ustanovení. Při nesplnění jakéhokoliv bodu (pokud není uvedeno jinak) je do zápisu uvedeno, že tým v dané jízdě obdržel 0 bodů. Obecným pravidlem pro všechny úlohy je, že se všechny roboty musí po spuštění pohybovat autonomně bez jakéhokoliv vnějšího zásahu. Dále je možné postavit robota pouze z dílů v přidělených stavebnicích. Není povoleno vnějších úprav na dílech ani jakákoliv modifikace či tvorba dalších dílů, jako je například využití 3D tisku.

5.2.1 Sledování černé čáry

Cílem první úlohy je seznámení studenta se základní stavbou robota, který bude schopen sledovat černou čáru, viz obrázek 5.1. Ke splnění úlohy je nutné ujet úsek dlouhý $0-10m$ v časovém intervalu jedné minuty (časový interval se může měnit v závislosti na délce úseku). Pro získání plného počtu bodů je pak nutné naprogramovat robota tak, aby do cíle dojel v daném časovém limitu. Pro takovýto čas je nutné využití regulace. V konkrétním případě je nejvhodnějším řešením PID regulátor. Do regulátoru vstupuje intenzita odraženého světla, ze senzoru pak vystupuje akční zásah v podobě rychlosti otáčení pro oba motory.

Pravidla [18]:

- Robot musí umět sledovat černou čáru zprava i zleva.
- Robota lze do startovní pozice položit ručně a odstartovat jízdu tlačítkem. Dále se musí robot pohybovat autonomně.
- Robot se nesmí odchýlit od černé čáry o více jak $20cm$ a nesmí si zkrátit cestu.
- Měří se čas jízdy od doby kdy robot protl startovní čáru. Jízda končí v okamžiku, kdy robot protne cíl.
- Maximální čas průjezdu je stanoven na $60s$.
- V době testování robota není dovoleno jakékoliv explicitní měření rozměrů dráhy.



Obrázek 5.1. Ukázka černé čáry. [18]

5.2.2 Sledování černé čáry a vyhýbání se překážce

Pro druhou úlohu lze využít řešení konstrukce robota z jízdy po čáře 5.2.1. V této úloze si student navíc vyzkouší funkci ultrazvukového detektoru. Robot by měl sledovat čáru, ale při detekci překážky položené na čáře jí musí objet, následně najít opět černou čáru a pokračovat v jejím sledování do cíle. Hodnocení je obdobné jako v první úloze. Robot musí dojet v daném časovém limitu a počet bodů je odstupňován od dosaženého času. Až na dvě jsou **pravidla** [18] stejná jako v předchozí úloze 5.2.1:

- Maximální čas průjezdu je stanoven na 120s.
- Robot se nesmí dotknout objížděné překázky.

5.2.3 Bludiště

V třetí úloze se student seznámí se základními principy prohledávání bludiště. Robot musí projet od startu bludiště do cíle v časovém intervalu $60s$, po jehož splnění získává maximální počet bodů. Dostatečné řešení pro splnění úkolu je sledování jedné stěny. Toto zaručí průjezd bludištěm až do cíle. Možnost pro zlepšení je lokalizace robota a mapovací algoritmus. Takto pokročilé programování se ovšem vyučuje až v předmětu KUI (Kybernetika a Umělá Inteligence), takže je úloha nastavena tak, aby šla splnit s co nejmenší znalostí prohledávacích algoritmů. Nejčastějším řešením je detekce stěn pomocí ultrazvukového senzoru s využitím PID. Robot může navíc získat bonus 5 bodů při zastavení v cílovém prostoru. Celkem tak za tuto úlohu může tým získat 25 bodů.
Pravidla [18]:

- Robot může projet bludiště libovolným způsobem.
- Robota spouští soutežící stisknutím tlačítka.
- Maximální čas průjezdu je stanoven na 60s.
- Robot nesmí mít v paměti zadán explicitně plán bludiště.

Soutěžní jízda je ukončena dříve pokud [18]:

- Robot není schopen bez pomoci pokračovat.
- Robot opustí hrací plochu.
- Soutěžící se dotkne robota.
- Robot poruší pravidlo o překonávání překážek.

Za předpokladu, že je jízda ukončena dříve, jsou započítány body, které robot do daného okamžiku získal.

Obrázek 5.2. Ukázka bludiště. [18]

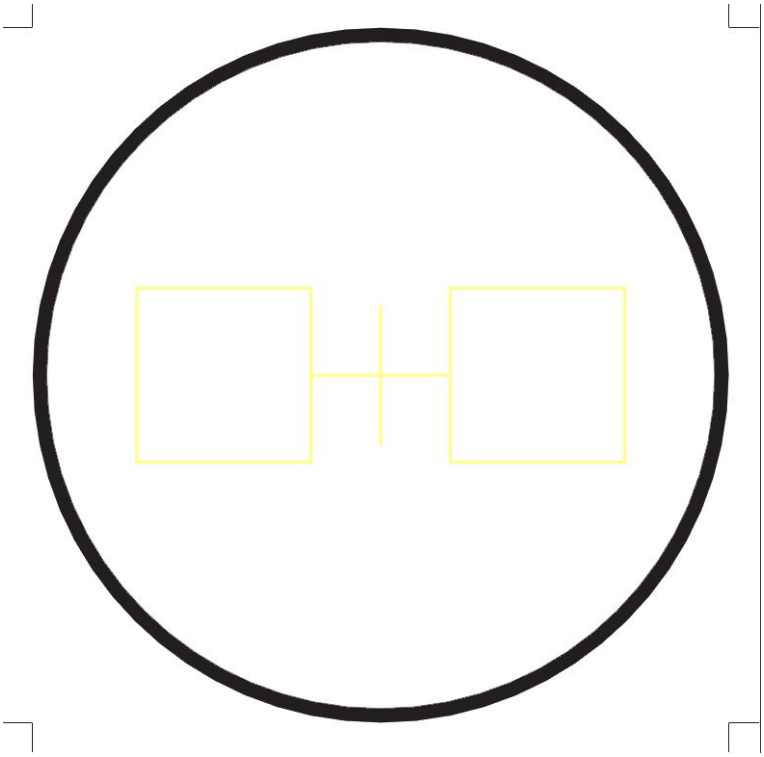
5.2.4 SUMO

V úloze sumo 5.3 je nutné vytlačit soupeřícího robota (V tomto případě je soupeřící robot nahrazen papírovou kostkou o rozměrech $25x25x25cm$) z kruhu s tím, že robot musí nadále zůstat na hrací ploše. Na splnění úkolu má robot 60 sekund. Pro řešení této úlohy je zpravidla využito světelné senzoru pro detekci černé čáry a ultrazvukový

senzor pro detekci kostky. Největším problémem u této úlohy bývá nepřesná detekce soupeřícího robota či kostky, pokud není natočena kolmo proti robotu.

Pravidla [18]:

- Robot má omezenou váhu a rozměry.
- Robot je umístěn ve startovním prostoru zády ke středu kruhu.
- Soutěžící robota naráz spustí. Robot musí vyčkat 5s a poté je spuštěn čas.
- Úmyslné naprogramování robota ke startu kratším 5s vede k okamžité diskvalifikaci.
- Robot by měl aktivně vyhledávat soupeřícího robota (či kostku).
- Doba trvání je 60s. Za předpokladu, že oba roboti (či robot a kostka) stále zůstávají na hřišti, výsledkem je remíza.
- Při souboji nesmí dojít k úmyslnému poškození robotu.



Obrázek 5.3. Ukázka hřiště na SUMO. [18]

5.2.5 Basketbal

Bonusovou úlohou každoročně bývá soutěžní úloha pro ROBOSOUTĚŽ PRO SŠ v daném kalendářním roce. Studenti mají možnost účasti ve FINÁLOVÉ ROBOSOUTĚŽI. V roce 2022 byl úlohou *Basketbal*, kde musí robot nasbírat co nejvíce míčků umístěných na hracím hřišti (maximálně 10 míčků) a umístit je do koše. Vzdálenost od koše, odkud robot míček hází, je ohodnocena bonusovými body. Za úlohu lze získat až 30 bodů + 10 bodů za účast ve FINÁLOVÉ ROBOSOUTĚŽI.

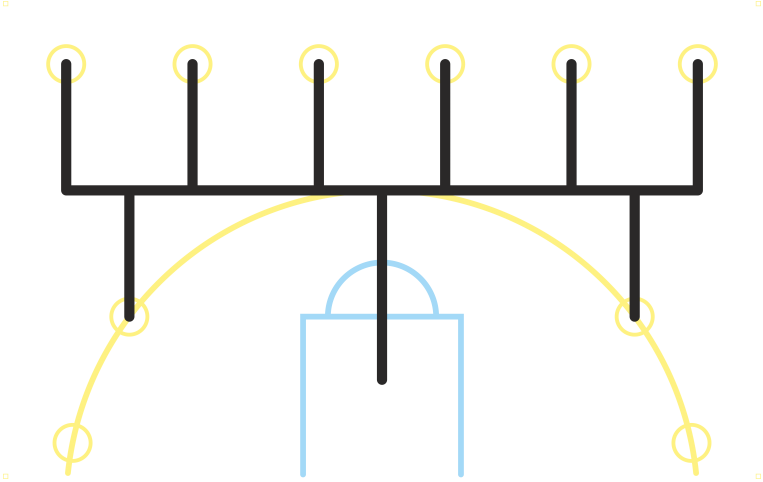
Pravidla [18]:

- Robota lze před zahájením jízdy položit na libovolné místo hrací plochy za předpokladu, že půdorys robota zasahuje do černé čáry.
- Po spuštění robota je spuštěna časomíra na 90s.
- Robot za každý vhozený míček do koše dostává 1, 2 nebo 3 body. Podle místa odhozu.

- Míček, který netrefí koš zůstává součástí hrací plochy a nelze ho odebrat, ani srovnat míčky, které byly trefeny.
- Pokud robot trefí všech 10 míčků v časovém intervalu (nehledě na počet bodů), tak získává bonus 5 bodů.
- Při zastavení v časovém intervalu, a zvukovém oznámení o této skutečnosti v časovém limitu, robot získává 3 body.

Jízda je ukončena dříve pokud [18]:

- Robot opustí hrací hřiště.
- Robot se dotkne svojí částí předmětu mimo dráhu.
- Robot není schopen pokračovat v jízdě.
- Při překročení limitu 90s.
- Soutěžící se dotkne robota.



Obrázek 5.4. Ukázka hřiště na basketbal. [18]

5.3 Nedostatky v současném řešení předmětu

Pro splnění předmětu na známku A je zapotřebí splnit pouze tři z pěti úloh. Pro studenta je výhodou, že si může jednotlivé úlohy vybrat, problémem však je nekoncepční postup díky různým řešením. To znamená, že každý student dostává po splnění jiný základ pro teorii robotiky. Dalším problémem pro čtyřkreditový předmět je časová náročnost třech potřebných úloh pro získání známky A. Většina studentů řeší úlohy Sledování čáry, Bludiště a SUMO. Časová náročnost splnění těchto třech úloh by odpovídala dvoukreditové dotaci 60 hodin, nicméně pro novou kreditovou dotaci je předmět nedostatečně časově náročný. Na druhou stranu předmět drží myšlenku motivace studentů ke studiu, takže veliký zásah do změn koncepce není ideální.

5.4 Návrh úpravy úloh

Myšlenkou je modernizace a úprava zadání laboratorních úloh. První velkou změnou pro předělání předmětu je možné využití novější stavebnice EV3. V současném předmětu je využito starší verze NXT. Tato změna přináší několik výhod, ale hlavně větší provázanost pro studenty programu KyR. V rámci prvního semestru studenti současně studují předmět ALP (Algoritmy a Programování), který obnáší základy algoritmizace

v programovacím jazyce Python. Díky využití EV3 je možné programování v MicroPython verzi pro LEGO® Mindstorms®. Toto řešení by mělo provázat oba dva technické předměty v prvním semestru a podpořilo by znalost Pythonu do dalšího studia. Jelikož Python společně s C a Matlabem je nejpoužívanějším programovacím jazykem v celém průběhu bakalářského programu, tak je toto řešení přijatelnější jak pro studenty, tak pro výuku.

Co se týče změny pro laboratorní úlohy, tak navrhovaným řešením je vytvoření postupného řešení úloh, které budou provázány v konečném semestrálním úkolu. Výhodou je nejen koncepce kontroly řešení jednotlivých studentů, ale také náznak organizace většiny předmětů v bakalářském programu, kde je standardně nutné splnit několik dílčích úloh, které jsou později využity pro splnění finálového semestrálního řešení.

Druhou možností řešení předmětu je splnění úlohy ROBOSOUTĚŽE pro SŠ a účasti na FINÁLOVÉ ROBOSOUTĚŽI. Tato možnost je pro studenty rozhodnutím, zda chtějí dělat jednu podstatně složitější úlohu, či dílčí učební úlohy. Toto je především pro studenty, kteří se již ROBOSOUTĚŽÍ zabývali při studiu střední školy a učební dílčí úlohy pro ně nejsou dostatečně zajímavé.

5.5 Python a EV3

Zásadní navrhovanou změnou pro novou koncepci předmětu je přechod na novější typ EV3. Tato změna má mnoho důvodů od vylepšení pracovních pomůcek a jejich využití, až po novější metody využité pro software. V rámci hardwaru je EV3 vybaveno novějšími a lepšími díly Technics, které dovolují studentům kreativnější řešení a navíc novější kostka s názvem EV3 je výkonnější a schopna složitějších operací. Nejdůležitějším důvodem je možnost programování celého robota v MicroPythonu, který je od roku 2018 oficiálně podporován od LEGO®. Pro programování je využito Visual Studio Code (dále VSC), které je zdarma, uživatelsky přívětivé a instalace pro EV3 spočívá pouze v instalaci podpůrné knihovny EV3 LEGO® Mindstorms®. Studenti by touto změnou měli být schopni propojit znalosti s předmětem ALP, který se vyučuje ve stejném semestru, a nemuseli by se navíc učit další dva programovací jazyky. Pro výuku v dalších semestrech toto autor shledává jako nepodstatný problém, ale v prvním semestru je mnoho studentů, kteří se na středních školách či gymnáziích programování neučili.

5.6 Návrh bodování úloh - systém hodnocení

úloha	bodové ohodnocení
1. úkol - Jízda po čáře	Max 20b
2. úkol - Dvoupatrové bludiště	Max 20b
3. úkol - Detekce barev	Max 20b
Semestrální úloha - Uklízeč	Max 40b
Účast v Robosoutěži	Splnění je klasifikováno známkou A

Tabulka 5.2. Bodové hodnocení nového systému.

Student by měl během předmětu splnit každou ze zadaných úloh tak, aby měl z prvních třech úkolů v součtu alespoň 30 bodů (50%) Následně musí splnit semestrální úlohu

minimálně na 20 bodů (50%), aby mohl dostat klasifikovaný zápočet. Splnění minima v jednotlivých úlohách se rovná potřebnému minimu pro získání klasifikovaného zápočtu hodnocení E. Takto řešená koncepce je opět stejná jako v mnoha dalších předmětech.

5.6.1 1. Úloha - Jízda po čáře

První úloha zůstává stále stejná jako 5.2.1. Cílem studentského týmu je sestrojení robota, který bude umět autonomně sledovat černou čáru dané délky.

Co tato úloha přinese studentům:

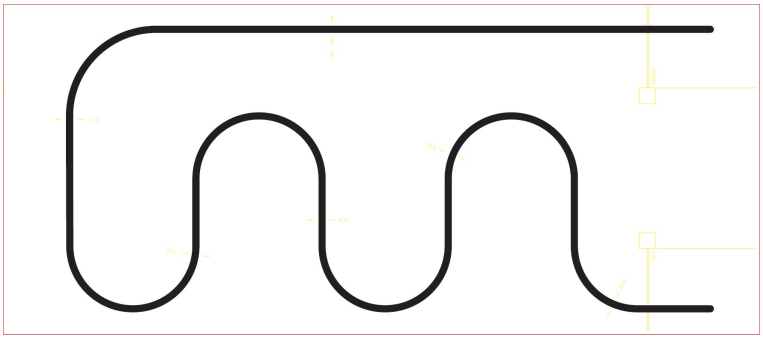
- Seznámení se stavbou robota s EV3.
- Sestavení robota, který se bude umět autonomně pohybovat.
- Základ znalostí regulace PID pro světelný senzor.

Hodnocení:

- Daný minimální čas: $20b$
- Za každých dalších 5 sekund navíc: $-1b$

Pravidla:

- Robot musí umět sledovat černou čáru zprava i zleva.
- Robota lze do startovní pozice položit ručně a odstartovat jízdu tlačítkem. Dále se musí robot pohybovat autonomně.
- Robot se nesmí odchýlit od černé čáry o více jak 20cm a nesmí si zkrátit cestu.
- Měří se čas jízdy od doby kdy robot protl startovní čáru. Jízda končí v okamžiku, kdy robot protne cíl.
- Maximální čas průjezdu je stanoven na $60s$.
- V době testování robota není dovoleno jakékoliv explicitní měření rozměrů dráhy.



Obrázek 5.5. Ukázka černé čáry. [18]

5.6.2 2. Úloha - Dvoupatrové bludiště

Cílem druhé úlohy je průjezd bludištěm. Na obrázku 5.6 je naznačen princip bludiště, které není pouze v jedné rovině, ale má nájezdy a sjezdy. Zeleně a žlutě je naznačena cesta průjezdu s tím, že pole označená písmenem N jsou nájezdy či sjezdy. Hodnoty jednotlivých polí jsou počty stavebních bloků položených na sebe.

Co tato úloha přinese studentům:

- Technické řešení robota, který bude schopen vyřešit průjezd bludištěm, které není vždy chráněno z obou stran stěnou (modře vyznačené prostory).
- Sledování stěny: hardwarové řešení pomocí dotyku o stěnu nebo regulace pomocí ultrazvukového detektoru či světelného senzoru.
- Řešení problému neideálního povrchu, který je v reálných aplikacích robotiky často řešeným problémem.

Hodnocení:

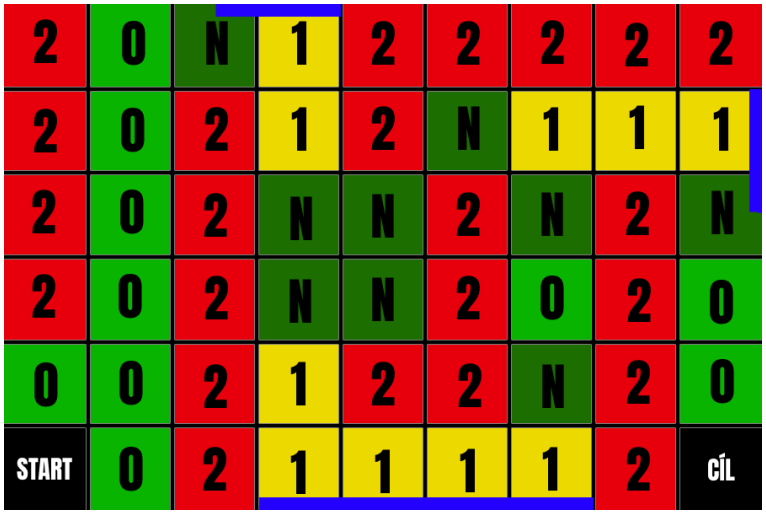
- za každé ujeté dva bloky (robot musí být svou většinou na modulu): 1 bod.
- Za dojetí do cíle: 3 body.
- Za splněný čas 90s v cíli: plný počet bodů za úlohu (V této konfiguraci tedy +5b k součtu.)

Pravidla:

- Robot může projet bludiště libovolným způsobem.
- Robota spouští soutěžící stisknutím tlačítka.
- Maximální čas průjezdu je stanoven na 90s.
- Robot nesmí mít v paměti explicitně zadán plán bludiště.

Soutěžní jízda je ukončena dříve pokud:

- Robot schopen bez pomoci pokračovat.
- Robot opustí hrací plochu.
- Soutěžící se dotkne robota.
- Robot poruší pravidlo o překonávání překážek.



Obrázek 5.6. Ukázka dvoupatrového bludiště. [13]

Poznámka: Řešení vícepatrového bludiště je důležité i kvůli technickému zpracování učebních bloků a především stolu, na kterém se bludiště staví. Pokud by nebylo zvednuto v nějakých částech o jedno patro výše, tak by nebyla možnost využití volných prostorů na hranách bludiště a úloha by byla podstatně triviálnější.

5.6.3 3. Úloha - Detekce barev

V této úloze se studenti seznámí s detekcí barev. Cílem úlohy je zjištění cílové barvy ve výběrovém prostoru a následném dojetí do cílového prostoru, kde musí robot zastavit na správně označeném místě. Takto řešený příklad motivuje studenty k využití jednoho senzoru pro více úkonů. Navíc je na plánu cest více, takže je nutné robota navigovat takovým způsobem, aby dojel do správného cílového prostoru.

Co tato úloha přinese studentům:

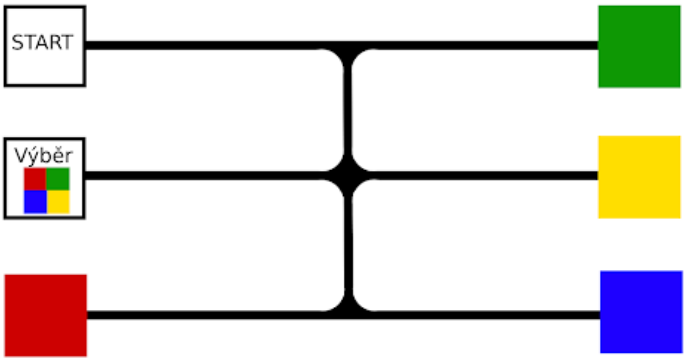
- Využití dalších funkcí světelného senzoru pro detekci nejen černé čáry, ale i RGB hodnot.
- Optimalizace kódu pro střídání světelných módů.
- Implementaci rozhodovacího algoritmu.

Hodnocení:

- Správná barva: 10 bodů
- Minimální čas: 10 bodů
- Za každé 2 sekundy přesčas: −1 bod (při správné barvě minimum 10 bodů)

Pravidla:

- Robota lze do startovní pozice položit ručně a odstartovat jízdu tlačítkem. Dále se musí robot pohybovat autonomě.
- Robot se nesmí odchýlit od černé čáry o více jak $20cm$ a nesmí si zkrátit cestu.
- Měří se čas jízdy od doby kdy robot opustil startovní blok. Jízda končí v okamžiku, kdy robot oznámí hledanou barvu v cílovém prostoru.
- Maximální čas průjezdu je stanoven na $60s$.
- V době testování robota není dovoleno jakékoliv explicitní měření rozměrů dráhy.
- Robot nesmí mít předem zadanou explicitní barvu.



Obrázek 5.7. Ukázka mapy pro detekci barev. [13]

5.6.4 4. Semestrální úloha - Uklízeč

Motivací k této úloze je automatický uklízecí robot, který by *"poházené"* předměty dovezl do příslušných *"příhrádek"* a tedy *"uklidil"* prostor. Student musí propojit všechny tři předešlé úlohy a vymyslet systém, který umožní dovoz kostek z bludiště na správně označená cílová pole. Tvar bludiště i čar je fixní, ale pozice barev v cílových polích se může měnit. Na některém z bloků o výšce 1 se bude nacházet barevná kostka, kterou robot musí dovézt do příslušného cílového prostoru. Robot musí při zvednutí kostky říci (nebo napsat na displej) barvu sebrané kostky. Za ukončení jízdy je považováno zastavení v cílovém prostoru a akustické oznámení o této skutečnosti.

Tato úloha je koncipována tak, aby student čelil problému jak ve stavbě, tak i v programování. Stavební problém je, že bude potřeba jednoho motoru na zvedání barevných kostek a zvednutí barevného senzoru. V algoritmu je poté překážkou vytvoření takového řešení, aby dokázal robot robustně plnit úlohu při různých vstupních podmínkách.

Co tato úloha přinese studentům:

- Komplexní řešení úloh.
- Využívání vlastních řešení a jejich propojení.
- Kombinaci jednoho motoru pro více úkolů.
- klasifikovaný zápočet

hodnocení	body
Vjezd do bludiště	5b
Opuštění bludiště	5b
Sebrání kostky	5b
Správně detekovaná kostka	5b
Kostka na správné barvě v cíli	10b
Zastavení robota v cílovém prostoru	10b

Tabulka 5.3. Bodové hodnocení nového systému.

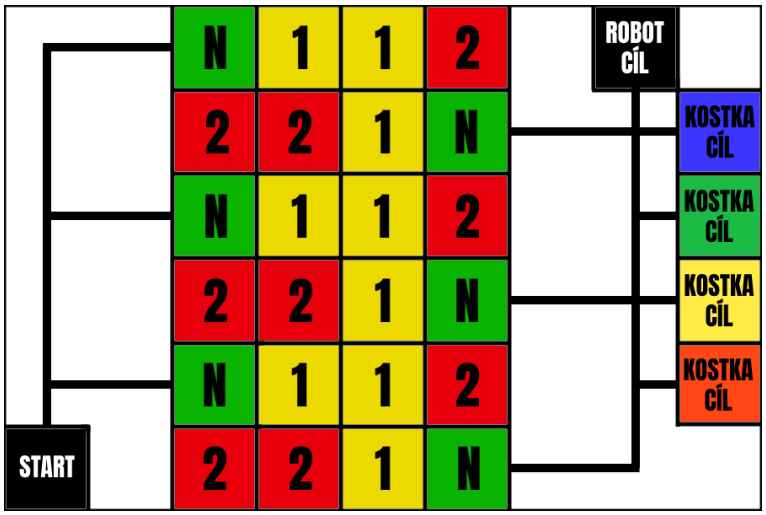
Hodnocení je vytvořeno tak, že student, který nesebere barevnou kostku, ale pouze dojede do cíle, dostává 50% bodů, tedy dostatek na udělení zápočtu. Takto hodnocená úloha je z důvodu hlavního mota předmětu Roboti, které je *"motivační předmět"*.

Pravidla:

- Robot se musí po startu pohybovat autonomně.
- Maximální časový limit na splnění úlohy je $90s$.
- Robot nesmí opustit hrací prostor.

Jízda bude ukončena dříve pokud:

- Robot není schopen pokračovat v jízdě.
- Robot opustí hrací plochu.
- Soutěžící se dotkne robota.
- Robot zastaví v cílovém prostoru v časovém limitu.



Obrázek 5.8. Ukázka mapy pro semestrální práci Uklízeč. [13]

5.6.5 Deadline úloh

Deadline úloh je koncipován tak, aby se vyhnul nejdůležitějším termínům v prvním semestru bakalářského programu Kybernetika a Robotika. V letošním roce je většina testů v 6. a pak 10.-11. týdnu výuky. Tyto termíny jsou diskutabilní a je velice pravděpodobné, že se mohou měnit.

úloha	deadline	poznámka
1. úloha	5. týden	*"až"* 5. týden - rozkoukání
2. úloha	7. týden	6. týden zápočty matematiky
3. úloha	9. týden	-
semestrální úloha	12. týden	10.-11. týden zápočty

Tabulka 5.4. Deadline jednotlivých úloh.

5.7 Úloha ROBOSOUTĚŽ

Za předpokladu, že si tým studentů vybere úlohu ROBOSOUTĚŽ, tak v daném semestru musí splnit úlohu zadanou v daném roce. Pro udělení zápočtu je nutná ukázka splnění úlohy vyučujícímu. V případě, že se tým zůčastní FINÁLOVÉ ROBOSOUTĚŽE, tak získá navíc 10 bodů. Každoroční úlohy jsou koncipované tak, že je potřeba několik desítek hodin práce pro pouhé splnění některého ze zadaných úkolů. Díky tomu lze říci, že náročnost této možnosti odpovídá podobné práci jako splnění dílčích úloh. Motivací k tomuto kroku je zlepšení kvality FINÁLOVÝCH ROBOSOUTĚŽÍ díky účasti vysokoškolských studentů.

5.8 Přednášky

Pro novou verzi předmětu byla vytvořena přednáška, která studentům vysvětlí základní principy využití EV3 a Pythonu pro robotiku. Zároveň jsou v ní vysvětleny rozdíly mezi verzemi Mindstorms® a programovacími jazyky.

5.8.1 Hardware

Část přednášky se zabývá informacemi funkčnosti EV3 kostky a možnostem jak využít jednotlivé stavební díly. Je kladen důraz na vysvětlení potřebných principů stavby robota tak, aby odpovídal potřebám pro všechny čtyři úlohy potřebné pro splnění předmětu. Jelikož se všude využívá robota na dvou kolech, tak jsou především vysvětleny způsoby, které lze využít na ovládání takového robota.

5.8.2 Software

V zbytku přednášky je vysvětleno využití programovacích jazyků pro programování EV3. Ačkoliv se student ve stejnou časovou dobu učí algoritmické znalosti v předmětu ALP, tak v předmětu Roboti je potřeba ukázat studentovi jak správně a robustně využívat jednotlivé knihovny pro ovládání robotů. Během přednášky je vysvětleno jak správně a úhledně psát kód pro robota, obeznámit ho jakým základním problémům se vyhnout (především při propojení programu s reálnými senzory) a poukázat na nepřesnost reálného světa se světem programovacím. Primárním vysvětlovaným programovacím jazykem je Python, ale lze se dozvědět jak využít i jiných programů, jako je například LeJos, Scratch atd.

Kapitola 6
Závěr

V této diplomové práci bylo diskutováno využití robotického systému EV3 s využitím Pybricks jak ve výuce, tak pro podpůrné moduly od firmy Mindsensors. Těmito moduly jsou NXTMMX, NXTCam5 a LED Matrix. Pro komunikaci se všemi moduly je využíváno I2C.

V první části této diplomové práce byla vytvořena komunikační knihovna využívající I2C. Tato knihovna je plně funkční a byla otestována na rozšiřujících modulech. Nicméně v rámci využití s vytvořenou knihovnou pro ovládání většiny Minsensors produktů nebyla tato část projektu využita a konečným řešením I2C komunikace je knihovna přímo od Minsensors, která byla vydána až po zadání této diplomové práce.

NXTMMX jsou multiplexery podporující I2C komunikaci, díky kterým je možné připojení více motorů na jeden LEGO® port u EV3 kostky. Díky tomuto řešení je možné ovládání větších robotů, které využívají více jak čtyři motory. Pro NXTMMX byla vytvořena kompletní knihovna funkcí, které jsou téměř identické s funkcemi v základní sadě Pybricks a dokáží ovládat motory na úhel, čas atd.

NXTCam5 je kamera, která je schopna detekovat objekty, hrany, obličeje a oči v reálném čase. Stejně jako u NXTMMX byla vytvořena knihovna, která umožňuje komunikaci pomocí I2C a díky tomu její ovládání přes Pybricks. Cílem této diplomové práce však nebyla úprava interního kódu kamery. Tato část se během testování stala největším problémem, jelikož zdrojový kód kamery není dostatečně odladěný. V současném stavu lze využít vytvořenou knihovnu pro kompletní komunikaci s kamerou, avšak kamera nevrací požadované hodnoty s dostatečnou jistotou.

LED Matrix je světelnou indikační maticí, která slouží pro indikaci a je zároveň designovým prvkem robota. Pro LED Matrix již existovala knihovna od firmy Mindsensors pro Pybricks, avšak neměla dostatečnou funkčnost pro potřebné projekty. Vytvořená knihovna v této diplomové práci dokáže ovládat LED Matrix s mnoha rozšiřujícími funkcemi jako je například rozsvícení či zhasnutí celé matice naráz.

Pro zjednodušení u použití rozšiřujících modulů byla vytvořena funkce, jejichž výstupem jsou všechny připojené moduly na EV3 kostce a to i s jejich adresami a jmény. Tato funkce není pro ovládání modulů nutná. Nicméně je velice často užitečná pro identifikaci modulů a to především po změně jejich základních adres.

Všechny vytvořené knihovny byly nabídnuty Mindsensors.com a budou po pár vyžadovaných úpravách využívány přímo v knihovně mindsensorsPYB.

Druhou částí bylo vytvoření nové koncepce pro předmět Roboti. Byl diskutován současný stav předmětu a byla navržena nová koncepce. Pro další roky byl navržen přechod na novější verzi LEGO Mindstorms EV3 a také přechod na programovací jazyk Python. Byly navrženy tři základní úlohy a semestrální práce. Dále bylo vytvořeno hodnocení pro jednotlivé úlohy a pro základy předmětu Roboti byly vytvořeny dvě přednáškové prezentace, které mají za účel vysvětlit studentovi základy robotiky a její programování.

Kapitola 7
Navazující práce

Při tvorbě podkladů bylo zjištěno mnoho nedostatků a možných vylepšení pro příklady využité v této diplomové práci.

- Pro zjednodušení použití NXTMMX by bylo vhodné nastavení v Pybricks, které by defaultně nastavovalo mód portu EV3 kostky na *'other-i2c'*. Bez této změny musí uživatel ručně nastavit port po každém restartu EV3 kostky. Tato možnost byla diskutována a řešení bylo nalezeno v dřívější verzi EV3DEV. Nicméně ve verzi Pybricks zatím není podpora nastavování portů. Byla zahájena komunikace s Pybricks developery. Tato funkčnost nebyla v termínu odevzdání diplomové práce implementována.
- Knihovna byla napsána způsobem stejným, jako je nastavení NXTMMX v ostatních programovacích jazycích. Tedy způsobem, kde je inicializován multiplexor a poté k přístupu k jednomu z motorů je nutné zavolat jeho funkci i s názvem motoru. Navrhovaná změna je udělání dvou *subclass*, které by se daly inicializovat pro každý motor zvlášť. Výhodou tohoto řešení by bylo přehlednější definování jednotlivých motorů, jelikož ne vždy je vhodným řešení využít mux pro párové motory.
- Dnešní jedinou možností upgradu firmwaru NXTMMX je připojení přes NXT kostku. Jelikož už není NXT po většinu času využíváno, tak by bylo vhodné vytvoření funkce pro update na EV3.
- NXTCam5 má mnoho využití, nicméně podpůrný software a hlavně jeho datové sety jsou nekompletní. Vytvoření nového řešení pro detekci obličejů a objektů.
- Přidání více vykreslovacích funkcí na LED Matrix. Knihovna vytvořená firmou Mindsensors.com je nedostatečná a nekompletní. V této práci byla vytvořena nová knihovna, která má více funkcí. Nicméně v oblasti vytvoření podpory pro komunikaci s ostatními senzory je stále mnoho nedostatků.

I přes splnění cílů této diplomové práce se objevilo mnoho problémů pro řešení do budoucna. Důvodem je nepropracovanost využívaných modulů. Avšak i přes všechny tyto problémy je výsledek práce vhodný k okamžitému využití.

Literatura

[1] Mindsensors.com. *Stránky prodejce rozšiřujících modulů.* http://www.mindsensors.com/.

[2] Brickpedia. *Historie RCX*. 2023-1-2. https://brickipedia.fandom.com/wiki/Mindstorms_RCX.

[3] BrickEconomy. *Zdroj obrázku - RCX*. 2023-1-2. https://www.brickeconomy.com/set/9709-1/mindstorms-rcx-programmable-lego-brick.

[4] Peter Peters. *Popis ARM7*. 2023-1-2. https://www.researchgate.net/figure/LEGO-platform-hardware-The-NXT-contains-an-32-bit-ARM7-microcontroller-with-256-Kbytes_fig2_229060454.

[5] Memet ÜÇGÜL. *Popis NXT*. August 2013. https://www.researchgate.net/publication/307833614_History_and_educational_potential_of_LEGO_Mindstorms_NXT.

[6] PyBricks. *Knihovna PyBricks.* https://pybricks.com/.

[7] *Zdroj obrázku - Velký motor.* https://www.zbozi.cz/vyrobek/lego-mindstorms-45502-ev3-velky-servo-motor/fotogalerie/.

[8] *Zdroj obrázku - Střední motor.* https://www.zbozi.cz/vyrobek/lego-mindstorms-45503-ev3-stredni-servo-motor/fotogalerie/.

[9] *Zdroj obrázku - Ultrazvukový senzor.* https://raisingrobots.com/product/ev3-ultrasonic-sensor/.

[10] *Zdroj obrázku - Barevný senzor.* http://www.legoengineering.com/ev3-sensors/.

[11] *Zdroj obrázku - Dotykový senzor.* http://www.legoengineering.com/ev3-sensors/.

[12] *Zdroj obrázku - Gyroskopický senzor.* http://www.legoengineering.com/ev3-sensors/.

[13] *Vlastní tvorba.*

[14] *Zdroj obrázku - LED Matrix.* http://www.mindsensors.com/ev3-and-nxt/212-led-matrix-interface-for-nxt-and-ev3.

[15] Robert Keim. *Vysvětlení I2C.* https://www.allaboutcircuits.com/technical-articles/the-i2c-bus-hardware-implementation-details/.

[16] Mindsensors.com. *Dokumentace NXTCam5*. http://www.mindsensors.com/pdfs/NXTCam5-User-Guide.pdf.

[17] Matěj Štětka. *Bakalářská práce - Robot Ludvík*. 2020. https://dspace.cvut.cz/handle/10467/87581.

[18] FEL CVUT. *Předmět Roboti*. https://moodle.fel.cvut.cz/course/view.php?id=7163.