České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídící techniky

Uživatelský manuál na použití programovacího jazyka Python pro LEGO Mindstorms EV3

User manual for using Python programming language for LEGO Mindstorms EV3

Bakalářská práce

Studijní program: Kybernetika a robotika

Vedoucí práce: ING. MARTIN HLINOVSKÝ, PH.D.

Adam Jáneš

Praha 2022

Vložení zadání práce bez podpisů

Prohlášení

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

Praha, 20.5.2022

Podpis autora práce

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Martinu Hlinovskému, Ph.D. za poskytnutí robota EV3 a soutěžních plánů, pro testování kódů v Pythonu.

Abstrakt: Cílem této bakalářské práce je seznámit studenty prvního ročníku bakalářského studia programu Kybernetika a robotika s LEGO MINDSTORMS EV3 a vytvořit instalační a uživatelský manuál programovacího jazyka Python. Využití programovacího jazyka Python demonstrováno na třech regulačních úlohách. Součástí této bakalářské práce jsou výukové materiály pro podporu výuky předmětu Roboti, který je součástí povinných předmětů prvního semestru studijního programu Kybernetika a robotika na Fakultě elektrotechnické ČVUT v Praze. V neposlední řadě vzniknout webové stránky, které budou fungovat jako dokumentace knihovny Pybricks v češtině.

Klíčová slova: Lego MINDSTORMS EV3, ev3dev, micropython, Pybricks, P regulátor, PI regulátor, PD regulátor, PID regulátor, stavový automat, normalizace

Abstract: The purpose of my Bachelor thesis is acquaint students with LEGO MINDSTORMS EV3 and create installation manual and user manual of using Python on EV3. Three control tasks will demonstrate use of Python on EV3. The part of Bachelor thesis is create educational material which supports compulsory subject Roboti in the first semestr in Kybernetic and Robotic study. Last but not least, a website will be created as documentation of library Pybricks in Czech language.

Key words: Lego MINDSTORMS EV3, ev3dev, micropython, Pybricks, P controller, PI controller, PD controller, PID controller, state machine, normalization

Obsah

Seznam tabulek

Seznam obrázků

Úvod

V roce 1998 byla poprvé společností LEGO představena programovatelná kostka označena RCX. Tato zkratka označovala Robotics Command eXplorers. K první generaci programovatelné kostky byly dodávány 2 servomotory, 2 dotykové senzory a jeden světelný senzor. Další generace byla označena LEGO MINDOSTORMS NXT, která byla vydána v roce 2006. Ke kostce bylo možné připojit a ovládat až 3 motory a 4 senzory nebo komunikovat s android zařízením. Oproti první generaci se zvýšila frekvence procesoru z 16 MHz na 48 MHz a také se navýšila paměť RAM z 32 KB na 64 KB. Kostku bylo možné programovat pomocí grafického prostředí NXT-G. V tomto prostředí se programuje pomocí přetahování a spojování bloků různých funkcí. Grafické prostředí je přívětivější pro ty, kteří se zatím s programováním nesetkali, a proto je tato varianta pro ně více intuitivní. Velkou nevýhodou grafického prostředí je nepřehlednost velkých projektů.

Předposlední vydanou generací programovatelných kostek LEGO MINDSTORMS je LEGO MINDSTORMS EV3, která byla vydána v roce 2013. Kromě navýšení frekvence procesoru na 300 MHz a kapacity paměti RAM na 64 MB došlo ke zlepšení konektivity. Oproti NXT dokáže kromě komunikace s android zařízeními komunikovat se zařízeními, které pracují na operačním systému iOS. Navíc EV3 podporuje standardy WiFi a Bluetooth 2.5.2. (1) a bylo vylepšeno drátové rozhraní v podobě portu USB typ A a slotu pro micro SD kartu. Zlepšená konektivita je podpořena i softwarově v podobě operačního systému ev3dev. Operační systém ev3dev je založen na Linuxovém jádře a postavený na distribuci Debian. Díky operačnímu systému je možné využívat vyšší programovací jazyky jako je například Python i s jeho knihovnami. Operační systém umožňuje instalovat programy nebo pracovat se soubory. (2)

Poslední vydaná sada řady LEGO MINDSTORMS je Robot Invertor. Základem sady Robot invertor je hub, který má v sobě zabudovaný akcelerometr, šestiosý gyroskopem a LED obrazovkou složenou z 25 pixelů. Tento hub je po hardwarové stránce stejný jako SPIKE Prime hub, ale má jinou barvu a jiný firmware. Bohužel se na nové programovatelné kostce nenachází USB port a firmware nepodporuje bootování linuxu, jako je to u EV3. Kostka v sadě Robot Invertor nepodporuje ani standart WiFi ani komunikaci s jinou programovatelnou kostkou. Na druhou stranu v sadě jsou rovnou 4 střední motory, které ale mají fixně připojené kabely. Celá sada Robot Invertor je velmi podobná sadě LEGO SPIKE Prime, která má místo 4 středních motorů jen 2 střední a jeden velký motor. Navíc sada Robot Invertor obsahuje dvakrát více součástek, než obsahuje sada LEGO SPIKE Prime. (3)

Tato bakalářská práce vás provede instalací, aby následně bylo možné kostku LEGO MINDSTORMS EV3 programovat pomocí jazyka Python. V teoretické části bude popsán postup, jak vytvořit projekt a jak nahrát kód v Pythonu do programovatelné kostky EV3. Následně budou popsány nejdůležitější knihovny a funkce pro regulační úlohy. V praktické části nalezneme, jak vyřešit 3 regulační úlohy za doprovodu skriptů v Pythonu.

Teoretická část

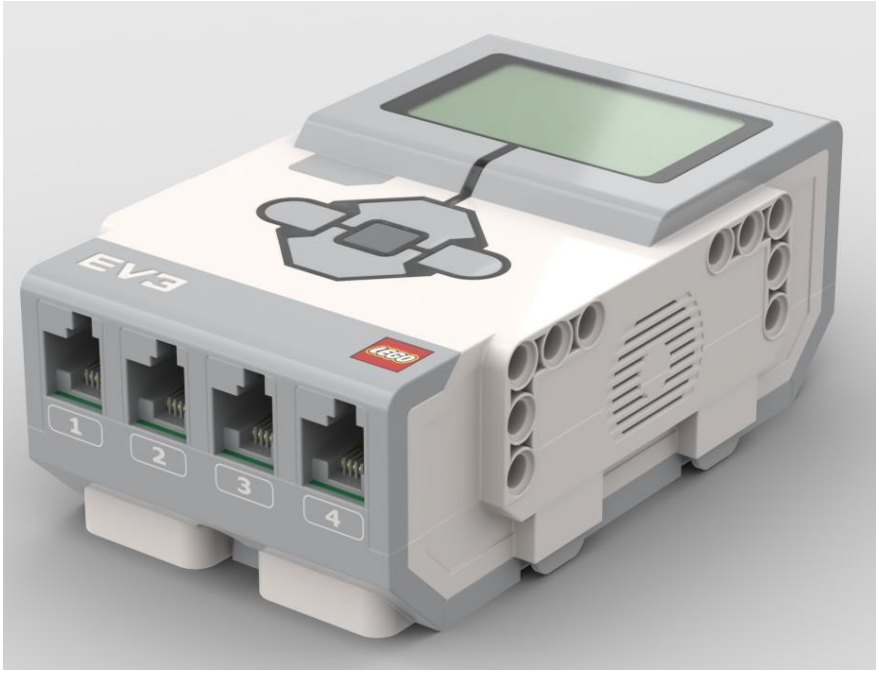
1 Obsah sady Lego Mindostorms EV3 education

Celá bakalářská práce byla testována na programovatelné kostce EV3 se sadou LEGO MINDSTORMS Education. Tato sada obsahuje programovatelnou kostku EV3, dva velké servomotory a jeden střední servomotor. Součástí balení jsou dva dotykové senzory, jeden ultrazvukový a jeden světelný senzor, který dokáže kromě odrazu světla rozeznávat i barvu předmětů, od kterých se světlo odráží. V neposlední řadě součástí sady je gyroskop a odnímatelná baterie. Pro propojení EV3 s počítačem je dodáván kabel se zakončeními USB-A a mini USB. Pro přenos informací mezi programovatelnou kostkou a senzory nebo motory jsou v balení kabely s konektory RJ12 na obou stranách. Zbytek sady jsou plastové spojovací součástky, které se vyskytují i v neprogramovatelných sadách firmy LEGO. Na webové stránce (4) nalezneme výčet všech součástek, které se nachází v základním balení LEGO MINDSTORMS Education.

Obr 1: Sada LEGO MINDSTORMS Education (5)

2 Motivace pracovat s programovacím jazykem Python

Protože programovatelná kostka EV3 má operační systém založený na Linuxovém jádře, je možné využívat vyšší programovací jazyky jako například Python. Python byl zvolen jako předmětem pro mou bakalářskou práci, protože je uživatelsky přívětivý a kostka podporuje objektově orientované programování, které je pro Python typické. Python se také vyučuje v prvním semestru bakalářského programu Kybernetika a robotika FEL ČVUT v Praze. Cílem mé bakalářské práce je vytvořit výukové materiály, které vysvětlí, jak programovat kostku EV3 pomocí jazyka Python, aby studenti prvního ročníku mohli využít nabyté znalosti programování v Pythonu z předmětu Algoritmy a programování i v předmětu Roboti, který se také vyučuje v prvním semestru bakalářského studia. Dalším úkolem mé bakalářské práce je vytvořit webovou stránku, která bude obsahovat dokumentaci knihovny Pybrics v češtině. Webové stránky jsou cílené na studenty středních a základních škol, kteří se zajímají o programování sady LEGO MINDSTORMS a o Robosoutěž, ale jejich úroveň anglického jazyka není dostatečná na to, aby porozuměli originální dokumentaci.



Obr 2: Programovatelná kostka EV3

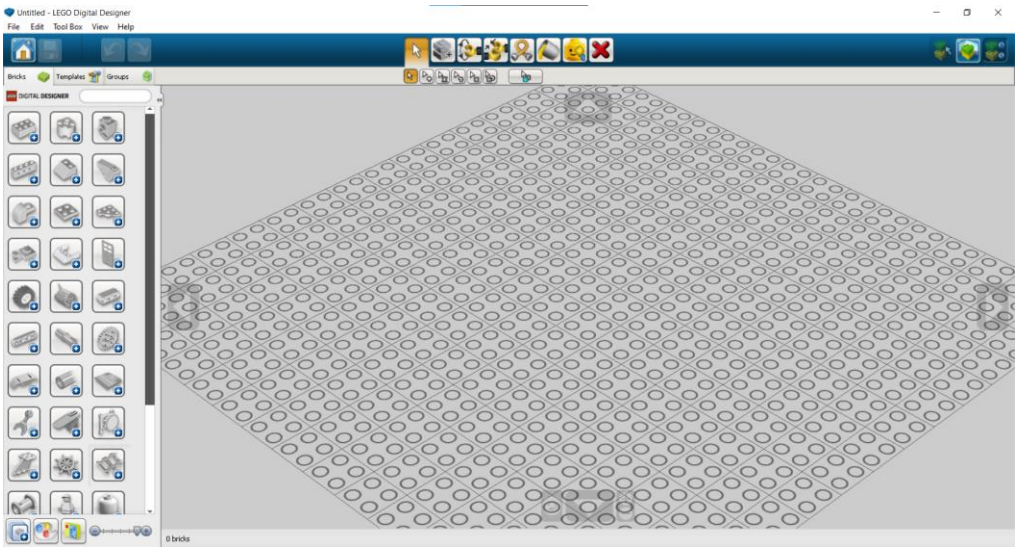
3 Lego Digital Designer a Studio 2.0

Pro vytváření digitální modelů budeme v bakalářské práci používat dvojici programů Lego Digital Designer[1] a Studio 2.0[2]. Lego Digital Designer používáme ve verzi 4.3.11 místo novější verze 4.3.12, protože verze 4.3.12 neobsahuje digitální model programovatelné kostky EV3 a její periférií. Lego Digital Designer bylo vybráno pro vytváření digitálních modelů z důvodu větší přehlednosti součástek a možného zapojení součástek do uzavřeného útvaru.

Po vytvoření digitálního modelu v Lego Digital Designer model otevřeme ve Studiu 2.0. Neexistuje oficiální podpora formátu mezi těmito programy, avšak otevření modelu vytvořeného v Lego Digital Designer ve Studiu 2.0 probíhalo jen s malými nepřesnostmi. Docházelo ke špatnému převodu modelů servomotorů a spojovacích kabelů. Tuto nepřesnost opravíme tak, že vymažeme servomotory z digitálního modelu a vložíme modely servomotorů nacházející se v nabídce Studia 2.0.

Hlavní předností Studia 2.0 je vytváření instrukčních manuálů, podle kterých lze sestavit robota. Ve Studiu 2.0 lze upravovat submodely, aby byl návod přehlednější. Navíc je možné vkládat do návodu popisky a značky, otáčet s modelem, aby provedené kroky byly lépe vidět. Jakmile po vizuální stránce upravíme návod do výsledného stavu, můžeme určit formát, ve kterém se návod vyexportuje. Považuji za velkou výhodu, že mezi podporovanými formáty je i formát PDF, protože Lego Digital Designer podporovalo pouze formát HTML, který byl nepřehledný a nedovoloval úpravu vzhledu návodu. Navíc některé kroky probíhaly nesystematicky, takže bychom se při doslovném dodržení návodu v některých případech museli vracet.

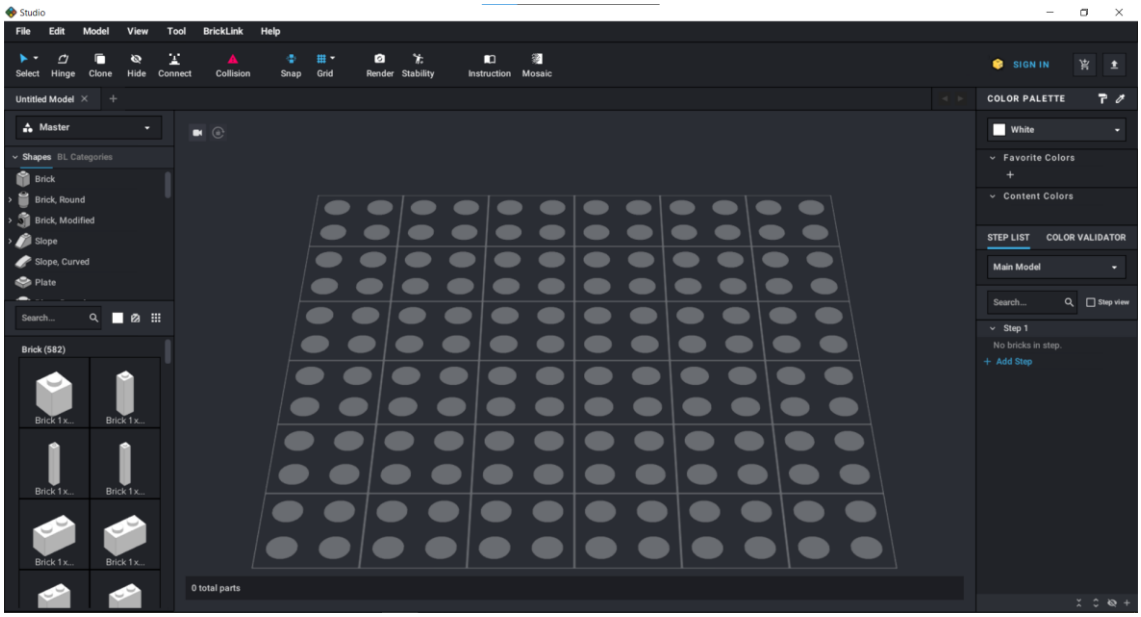
Z těchto důvodů jsem kombinoval dvojici Studio 2.0 a Lego Digital Designer. Hlavní předností Lego Digital Designer je přehledné a snadné sestavení digitální modelů z lega. Studio 2.0 bylo využíváno kvůli možnosti upravovat návod na sestavení a vybrat formát exportování. Na Obr 3 a Obr 4 nalezneme pracovní prostředí jednotlivých aplikací.



Obr 3: Prostředí Lego Digital Designer

[1] Aplikace Lego Digital Designer lze stáhnout pomocí odkazu: https://web.archive.org/web/20190622153357/https://lc-www-live-s.legocdn.com/downloads/ldd2.0/installer/setupLDD-PC-4_3_11.exe

[2] Aplikace Studio 2.0 lze stáhnout ze stránky: https://www.bricklink.com/v3/studio/download.page



Obr 4: Prostředí Studia 2.0

4 MicroPython (6)

MicroPython je odlehčená verze Pythonu určená především pro mikrokontroléry. Pro práci s MicroPythenem je potřeba minimální paměť zařízení 256 kB a 16 kB paměti RAM. Využívání MicroPythonu je možné pod licencí MIT. MicroPython je možné využívat zdarma pro osobní, edukativní i komerční účely. Velkou výhodou je možnost nahlédnout do zdrojového kódu, který je dostupný na GitHubu.

5 Instalace

5.1 Příprava

Abychom byli schopni programovat kostku LEGO MINDOSTORMS EV3 pomocí jazyku Python, budeme potřebovat:

- Programovatelnou kostku LEGO MINDSTORMS EV3
- Notebook se slotem pro SD kartu a volnou pamětí 360 MB
- Micro SD kartu s maximální kapacitou 32 GB[3]
- Redukci z micro SD karty na SD kartu
- Software pro vytvoření bootovacího média (balenaEtcher)
- Visual Code Studio

5.2 Vytvoření bootovací micro SD karty

Nejdříve stáhneme zazipovaný soubor[4], který nalezneme na stránkách LEGA o programování v Pythonu (7). Stažený soubor je obraz micro SD karty, který necháme v zazipované formě. Následně si zálohujeme veškerá data, která jsou uložena na micro SD kartě a o která bychom nechtěli přijít. Následně si stáhneme a nainstalujeme software pro vytvoření bootovacího média. V následujících kapitolách si ukážeme, jak postupovat pro aplikace balenaEtcher [5] a Rufus[6].

5.2.1 balenaEtcher

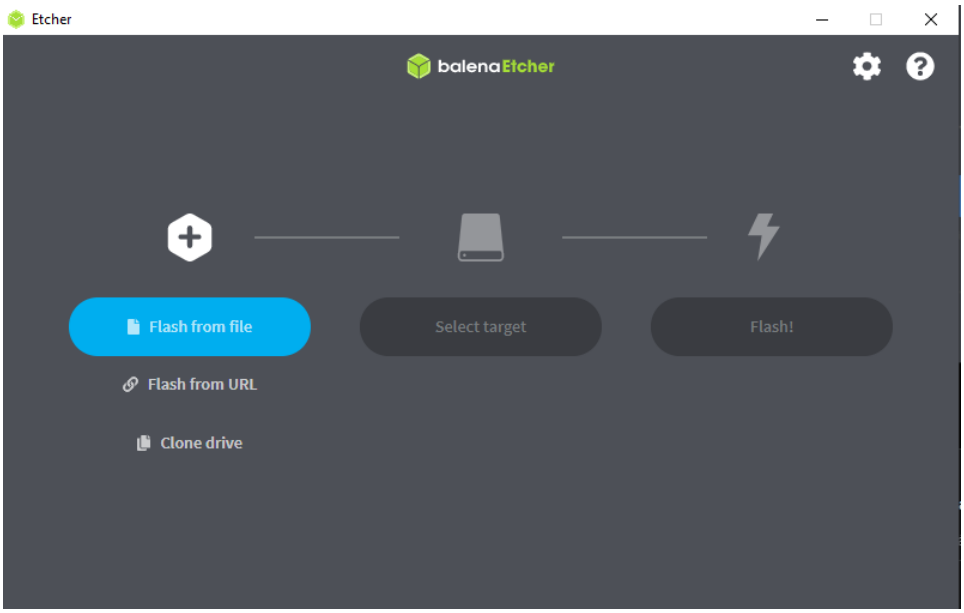
Po stažení a instalaci softwaru aplikaci spustíme a uvidíme následující okno (Obr 5):

[3] Maximální velikost úložiště micro SD karty je dle webu LEGA (20) 16 GB. Z vlastní zkušenosti jsem používal micro SD kartu o velikosti 32 i 64 GB a během testování nedocházelo k potížím způsobené micro SD kartou.

[4] Soubor lze stáhnout ze stránky lego pomocí odkazu „https://education.lego.com/v3/assets/blt293eea581807678a/blt9df409c9a182ab9c/5f88191a6ffd1b42dc42b8af/ev3micropythonv200sdcardimage.zip“

[5] Aplikace balenaEtcher lze stáhnout ze stránky https://www.balena.io/etcher/

[6] Aplikace Rufus lze stáhnout ze stránky https://rufus.ie/cs/



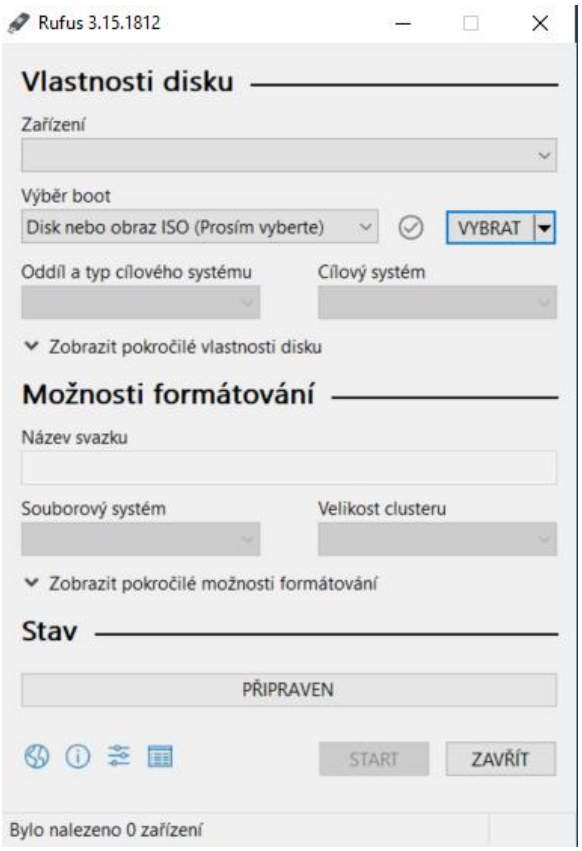
Obr 5: Spuštěná aplikace balenaEtcher

Po otevření softwaru balenaEtcher budeme postupovat následujícími kroky:

- Klikneme na tlačítko „Flash from file“ a vybereme stažený a zazipovaný obraz micro SD karty.
- Klikneme na tlačítko „Select target“ a vybereme naši připravenou micro SD kartu.
- Klikneme na tlačítko „Flash!“.
- Vyčkáme na dokončení bootování naší micro SD karty.
- Pokud bootování proběhlo v pořádku, proces bootování bude ukončen vypsáním hlášky „Flash Complete!“.

5.2.2 Rufus

Výhoda oproti aplikaci balenaEtcher je, že Rufus není potřeba instalovat. Po stažení stačí pouze otevřít stažený soubor a zobrazí se následující okno (Obr 6):



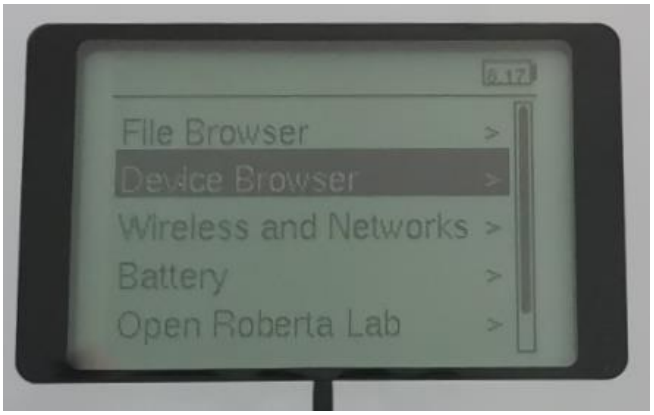
Obr 6: Spuštěná aplikace Rufus

Pro vytvoření bootovacího média budeme postupovat dle následujících kroků:

- Klikneme na pole pod nápisem „Zařízení“ a vybereme ze seznamu dostupných zařízení micro SD kartu, kterou chceme použít k bootování.
- Klikneme na tlačítko „VYBRAT“ a vybereme stažený zazipovaný soubor.
- Klikneme na tlačítko „START“.

5.3 Bootování z micro SD karty

Po vytvoření bootovacího média v podobě micro SD karty, kartu vyjmeme z notebooku i z redukce na SD kartu. Následně je doporučeno na protilehlou stranu micro SD karty od dotykových plošek nalepit pásku, která usnadní vytahování micro SD karty z programovatelné kostky EV3. Poté LEGO MINDSTORMS EV3 vypneme. Když je programovatelná kostka EV3 vypnutá, vložíme micro SD kartu do příslušného slotu a EV3 zapneme. Po zapnutí kostky EV3 se začne bootovat software z micro SD karty. Po úspěšném bootování display EV3 bude vypadat následovně (Obr 7):



Obr 7: Hlavní menu po bootování z micro SD karty

5.4 Menu EV3

Po bootování z micro SD karty uvidíme hlavní menu. Po kliknutí na možnost „File Browser“ získáme seznam nahraných projektů v kostce EV3, které následně můžeme spustit. Protože jsme zatím nevytvořili žádný projekt, tato složka bude prázdná. Po rozkliknutí druhé možnosti „Device Browser“ se zobrazí 3 položky, které lze najít na Obr 8.

Obr 8: Vedlejší menu v záložce Device Browser

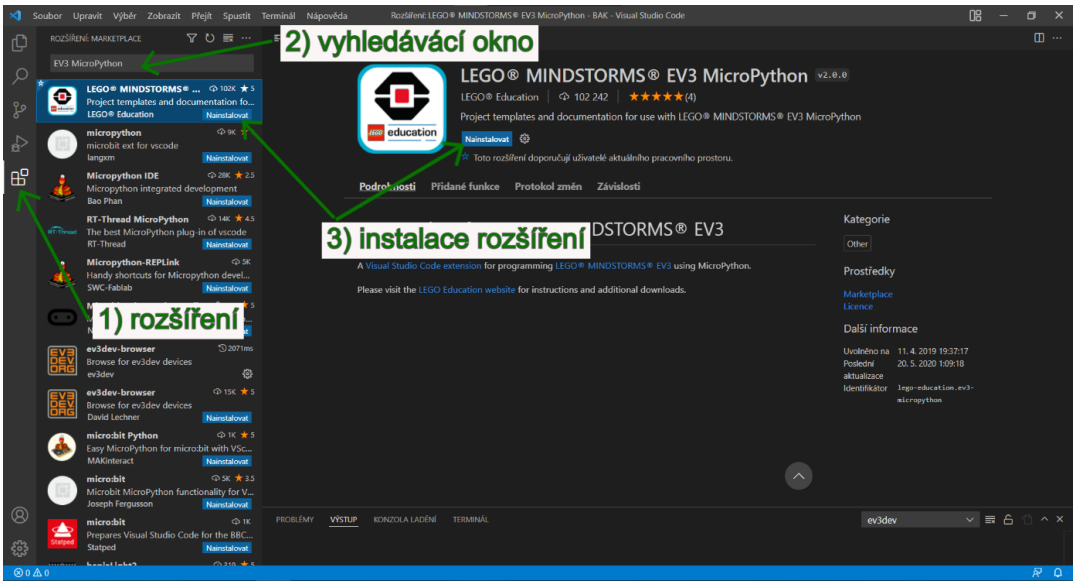
V záložce „Ports“ je možné nastavit chování daného portu. Můžeme nastavit přenos informací pomocí sběrnice I2C, v analogové podobě nebo digitální podobě pomocí uartu. Možné je také nastavit, jestli informace jsou od senzoru ze sady NXT, EV3 nebo z jiné sady. Pokud se těmito nastaveními nechceme zabývat, je zde možnost automatického nastavení.

Druhá možnost z výběru se nazývá „Sensors“. Tato záložka pracuje pouze s připojenými senzory a umožňuje získat současné hodnoty připojených senzorů. Poslední položka z výběru je „Motors“, která pracuje s připojenými servomotory.

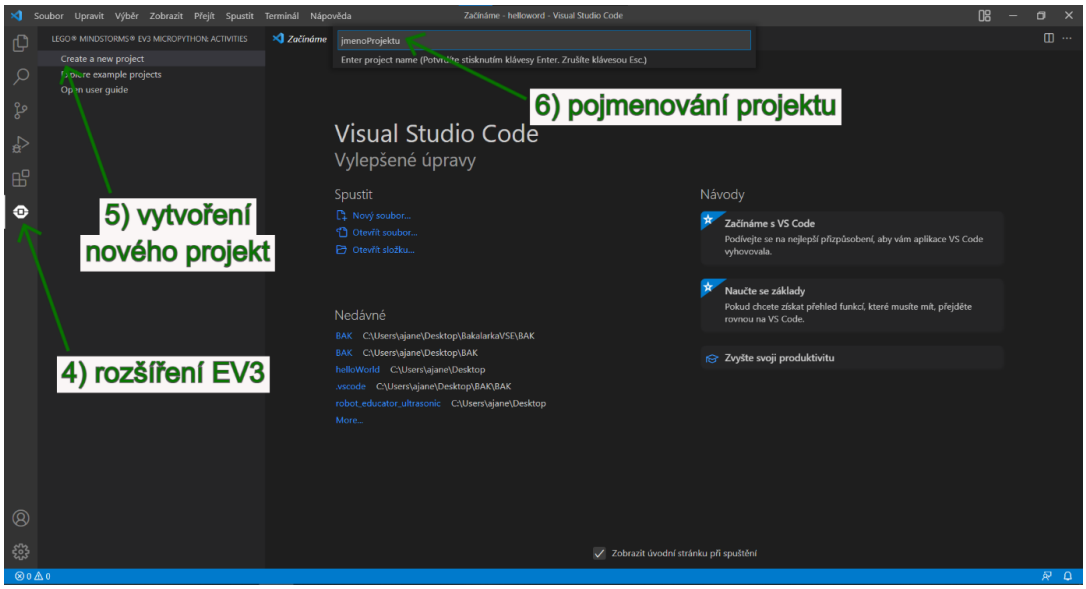
5.5 Vytvoření a spuštění souboru

Pro vytvoření skriptu v MicroPythonu, budeme využívat vývojové prostředí Visual Studio Code[7]. Po stažení a instalaci budeme postupovat dle následujících kroků, které pro větší přehlednost doprovází Obr 9 a Obr 10:

- Klikneme na tlačítko „Rozšíření“.
- Do vyhledávacího okna zadáme „EV3 MicroPython“ a potvrdíme.
- Nainstalujeme rozšíření s názvem „LEGO MINDSTORMS EV3 MicroPython“.
- Rozklikneme na levé straně ikonu pro rozšíření EV3.
- Klikneme na „Create a new project“.
- Napíšeme název projektu a potvrdíme enterem.
- Vybereme umístění, kam se budou ukládat soubory nově vytvořeného projektu.
- Rozklikneme soubor „main.py“



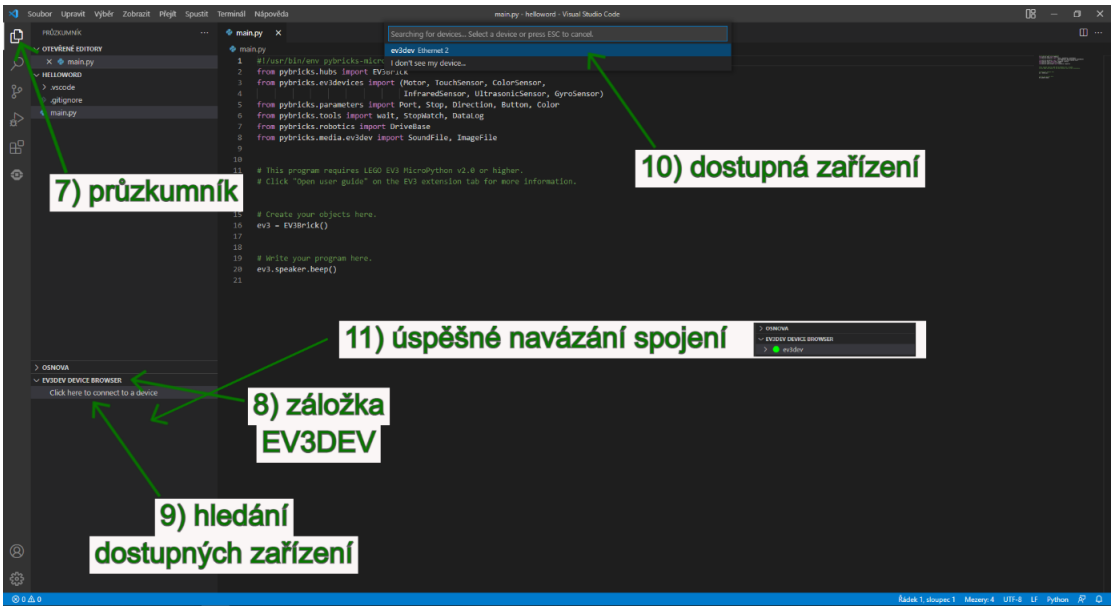
Obr 9: Znázornění postupu instalace rozšíření pro EV3



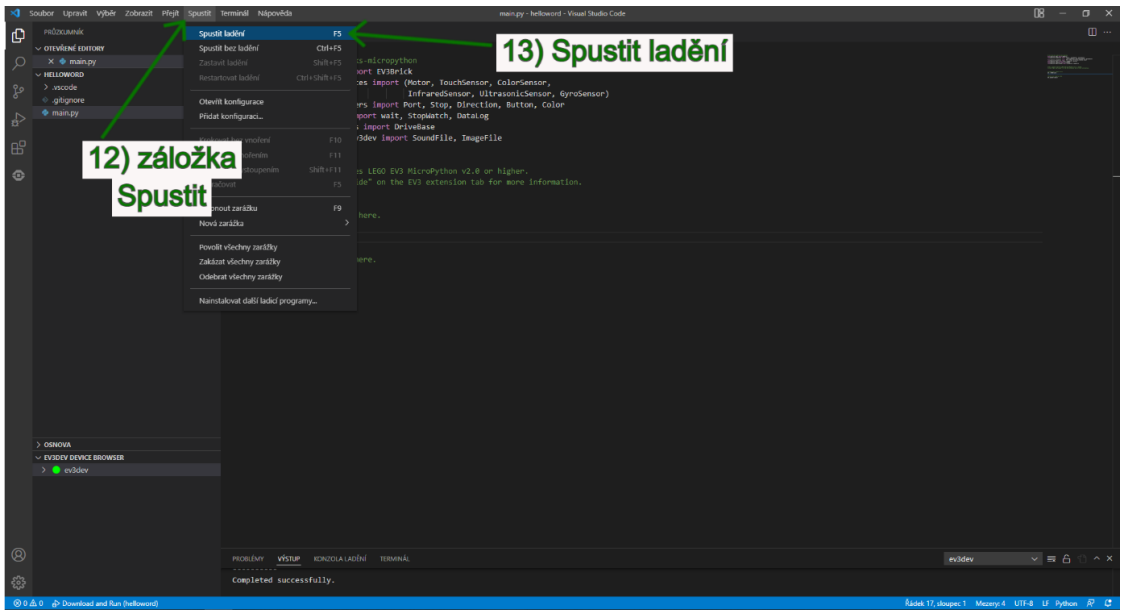
Obr 10: Znázornění postupu pro vytvoření nového projektu

[7] Visual Studio Code je možné zdarma stáhnout ze stránky https://code.visualstudio.com/.

Tento soubor je ukázkový soubor s naimportovanými knihovnami a jediným příkazem. Funkce beep() způsobí vydání zvukové efektu programovací kostkou EV3. Tento skript využijeme jako testování připojení a nahrávání skriptů. Abychom spustili skript na kostce EV3, musíme připojit zapnutou programovatelnou kostku EV3 pomocí kabelu s koncovkami USB-A a mini USB k notebooku. Dříve, než budeme nahrávat skript do kostky EV3, musíme vyhledat kostku EV3 a navázat s ní spojení. Rozklikneme záložku „Průzkumník“ a záložku „EV3DEV DEVICE BROWSER“, následně klikneme na text pro hledání zařízení a navázání s ním spojení. Nakonec mezi dostupnými zařízeními vybereme kostku EV3, se kterou chceme spojení navázat. Po vybrání naší EV3 kostky se naváže spojení. Pokud v záložce „EV3DEV DEVICE BROWSER“ nalezneme EV3 kostku se zeleným kolečkem, znamená to, že spojení bylo navázáno úspěšně. Pro následné nahrání skriptu do kostky musíme ve Visual Studio Code rozkliknout tlačítko „Spustit“ na horní liště kliknout na tlačítko „Spustit ladění“ nebo zmáčknout klávesovou zkratku *F5*. Po stisknutí se nahraje skript do kostky a provede se. Nyní jsme nechali provést náš první skript v MicroPythonu. Pro lepší přehlednost je na Obr 11 a Obr 12 znázorněn postup, jak navázat spojení s EV3 a jak nahrát skript do programovatelné kostky.



Obr 11: Znázornění postupu pro navázání spojení s EV3

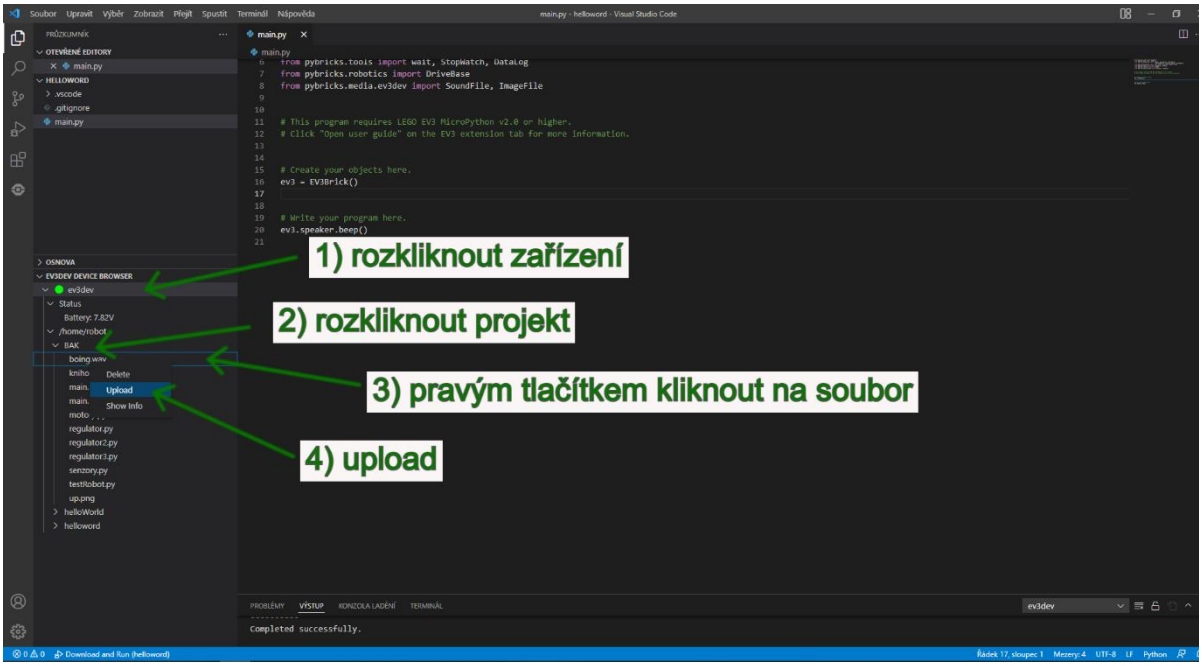


Obr 12: Znázornění postupu pro nahrání skriptu do EV3

5.6 Práce se soubory

Při pokročilejším programování kostky EV3 se naučíme, jak vytvořit soubor a jak do něj zapisovat data. Možnost zapisovat hodnoty je výhodné pro popis chování robota. Data mohou být například hodnoty ze senzorů a motorů nebo například rychlost robota. Ze zaznamenaných hodnot lze popsat chování jednotlivých senzorů a motorů. Abychom otevřeli uložený soubor a mohli procházet uložená data, nejpravděpodobněji si budeme chtít soubor stáhnout z paměti EV3 do paměti počítače.

Abychom toho dosáhli, musíme zapnout kostku EV3, propojit ji s počítačem kabelem a navázat spojení. Následně rozklikneme v levém dolním rohu nápis označující úspěšné navázání spojení s EV3. Po rozkliknutí se zobrazí seznam všech projektů, které jsou v EV3 nahrané. Po otevření projektu nalezneme jednotlivé soubory, které jsou součástí daného projektu. Pokud bychom chtěli některý ze souborů stáhnout do počítače, klikneme na soubor pravým tlačítkem a zvolíme možnost „Upload“. Nakonec vybereme umístění, kam chceme soubor uložit a naši volbu potvrdíme. Pro větší přehlednost Obr 13 znázorňuje postup, jak stáhnout soubor z kostky EV3.



Obr 13: Znázornění postupu pro stažení souboru z EV3

Většinou budeme data zaznamenávat do CSV souboru, ale můžeme například vytvořit i textový soubor. Pokud vytvoříme CSV soubor a stáhneme ho do počítače, je možné, že Excel bude mít problém s formátováním hodnot a všechny hodnoty se budou nacházet v jednom sloupci. Tento problém lze vyřešit tak, že otevřeme nový Excel soubor, klikneme na záložku „Data“ a položku „Z textu/CSV“. Následně vybereme náš CSV soubor, který jsme stáhli z kostky EV3. Nakonec se naimportují data do souboru Excel v několika sloupcích, jak bychom očekávali.

6 Pybrics

Knihovna Pybrics umožňuje snadnější programování chytrých stavebnic LEGO v MicroPythonu. Pybricks je možné použít pro LEGO Technic, LEGO City a LEGO MINDSTORMS. Hlavní výhodou je neměnnost kódu na všech výše zmíněných platformách stavebnic LEGO. Dle webu pybrics (8) díky asynchronního průběhu kódu, který je podporován i knihovnami Pybrics, dochází až ke 100 x rychlejšímu průběhu programu. Pybrics byl navrhován pro zařízení s nízkými požadavky na výkon, a proto u výkonnějších zařízení zbývá mnoho výpočetního výkonu. Navíc senzory a motory pomocí knihovny Pybrics pracují s vyšší přesností, čímž zajistí vyšší kontrolu při řízení periferií. V neposlední řadě se jedná o open source software, který má k dispozici i dokumentaci v online podobě (9).

V následujících kapitolách si popíšeme některé funkce, především jaké jsou jejich vstupní a výstupní parametry, a jaký je jejich účel. V této zprávě budou uvedeny pouze ty funkce, které jsou nezbytně důležité pro řízení robota. Pokud by popis v bakalářské práci nebyl dostatečný, komplexní popis všech funkcí nalezneme v příloze 1, která byla vytvořena jako součást bakalářské práce.

V následujících kapitolách si popíšeme některé funkce. U každé zmíněné funkce nalezneme seznam vstupních parametrů a jaký účel funkce má. Níže budou zastoupeny pouze ty funkce, které jsou nezbytně důležité pro řízení robota. Pro detailnější popis knihoven prosím navštivte oficiální dokumentaci LEGA (9), avšak celé webové stránky jsou v anglickém jazyce. Dokumentace těchto knihoven v českém jazyce vznikla jako součást této bakalářské práce a lze ji nalézt na webových stránkách (XXX). Dokumentace ve formátu PDF (příloha 1) bude dostupná na moodlu předmětu Roboti.

6.1 Parametry a konstanty

Knihovna parameters definuje konstantní parametry pro lehčí inicializaci ostatních tříd. Součástí této knihovny je možnost definovat port, směr otáčení, styl zastavení motoru, určení barvy nebo odlišení tlačítek na programovatelné kostce EV3.

6.1.1 Port

Pomocí této třídy dokážeme definovat zapojený port. Porty popsané písmeny jsou určeny pro servomotory. Na programovatelné kostce EV3 si můžeme vybrat z následujících portů pro motory {Port.A, Port.B, Port.C, Port.D}. Porty označené písmenem S a číslem jsou určeny pro senzory. Pro kostku EV3 můžeme použít následující definované porty {Port.S1, Port.S2, Port.S3, Port.S4}.

6.1.2 Direction

Třída Direction určuje kladný směr otáčení. Pokud si přejeme, aby se za kladný směr považoval směr po směru hodinových ručiček, definujeme to proměnnou Direction.CLOCKWISE. Pokud bychom chtěli, aby kladný směr byl proti směru hodinových ručiček, použijeme proměnnou Direction.COUNTERCLOCKWISE. Pokud neurčíme kladný směr, tak EV3 za kladný směr považuje směr pohybu hodinových ručiček.

6.1.3 Stop

Třída Stop určuje chování servomotorů po zastavení a obsahuje tři proměnné.

Proměnná Stop.COAST nijak nebrzdí motory, a proto je možné s malou námahou roztočit servomotor.

Druhá proměnná Stop.BRAKE po zastavení pasivně brzdí motory, a proto dokáže zamezit otáčení servomotorů, pokud na ně působí malá vnější síla.

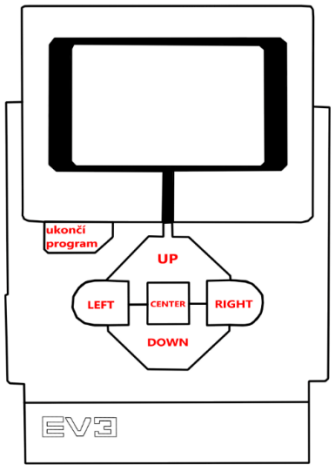
Poslední proměnná Stop.HOLD udržuje kontrolu nad motorem i po zastavení a zamezuje jakémukoliv vychýlení od cíleného úhlu.

6.1.4 Color

Třída Color definuje 9 barev, které vrací například funkce color() za pomocí světelného senzoru. Třída Color rozlišuje černou, modrou, zelenou, žlutou, červenou, bílou, oranžovou a fialovou barvu.

6.1.5 Button

Třída Button rozlišuje tlačítka, která se nachází na programovatelné kostce EV3. Třídu Button využívá například funkce buttons(), která vrací seznam zmačknutých tlačítek. Na Obr 14 nalezneme nákres, se všemi pojmenovanými tlačítky na kostce EV3.



Obr 14: Tlačítka kostky EV3

6.2 Určení času

Pro všechny níže uvedené funkce kromě wait() je nutné inicializovat třídu StopWatch. Níže vypsané funkce nám pomáhají pracovat s časem.

6.2.1 wait(time)

Funkce wait() pozastaví průběh programu na zadaný čas time v milisekundách.

6.2.2 time()

Funkce time() vrátí současný čas.

6.2.3 pause()

Funkce pause() pozastaví měření času třídou StopWatch.

6.2.4 resume()

Funkce resume() zajistí pokračování v měření času třídou StopWatch.

6.2.5 reset()

Funkce reset() vynuluje měření času třídou StopWacht. Pokud je měření času pozastaveno, funkce reset() vynuluje čas a měření bude stále pozastaveno. Pokud čas není pozastaven, funkce reset() čas se začne měřit od 0.

6.3 Třída EV3Brick

Tato knihovna zaručuje práci s tlačítky na programovatelné kostce EV3, ovládá podsvícení tlačítek, vydává zvuky pomocí reproduktoru, umožňuje zjistit informace o baterii a pracovat s obrazovkou. Pro využití všech níže zmíněných funkcí je nutné importovat a inicializovat třídu EV3Brick.

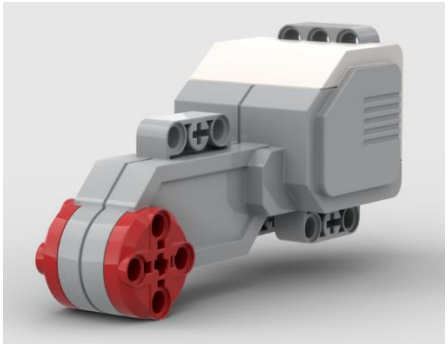
6.3.1 buttons.pressed()

Funkce buttons.pressed() zjistí, která tlačítka na kostce EV3 jsou zmáčknutá. Pokud žádné tlačítko není zmáčknuté, funkce vrátí prázdné pole.

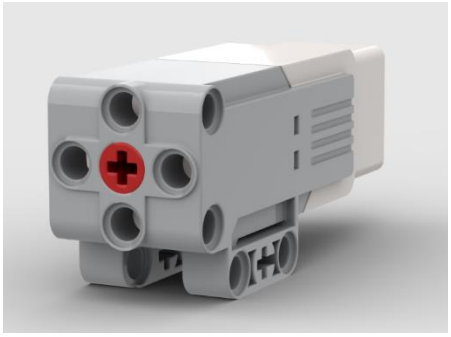
6.4 Práce s motorem

Servomotor je základní periferie programovatelné kostky EV3, která se využívá pro pohyb. Sada LEGO MINDSTORMS Education obsahuje dva typy servomotorů. Na Obr 15 lze vidět velký servomotor a na Obr 16 lze vidět střední servomotor. Dříve než začneme pracovat s motorem, musíme inicializovat třídu Motor(port, positive_direction=Direction.CLOCKWISE, gears=None).

První vstupní parametr určuje, na jakém portu je servomotor připojen. Proměnná musí být typu Port (kapitola 6.1.1). Proměnná positive_direction určuje, jestli kladný směr otáčení servomotoru je brán po nebo proti směru hodinových ručiček. Tato proměnná musí být typu Direction (kapitola 6.1.2). Poslední parametr určuje, zda je pohyb zpřevodován pomocí ozubených koleček. Proměnná musí být ve formátu pole například [5, 20]. Toto pole nám říká, že ozubené kolo spojené napevno se servomotorem má 5 zubů a otáčí ozubeným kolem s 20 zuby. Z toho plyne, že součástka spojená s větším ozubeným kolem bude mít 4 x menší úhlovou rychlost než servomotor. Převody nejsou důležité při nastavování servomotoru, ale jsou důležité při příkazech týkajících se pohybu celého robota.



Obr 15: Velký servomotor



Obr 16: Střední servomotor

6.4.1 run(speed)

Tato funkce otáčí servomotorem konstantní rychlostí speed, dokud neskončí kód nebo neproběhne jiný příkaz pro servomotor.

6.4.2 run_time(speed, time, then=Stop.HOLD, wait=True)

Tato funkce otáčí servomotorem konstantní rychlostí speed po dobu time. Po zastavení se servomotor bude chovat dle proměnné Stop (kapitola 6.1.3).

6.4.3 run_angle(speed, rotation_angle, then=Stop.HOLD, wait=True)

Tato funkce otočí servomotorem konstantní rychlostí speed o úhel rotation_angle. Po zastavení se servomotor bude chovat dle proměnné Stop (kapitola 6.1.3).

6.4.4 run_target(speed, target_angle, then=Stop.HOLD, wait=True)

Tato funkce otáčí servomotorem konstantní rychlostí speed, dokud nedosáhne úhlu rotation_angle. Po zastavení se servomotor bude chovat dle proměnné Stop (kapitola 6.1.3).

6.4.5 run_until_stalled(speed, then=Stop.COAST, duty_limit=None)

Tato funkce otáčí servomotorem konstantní rychlostí speed, dokud motor není zastaven vnější silou větší než duty_limit. Po zastavení funkce vrací konečný úhel, na kterém se servomotor zastavil.

6.4.6 dc(duty)

Funkce umožní používat motor jako jednoduchý stejnosměrný motor. Proměnná duty určuje výkon motor v procentech.

6.4.7 stop()

Funkce zastaví motor, ale dále nepůsobí silou proti změně úhlu. Zastavení je postupné, protože brzdění je prováděno pouze pomocí tření.

6.4.8 brake()

Funkce zastaví motor a dále působí pasivní silou proti změně úhlu. Zastavení je téměř okamžité, protože brzdění je prováděno pomocí tření a napětí.

6.4.9 hold()

Funkce zastaví motor a dále udržuje konečný úhel servomotoru. Zastavení je téměř okamžité, protože brzdění je prováděno pomocí tření a napětí.

6.4.10 speed()

Funkce vrací rychlost motoru ve stupních za sekundu.

6.4.11 angle()

Funkce vrací úhel servomotoru ve stupních.

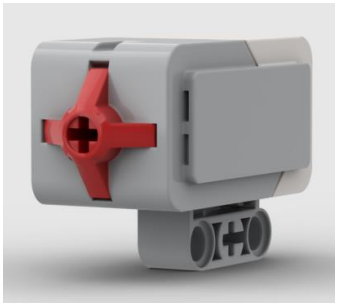
6.4.12 reset_angle(angle)

Funkce změní hodnotu úhlu na hodnotu proměnné angle.

6.5 Dotykový senzor

Dotykový senzor můžeme použít jako indikaci nárazu nebo jako spouštěč akcí. V pohyblivé části senzoru je výřez, do kterého je možné vložit další součástky ze stavebnice LEGO. Další součástky nám mohou například zvýšit plochu, ve které dokážeme náraz zaznamenat.

Před použitím dotykového senzoru je potřeba inicializovat třídu TouchSensor(port). Jediný parametr udává port, do kterého je senzor zapojen. Tato proměnná musí být typu Port (kapitola 6.1.1).



Obr 17: Dotykový senzor

6.5.1 pressed()

Funkce vrací bool hodnotu, zda je daný senzor zmáčknutý. Pokud je hodnota True, senzor je sepnutý. Pokud je hodnota False, senzor zmačknutý není.

6.6 Světelný senzor

Světelný senzor můžeme použít pro zjištění předmětu, který je před světelným senzor. Další využití může být měření okolního osvětlení a podle hodnot ze senzoru například můžeme měnit mezi světlým a tmavým režimem. Pro rozpoznání různých povrchů můžeme využívat reflexe, která není ovlivněna okolním světlem. Při rozpoznávání povrchů můžeme použít červené, modré nebo zelené světlo.

Před použitím světelného senzoru je nutné inicializovat třídu ColorSensor(port). Jediný parametr udává port, do kterého je senzor zapojen. Tato proměnná musí být typu Port (kapitola 6.1.1).

Obr 18: Světelný senzor

6.6.1 color()

Funkce vrací hodnotu v podobě barvy. Typ vrácené hodnoty Color (kapitola 6.1.4). Pokud barva není součástí seznamu barev nebo není rozpoznána, funkce vrací hodnotu None.

6.6.2 ambient()

Funkce vrací hodnotu osvětlení dopadající na senzor. Hodnota je vrácena pomocí čísel od 0 do 100. Čím je vrácená hodnota vyšší, tím více světla dopadlo na senzor.

6.6.3 reflection()

Funkce vrací hodnotu odraženého červeného světla, které senzor vysílá. Hodnota je vrácena pomocí čísel od 0 do 100. Čím je vrácená hodnota vyšší, tím více odraženého světla bylo na senzoru zachyceno. Okolní osvětlení na tuto funkci nemá téměř žádný vliv, protože světelný senzor pracuje ve spínavém režimu.

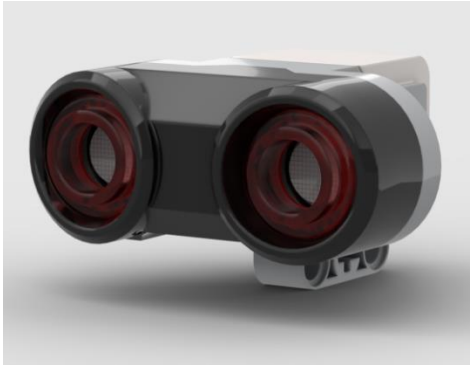
6.6.4 rgb()

Funkce vrací pole o 3 prvcích. První hodnota určuje míru zachyceného červeného světla, které senzor vysílá. Druhá hodnota určuje míru odraženého zeleného světla a třetí hodnota určuje míru odraženého modrého světla, které senzor vysílá. Všechny hodnoty pole jsou čísla od 0 do 100. Čím je hodnota vyšší, tím více odraženého světla bylo na senzoru zachyceno. Okolní osvětlení na tuto funkci nemá téměř žádný vliv, protože světlený senzor pracuje ve spínavém režimu.

6.7 Ultrazvukový senzor

Ultrazvukový senzor pracuje na principu vysílání a přijímání ultrazvukových vln. Pomocí rozdílu času vyslané vlny a času vlny přijaté dokážeme určit vzdálenost předmětu, který se před senzorem nachází.

Před použitím ultrazvukového senzoru je potřeba inicializovat třídu UltrasonicSensor(port). Jediný parametr udává port, do kterého je senzor zapojen. Tato proměnná musí být typu Port (kapitola 6.1.1).



Obr 19: Ultrazvukový senzor

6.7.1 distance(silent=False)

Funkce vrací vzdálenost v milimetrech. Proměnná silent určuje, zda po naměření se ultrazvukový senzor přepne do tiché režimu.

6.7.2 presence()

Funkce vrací hodnotu True, pokud v blízkosti je jiný ultrazvukový senzor, který vysílá ultrazvukové vlny. Pokud je hodnota False, nedochází k interferenci s jiným ultrazvukovým senzorem.

6.8 Gyroskopický senzor

Gyroskop můžeme použít pro získání náklonu robota nebo úhlové rychlosti.

Před použitím gyroskopického senzoru je potřeba inicializovat třídu GyroSensor(port, positive_direction=Direction.CLOCKWISE). První parametr udává port, do kterého je senzor zapojen. Tato proměnná musí být typu Port (kapitola 6.1.1). Druhý parametr určuje kladný směr rotace senzoru. Tento parametr musí být typu Direction (kapitola 6.1.2)

Obr 20: Gyroskopický senzor

6.8.1 angle()

Funkce vrací úhel senzoru od začátku skriptu nebo od posledního resetování úhlu. Výstupní hodnota je udávána ve stupních.

6.8.2 speed()

Funkce vrací úhlovou rychlost senzoru. Výstupní hodnota je udávána ve stupních za sekundu. Funkce by neměla být použita ve stejném programu jako funkce angle(), protože při použití funkce speed() se vynuluje úhel, a proto by funkce angle() nevracela správné hodnoty.

6.8.3 reset_angle(angle)

Funkce reset_angle() změní hodnotu úhlu na hodnotu proměnné angle.

6.9 Komplexní funkce pro pohyb robota

Níže vypsané funkce jsou užitečné pro dvoukolového robota se třetím volným kolečkem. Pomocí komplexnějších funkcí je možné ovládat celého robota jako celek. Kdybychom tuto možnost neměli k dispozici, celkový pohyb robota by musel být řízen jednotlivými servomotory zvlášť. Funkce dokážou otáčet robota na místě, jet s robotem dopředu, nastavit maximální rychlost a zrychlení. Funkce nabízí uživatelsky přívětivější vstupní parametry v podobě vzdáleností.

Pro využívání níže vypsaných funkcí je nutné správně inicializovat třídu DriveBase(left_motor, right_motor, wheel_diameter, axle_track). Vstupní parametr left_motor označuje třídu Motor() definující levý motor robota. Proměnná right_motor označuje třídu Motor() pro pravý motor robota. Parametr wheel_diameter určuje průměr kol v milimetrech, kterými servomotory otáčí. Posledním vstupním parametrem je vzdálenost v milimetrech mezi body, kde se kola dotýkají se země.

Pokud parametr wheel_diameter není správně nastavený, ujeté vzdálenosti nebudou odpovídat zadání. Pomocí funkce straight() (kapitola 6.9.1) můžeme zajistit určení průměru kol s vyšší přesností než klasickým měřením metrem či pravítkem. Pro přesnější odhad průměru kol budeme potřebovat pásmo, robota a prvotní odhad průměru kol. Po vytvoření třídy DriveBase() zavoláme funkci straight() s parametrem 1000. Z toho vyplývá, že chceme, aby robot urazil jeden metr bez zatáčení. Pro zjištění vzdálenosti, kterou robot ve skutečnosti ujel, použijeme pásmo. Pokud uražená vzdálenost je menší než jeden metr, je potřeba hodnotu průměru kol snížit a proces opakovat. Pokud uražená vzdálenost je větší než jeden metr, je nutné hodnotu průměru kol zvýšit a měření opakovat. Takto upravujeme hodnotu průměru kol, dokud nejsme s výsledkem spokojeni. (10)

Čím větší bude vzdálenost, kterou robot musí urazit, tím měření bude přesnější. Abychom při měření mohli zanedbat setrvačnost robota a dosáhnout tak přesnějších výsledků, je vhodné omezit zrychlení a rychlost robota pomocí funkce settings() (kapitola 6.9.3).

Po kalibraci hodnoty průměru kol můžeme obdobným způsobem zkalibrovat i parametr axle_track. K této kalibraci budeme potřebovat prvotní odhad vzdálenosti kol a označený bod na podložce, podle kterého dokážeme identifikovat úhel, o který se robot otočil. Po inicializaci třídy DriveBase() použijeme funkci turn() (kapitola 6.9.2) s parametrem 360. V tuto chvíli by se robot měl otočit namístě o 360 stupňů. Pokud se robot otočil o méně než 360 stupňů, je nutné hodnotu parametru axle_track zvýšit. Pokud se robot otočil o více než 360 stupňů, je nutné hodnotu proměnné axle_track snížit. Měření budeme opakovat do chvíle, než budeme s výsledkem spokojeni. Čím větší bude úhel, o který se robot musí otočit, tím přesnějších výsledků proměnné axle_track dokážeme dosáhnout. Obdobně jako v měření vzdálenosti je vhodné omezit rychlost a zrychlení otáčení robota pomocí funkce settings(). (10)

6.9.1 straight(distance)

Robot ujede vzdálenost distance bez zatáčení.

6.9.2 turn(angle)

Robot se na místě otočí o úhel angle.

6.9.3 settings (straight_speed, straight_acceleration, turn_rate, turn_acceleration)

Funkce nastaví maximální rychlost straight_speed a maximální zrychlení straight_acceleration. Nastaví také maximální rychlost rotace robota turn_rate a maximální zrychlení rotace robota

turn_acceleration. Funkce může být zavolána pouze, pokud robot není v pohybu. Pro zastavení robota je možné použít funkci stop() (kapitola 6.9.5).

6.9.4 drive(drive_speed, turn_rate)

Robot se začne pohybovat rychlostí drive_speed a otáčí se rychlostí turn_rate.

6.9.5 stop()

Funkce zastaví robota, ale po zastavení nepůsobí silou proti změně úhlu servomotorů.

6.9.6 reset()

Funkce vynuluje uraženou vzdálenost a rotaci robota.

6.9.7 distance()

Funkce vrací vzdálenost v milimetrech, kterou robot urazil od posledního zavolání funkce reset() nebo od začátku kódu.

6.9.8 angle()

Funkce vrací úhel ve stupních, o který se robot otočil od posledního zavolání funkce reset() nebo od začátku kódu.

6.9.9 state()

Funkce vrací pole o 4 hodnotách. První hodnota určuje uraženou vzdálenost v milimetrech, druhá hodnota znázorňuje rychlost robota v milimetrech za sekundu. Třetí hodnota je úhel ve stupních, o který se robot otočil od začátku kódu nebo od posledního zavolání funkce reset(). Poslední hodnota označuje rychlost ve stupních za sekundu, kterou se robot otáčí.

6.10 Záznam hodnot do souboru

Pro identifikaci motorů robota a dalších periferií je vhodné hodnoty ukládat do CSV souboru. Druhou možností je vypisovat hodnoty na obrazovku EV3 nebo na terminál, když je při průběhu kódu robot stále připojen k počítači kabelem. Avšak první možnost je více elegantní a k datům se dostaneme kdykoliv a nejen po dokončení programu. Proto součástí knihovny Pybricks je třída DataLog(*headers, name='log', timestamp=True, extension='csv', append=False), která vytvoří soubor, do kterého můžeme zapisovat data.

První parametr *headers určuje jména sloupců, která budou součástí souboru. Druhá proměnná name označuje název vytvořeného souboru. Položka timestamp určuje, zda za název bude přidána informace o datumu a čase vytvoření souboru. Pokud tato hodnota bude True, každý vytvořený soubor bude unikátní. Předposlední parametr extension určuje formát souboru, který se určuje pomocí přípony souboru. Poslední položka append určuje, zda si přejeme otevřít již vytvořený soubor a data přidávat za poslední řádek. Pokud je hodnota False, otevřeme soubor, který již existuje, ale všechna data v souboru budou smazána.

6.10.1 log(*values)

Funkce uloží jednu nebo více hodnot na nový řádek souboru.

7 Regulátory

Regulátory tvoří velmi důležitou část automatického řízení. V této kapitole si představíme Cik-Cak regulátor, proporcionální regulátor, proporcionálně derivační regulátor a proporcionálně integračně derivační regulátor. Názvy těchto regulátorů vznikly kvůli charakteristickému pohybu robota s tímto regulátorem nebo podle složek, které regulátor obsahuje. Vyjmenované regulátory jsou seřazeny podle složitosti implementace. Využití výše zmíněných regulátorů si vysvětlíme na úloze sledování černé čáry. V této úloze budeme uvažovat, že používáme světelný senzor, který nám vrací hodnoty v rozmezí 0 až 100. Hodnota 0 odpovídá ideálně černé čáře a hodnota 100 odpovídá bílému pozadí. Podle hodnot ze světelného senzoru se ovládají 2 servomotory, které určují směr, kterým se robot bude pohybovat.

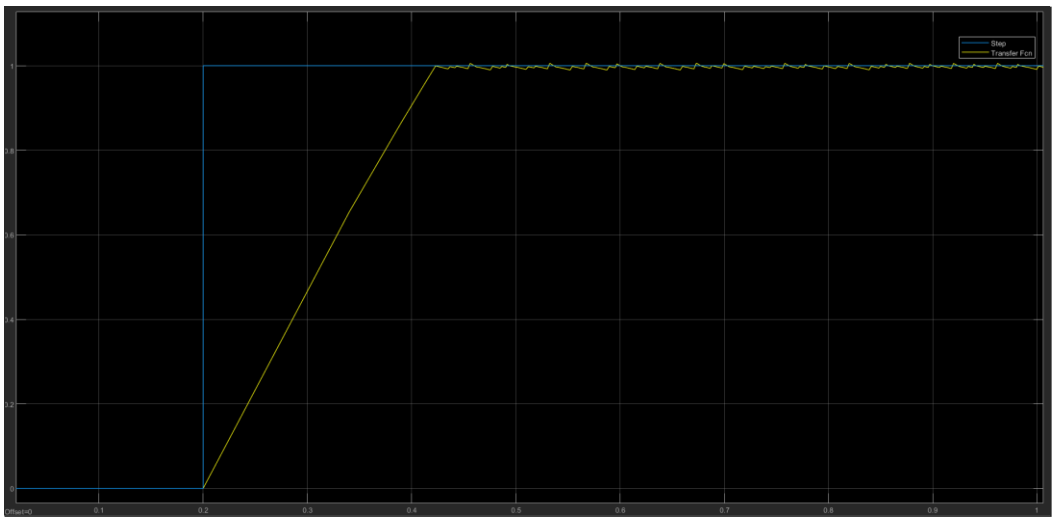
7.1 Cik-cak regulátor

Tento regulátor je pojmenován podle typické trajektorie robota při sledování černé čáry. Pohyb „Cik-Cak“ je způsoben faktem, že tento regulátor je natolik jednoduchý, že neumožňuje robotovi jet rovně.

Při sledování černé čáry chceme dosáhnout situace, že senzor nám bude vracet hodnotu 50, která označuje, že polovina senzoru zasahuje do černé čáry a druhá polovina zasahuje do bílého pozadí. Abychom tohoto stavu dosáhli, regulátor vyhodnocuje situaci tak, že pokud v senzoru převažuje bílé pozadí, robot zastaví motor blíže k černé čáře. Pokud senzor bude vracet hodnotu menší než 50, regulátor zastaví servomotor, který je vzdálenější od černé čáry, aby se senzor přiblížil k hodnotě 50.

Druhou možností je místo zastavení jednoho motoru pouze snížit jeho rychlost. Z toho plyne, že pokud v senzoru převažuje bílé pozadí, robot sníží rychlost motoru blíže k černé čáře. Toto snížení rychlosti servomotoru je konstantní a nezáleží, jak velká je odchylka od hodnoty 50. Pokud senzor bude vracet hodnotu menší než 50, regulátor sníží rychlost druhého servomotoru, aby se senzor přiblížil k hodnotě 50. Tato varianta je níže popsána, protože se robot po dráze pohybuje rychleji než v prvním případě.

Protože snížení rychlosti jednoho servomotoru je konstantní, vždy dochází k překmitu. Pokud bychom se podívali na trajektorii robota s tímto regulátorem, trajektorie bude vypadat jako pila.



Obr 21: Chování Cik-Cak regulátoru v Simulinku

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 150
zatoc=0
while True:
    #normalizace
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    if chybaP >stred:
        zatoc =15
    else:
        zatoc=-15
    rob.drive(rychlost, zatoc)
```

7.2 Cik-cak regulátor s deadzone

Abychom vylepšili chování Cik-Cak regulátoru, mohli bychom další podmínkou zajistit, že regulátor v blízkosti černé čáry nebude omezovat rychlost ani jednoho servomotoru. Této podmínce se říká „dead zone“ a určuje zónu, kde regulátor neovlivňuje ani jeden motor, a proto robot může jet i rovně.

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 200
zatoc=0
while True:
    #normalizace
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    if chybaP > (stred+5):
        zatoc =15
   elif chybaP < (stred-5):
       zatoc = -15
    else:
        zatoc=0
    rob.drive(rychlost, zatoc)
```

Pokud bychom chtěli vytvořit přesnější regulátor, mohli bychom určit například 5 intervalů. V Intervalu s nejmenšími hodnotami robot znatelně omezí rychlost servomotoru blíže k bílému pozadí. U větších vrácených hodnot senzoru ale menších než 50 by regulátor omezil rychlost servomotoru, ale snížení by nebylo tak razantní jako v předchozím případě. Kolem hodnoty 50 by se vytvořila „dead zone“, která by nesnižovala rychlost ani jednoho servomotoru. Druhý největší interval by lehce snížil rychlost

otáčení servomotoru blíže k černé čáře. Poslední interval by razantně snížil rychlost servomotoru blíže k černé čáře.

Tímto způsobem by bylo možné zvyšovat počet intervalů, které by zpřesnilo sledování černé čáry, avšak každá změna regulátoru by znamenala přepisování mnoha hodnot v podmínkách. Z tohoto důvodu si ukážeme proporcionální regulátor, který se bude chovat velmi podobně, ale bude mnohem přehlednější.

7.3 P regulátor

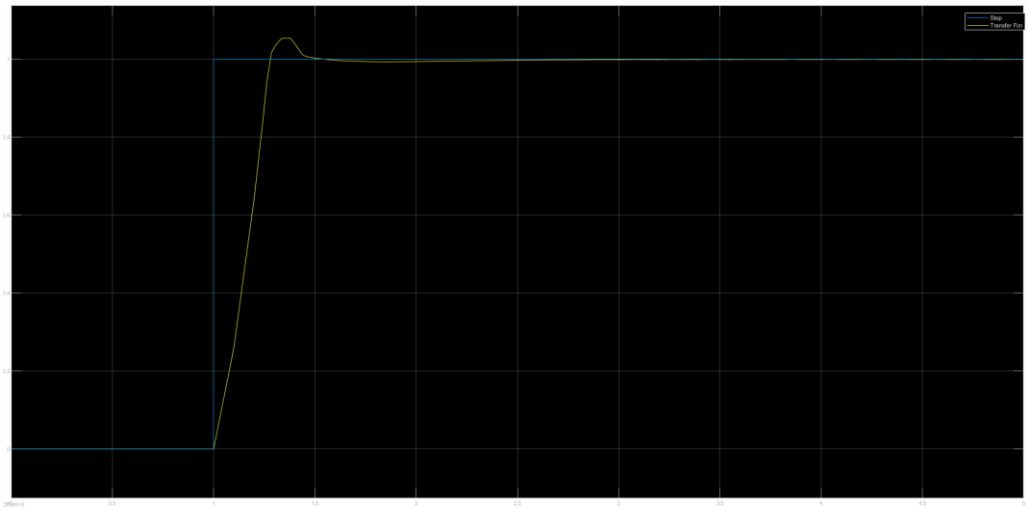
P regulátor obsahuje pouze proporcionální složku. Proporcionální složka v tomto případě je určena jako rozdíl mezi současnou a žádanou hodnotou senzoru. V našem případě je žádaná hodnota senzoru při sledování černé čáry 50. Čím je rozdíl aktuální a žádané hodnoty větší, tím i snížení rychlosti motoru bude razantnější. Lineární variantu proporcionálního regulátoru je možné implementovat tak, že rozdíl hodnot světelného senzoru vynásobíme konstantním členem a výsledek určuje rotaci robota.

Pokud bychom nechtěli používat komplexní funkce třídy DriveBase, místo nastavení úhlu celého robota bychom měnili rychlosti obou servomotorů. Představme si situaci, kde nejdříve nastavíme rychlosti obou servomotorů na hodnotu 300 stupňů za sekundu. Následně získáme hodnotu ze senzoru a vypočítáme rozdíl aktuální a žádané hodnoty. Tento rozdíl vynásobíme proporcionálním koeficientem a získáme změnu rychlostí obou motorů. Následně výsledek k jedné rychlosti motoru přičteme a od druhé rychlosti výsledek odečteme. Jelikož rozdíl skutečné a žádané hodnoty může být i záporný, vyplývá z toho, že implementace bude fungovat pro přibližování k čáře i oddalování od čáry bez nutnosti podmínek if.

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 200
zatoc=0
P=1.5
while True:
    #normalizace
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    zatoc= P*chybaP
    rob.drive(rychlost, zatoc)
```

V ideálním případě bychom dokázali ovládat robota způsobem, že po konečném čase bude dokonale sledovat rovnou černou čáru. Bohužel při ovládání reálného servomotoru dochází k jevům, že v konečném čase proporcionální regulátor nedokáže dosáhnout cílené hodnoty. Nenulová konečná chyba je způsobena tím, že každý reálný servomotor má svou „dead zone“. Tato „dead zone“ se projevuje tak, že motor se začne otáčet až od určité hodnoty napětí, neboli pro velmi malé napětí servomotor nereaguje a zůstává v klidu.



Obr 22: Chování P regulátor v Simulinku

7.4 PD regulátor

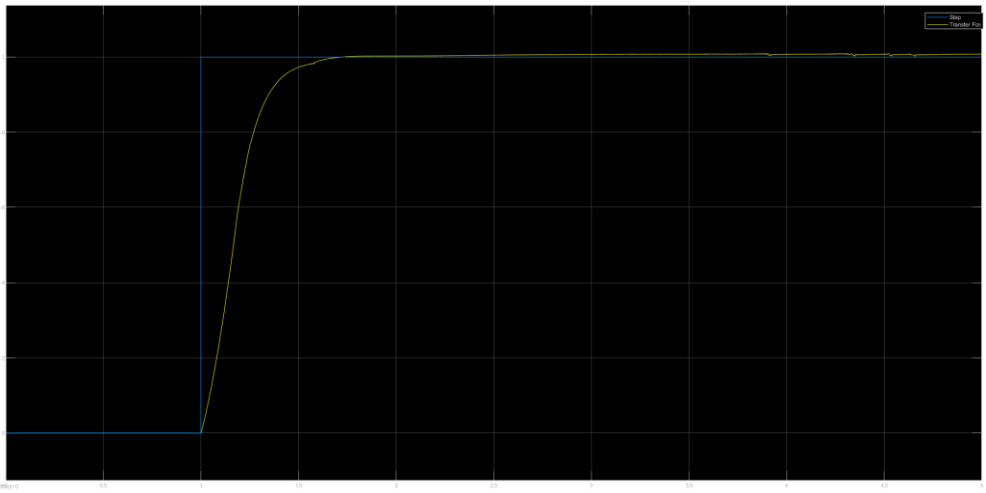
Kdybychom chtěli, aby robot dokázal projet ostřejší zatáčky, než zvládne robot se samotnou proporcionální složkou, přidáme k proporcionálnímu regulátoru derivační složku. Derivační složka pomáhá k odhadování zatáček černé čáry. Derivační složka je určena rozdílem mezi současnou a minulou hodnotou světelného senzoru. Pokud rozdíl po sobě jdoucích hodnot světelné senzorů je velký, bude i derivační složka více zasahovat do řízení motorů. Jakým dílem se bude derivační složka podílet na řízení motorů určuje konstanta, kterou je rozdíl hodnot vynásoben.

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 200
zatoc=0
P=1.5
D=0.071
staryD=0
novyD=0
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    staryD=novyD
    novyD=chybaP
    chybaD=novyD-staryD
    zatoc= P*chybaP + D * chybaD
    rob.drive(rychlost, zatoc)
```

Díky spojení proporcionální a derivační složky dokážeme rychleji dosáhnout cílené hodnoty s menším překmitem, avšak i v nekonečném čase bude chyba od cílené hodnoty nenulová, protože v blízkosti žádané hodnoty proporcionální složka nemá možnost změnit rychlost motoru, protože změna by byla velmi malá a tím pádem by zasahovala do „dead zone“ daného motoru. Pokud se nebude měnit

hodnota senzoru, rozdíl, který určuje derivační složku, bude nulový. Z tohoto důvodu dochází k nenulové chybě v nekonečném čase, protože součástí regulátoru není složka, která by zajistila nulovou chybu v nekonečném čase.



Obr 23: Chování PD regulátor v Simulinku

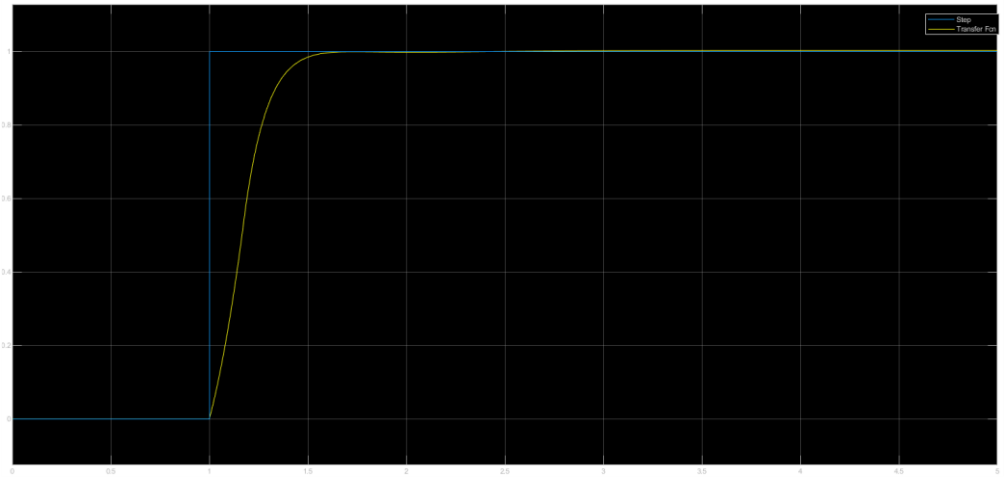
7.5 PID regulátor

V PID regulátoru se k proporcionální a derivační složce přidala i složka integrační. Integrační složka pracuje na principu sčítání hodnot proporcionálních odchylek. Čím vyšší je suma, tím více bude integrační složka zasahovat do řízení motorů.

Využití integrační složky docílíme k rychlejšímu dosažení cílené hodnoty, avšak za cenu možného překmitu. Integrační složka má další velmi důležitou vlastnost a tím je docílení nulové chyby v nekonečném čase. Pokud se nachází robot velmi blízko černé čáry, avšak senzor neukazuje ideální hodnotu, je velmi pravděpodobně, že se nacházíme v „dead zone“ servomotoru. Díky integrační složce se sčítají chyby od žádané hodnoty, avšak samotné proporcionální chyby nemají vliv na řízení. Díky sčítání těchto chyb se integrační složka dostane z „dead zone“ motoru a dokáže robota přiblížit k cílené hodnotě. Díky této vlastnosti je v nekonečném čase konečná chyba nulová.

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 200
zatoc=0
P=1.5
D=0.071
I=0.0071
chybaI=0
staryD=0
novyD=0
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    staryD=novyD
    novyD=chybaP
    chybaD=novyD-staryD
    chybaI=chybaI + chybaP
    zatoc= P*chybaP + D * chybaD + I* chybaI
    rob.drive(rychlost, zatoc)
```



Obr 24: Chování PID regulátor v Simulinku

7.6 PID regulátor s anti-windupem

Integrační složka je velmi užitečná, abychom dosáhli nulové chyby v nekonečném čase, avšak kvůli integrační složce dochází k nežádoucímu jevu zvané windup. Tento jev nastává, když naše současná hodnota je velmi vzdálená od cílené hodnoty a během přibližování k cílené hodnotě se nasčítá velká integrační chyba, která nakonec vede k velkému překmitu. Tuto situaci si můžeme představit u tempomatu v osobním automobilu. Jedeme rychlostí 50 km/h a na tempomat nastavíme rychlost 90 km/h. Auto díky tempomatu začne zrychlovat a nakonec skončí na rychlosti 90 km/h, avšak před ustálením rychlosti může dojít k velkému překmitu. Protože maximální rychlost je v daném úseku 90 km/h, nemůžeme si dovolit, aby auto vinou tempomatu překračovalo maximální povolenou rychlost.

Z tohoto důvodu se implementuje funkcionalita jménem „anti-windup“. Tato funkcionalita má za hlavní cíl zamezit překmit způsobený integrační složkou. Je to docíleno tím, že pokud je dosaženo cílené hodnoty, nasčítaná integrační chyba je vynulována, aby nedocházelo k překmitu.

```
cerna = 3
bila = 58
stred = 50
rychlost = 200
zatoc=0
P=1.5
D=0.071
I=0.0071
chybaI=0
staryD=0
novyD=0
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    staryD=novyD
    novyD=chybaP
    chybaD=novyD-staryD
    chybaI=chybaI + chybaP
    #anti-windup
    if (chybaI>20000 and chybaP<-47) or (chybaI<-20000 and chybaP>47):
        chybaI=0
    zatoc= P*chybaP + D * chybaD + I* chybaI
    rob.drive(rychlost, zatoc)
```

8 Knihovna DriveBase

Knihovna DriveBase nám umožňuje pracovat s robotem jako s celkem. Funkce straight() této třídy dovoluje po zadání vzdálenosti v milimetrech, aby robot urazil danou vzdálenost bez zatáčení. Pomocí funkce turn() je možné otočit robotem na místě o zadaný úhel. Pro úlohu sledování černé čáry je možné použít funkci drive(). Její vstupní parametry určují translační rychlost robota a rychlost rotace robota. Další funkce jsou pouze informativní a vrací hodnoty ohledně natočení a rychlosti robota.

Pro větší přehlednost jsem se výše zmíněné funkce snažil napodobit, aby každý uživatel si mohl představit, jak daná funkce probíhá. Níže nalezneme podkapitoly, kde každá kapitola popisuje jednu funkci a na konci každé podkapitoly nalezneme kód v Pythonu.

8.1 Class rizeni

V první řadě musíme vytvořit třídu. Při inicializaci je nutné zadat 4 proměnné. První dvě proměnné určují levý a pravý motor, které jsou ve tvaru třídy Motor. Třetí proměnná určuje poloměr použitých kol v milimetrech. Poslední proměnná určuje vzdálenost v milimetrech mezi levým a pravým kolem. Součástí funkce __init__() kromě výše zmíněných proměnných jsou i proměnné určující koeficienty PID regulátoru, proměnné určující maximální rychlosti, zrychlení a odchylku od požadované hodnoty. V neposlední řadě se inicializuje třída StopWatch, zaznamenává se obvod kol a délka trajektorie kola při otočení robota o 360 ° na místě. Výchozí nastavení koeficientů PID regulátoru jsou nastaveny tak, aby nedocházelo k překmitu, ale nejsou optimalizovány na nejrychlejší dosažení cílené hodnoty, avšak díky funkci pid() je možné koeficienty libovolně měnit.

```
def __init__ (self,motorLevy, motorPravy, polomer, rozvor):
        self.motorLevy=motorLevy
        self.motorPravy=motorPravy
        self.polomer=polomer
        self.rozvor=rozvor
        self.delkaKolo=2*math.pi *polomer
        self.delkaRozvor=math.pi*rozvor
        self.Kp=0.5
        self.Ki=0.0001
        self.Kd=0.02
        self.maxRychlost=150#750stupnu/s
        self.maxZrychleni=10
        self.maxRotace=30
        self.maxZrotace=100
        self.maxOdchylka=2
        self.cas=StopWatch()
```

8.2 funkce settings(rychlost, zrychleni, rotace,zrotace)

Funkce settings() se volá se 4 parametry. První parametr určuje dopřednou rychlost robota v milimetrech za sekundu. Proměnná zrychleni určuje zrychlení robota v milimetrech za sekundu na druhou. Předposlední proměnná určuje maximální rychlost robota ve stupních za sekundu. Poslední proměnná stanovuje maximální zrychlení rotace robota ve stupních za sekundu na druhou. Po zavolání této funkce se změní proměnné třídy řízení, a proto se budou zadané maximální hodnoty aplikovat na funkce straight() a turn(), které bude zavolány po provedení funkce settings().

```
def settings(self, rychlost, zrychleni, rotace,zrotace):
        self.maxRychlost=rychlost
        self.maxRychlostDeg=(self.maxRychlost*360)/self.delkaKolo
        self.maxZrychleni=zrychleni
        self.maxRotace=rotace
        self.maxZrotace=zrotace
```

8.3 funkce pid(Kp,Ki, Kd)

Funkce pid() mění koeficienty PID regulátoru. Proměnná Kp určuje koeficient proporcionální složky regulátoru. Proměnná Ki určuje koeficient integrační složky regulátoru. Poslední proměnná určuje koeficient derivační složky regulátoru. Všechny proměnné mění hodnoty PID regulátoru, který je součástí třídy rizeni. Změnou konstant PID regulátoru je možné zrychlit dosažení cílené hodnoty ve funkcích straight() a turn(), ale při špatném nastavení koeficientů ,je možné vytvořit nestabilní řízení.

```
def pid(self,Kp,Ki, Kd):
        self.Kp=Kp
        self.Ki=Ki
        self.Kd=Kd
```

8.4 funkce straight(delka)

Funkce straight() zajistí, aby robot urazil vzdálenost delka bez zatáčení. Proměnná delka určuje vzdálenost v milimetrech. Po uražení vzdálenosti se robot zastaví, ale po zastavení jsou kola brzděna pouze třecí silou. V první části se inicializují proměnné a určuje se úhel, o který se musí oba motory otočit. Následně pomocí PID regulátoru se přibližujeme k žádaným hodnotám úhlů obou motorů. V každém průchodu while cyklu vypočítáme rychlost, kterou bychom nastavili jako úhlovou rychlost motoru, pokud bychom neměli žádné omezení. Protože třída rizeni obsahuje omezení rychlosti a zrychlení, musíme kontrolovat, zda vypočítaná rychlost nepřevyšuje maximální rychlost nebo maximální zrychlení. Pokud je jedna z omezení porušena, je nastavena maximální hodnota, která je přípustná. Poté, co oba motory dosáhnout cíleného úhlu, robot se zastaví.

```
 def straight(self, delka):
   sumaL=0
   sumaP=0
   minHodnotaL=0
   minHodnotaP=0
   otocKolo=delka/self.delkaKolo
   uhel=(delka/self.delkaKolo)*360
   rychlostL=0
   rychlostR=0
   cas2=0
   while ((abs(self.motorLevy.angle()-uhel)>self.maxOdchylka) or
(abs(self.motorPravy.angle()-uhel)>self.maxOdchylka)):
       cas1=cas2
       cas2=self.cas.time()
       uhelL=self.motorLevy.angle()
       uhelP=self.motorPravy.angle()
       Pl=(uhel-uhelL)
       Pp=(uhel-uhelP)
       Dl=Pl-minHodnotaL
       Dp=Pp-minHodnotaP
       sumaL+=Pl
       sumaP+=Pp
       rychlostLmin=rychlostL
       rychlostRmin=rychlostR
       rychlostL=self.Kp*Pl+self.Kd*Dl+self.Ki*sumaL
       rychlostR=self.Kp*Pp+self.Kd*Dp+self.Ki*sumaP
       if (self.degtomm((rychlostL-rychlostLmin)/((cas2-cas1)/1000))>self.maxZrychleni):
            rychlostL=rychlostLmin+self.mmtodeg(self.maxZrychleni)
       if (self.degtomm((rychlostR-rychlostRmin)/((cas2-cas1)/1000))>self.maxZrychleni):
            rychlostR=rychlostRmin+self.mmtodeg(self.maxZrychleni)
       if abs(self.degtomm(rychlostL))>self.maxRychlost:
            if rychlostL<0:
                rychlostL=-self.mmtodeg(self.maxRychlost)
            else:
                rychlostL=self.mmtodeg(self.maxRychlost)
       if abs(self.degtomm(rychlostR))>self.maxRychlost:
            if rychlostR<0:
                rychlostR=-self.mmtodeg(self.maxRychlost)
            else:
                rychlostR=self.mmtodeg(self.maxRychlost)
       self.motorLevy.run(rychlostL)
       self.motorPravy.run(rychlostR)
   self.motorLevy.brake()
   self.motorPravy.brake()
```

8.5 funkce turn(uhel)

Funkce turn() zajistí, že se robot otočí na místě o úhel uhel. Po dosažení žádaného úhlu se robot zastaví, ale kola budou brzděna pouze třecí silou.

V první části kódu se inicializují všechny proměnné a vypočítá se úhel, o který se musí každý motor otočit. Ve while cyklu se změří čas, který programu trvá projít jednou while cyklem. Podle času a rozdílu úhlu robota se vypočítá rychlost rotace robota. Podle rozdílu rychlostí a časů se vypočítá i zrychlení rotace robota. Po výpočtu nové rychlosti motoru se zkontroluje, jestli zrychlení rotace není vyšší než

maximální povolené. Nakonec se zkontroluje, zda vypočítaná rychlost rotace robota není vyšší než maximální povolená. Poté, co motory dosáhnou úhlů, které jsou žádané, se oba motory zastaví.

```
def turn(self,uhel):
    sumaL=0
    sumaP=0
    minHodnotaL=0
    minHodnotaP=0
    delkaOtocky=math.pi * self.rozvor*uhel/360
    otocKolo=delkaOtocky/self.delkaKolo
    uhel=(delkaOtocky/self.delkaKolo)*360
    rychlostOtacky=0
    uhel2=0
    cas2=0
    rychlostL=0
    rychlostR=0
    while ((abs(self.motorLevy.angle()-uhel)>self.maxOdchylka) or
(abs(self.motorPravy.angle()+uhel)>self.maxOdchylka)):
        cas1=cas2
        cas2=self.cas.time()
        uhel1=uhel2
        uhel2=self.angle()
        uhelL=self.motorLevy.angle()
        uhelP=self.motorPravy.angle()
        Pl=(uhel-uhelL)
        Pp=(uhel+uhelP)
        Dl=Pl-minHodnotaL
        Dp=Pp-minHodnotaP
        sumaL+=Pl
        sumaP+=Pp
        rychlostOtackyMin=rychlostOtacky
        rychlostOtacky=(uhel2-uhel1)/((cas2-cas1)/1000)
        rychlostLmin=rychlostL
        rychlostRmin=rychlostR
        rychlostL=self.Kp*Pl+self.Kd*Dl+self.Ki*sumaL
        rychlostR=self.Kp*Pp+self.Kd*Dp+self.Ki*sumaP
        zrychleniOtacky=self.degtorot((rychlostL-rychlostLmin)/((cas2-cas1)/1000))
       if (abs(zrychleniOtacky)>self.maxZrotace):
            d=self.rottodeg(self.maxZrotace)
            if (zrychleniOtacky) >0:
                rychlostL=rychlostLmin+d
                rychlostR=rychlostRmin+d
            else:
                rychlostL=rychlostLmin-d
                rychlostR=rychlostRmin-d
       if (abs(rychlostL)>self.rottodeg(self.maxRotace)):
            if rychlostOtacky >0:
                rychlostL=(self.rottodeg(self.maxRotace))
                rychlostR=(self.rottodeg(self.maxRotace))
            else:
                rychlostL=-(self.rottodeg(self.maxRotace))
                rychlostR=-(self.rottodeg(self.maxRotace))
        self.motorLevy.run(rychlostL)
        self.motorPravy.run(-rychlostR)
        wait(10)
    self.motorLevy.brake()
    self.motorPravy.brake()
```

8.6 funkce drive(rychlost, otaceni)

Funkce drive() má 2 vstupní parametry. První proměnná určuje rychlost v milimetrech za sekundu dopředným směrem. Druhá proměnná určuje úhel ve stupních, o který se robot otočí za jednu sekundu.

Funkce nejdříve vypočítá rychlost obou motorů, aby se robot pohyboval rychlostí rychlost a otáčel se rychlostí otaceni. Protože vstupem funkce je rychlost translace a rotace, maximální hodnoty definované ve třídě rizeni nebudou kontrolovány.

```
def drive(self, rychlost, otaceni): #mm/s, deg/s
    rychlostRov=(rychlost*360)/(self.delkaKolo)
    rychlostOtoc=(self.rozvor*otaceni)/(2*self.polomer)
    self.motorLevy.run(rychlostRov+rychlostOtoc)
    self.motorPravy.run(rychlostRov-rychlostOtoc)
```

8.7 funkce distance()

Funkce distance() má za úkol vypočítat, jakou vzdálenost robot urazil od začátku kódu nebo od posledního zavolání funkce reset(). Uraženou vzdálenost vypočítá pomocí úhlu každého motoru. Úhel motoru následně převedeme na uraženou vzdálenost v milimetrech a danou hodnotu funkce vrátí.

```
def distance(self):
    #vraci urazenou vzdalenost
    stupneL=self.motorLevy.angle()
    stupneR=self.motorPravy.angle()
    delka=(self.delkaKolo*stupneL/360+self.delkaKolo*stupneR/360)/2
    return delka
```

8.8 funkce angle()

Funkce angle() vrací úhel, o který se robot otočil od začátku kódu nebo od posledního zavolání funkce reset(). V první řadě získáme rozdíl mezi úhly motorů. Následně rozdíl úhlů motorů převedeme a získáme úhel otočení robota.

```
def angle(self):
   #vraci uhel otoceni robota
   stupneL=self.motorLevy.angle()
   stupneR=self.motorPravy.angle()
   rozdilStupne=(stupneL-stupneR)/2
   delkaObvod=self.delkaKolo*rozdilStupne/360
   otoceni=(delkaObvod/self.delkaRozvor)*360
   return otoceni
```

8.9 funkce state()

Funkce state() vrací pole se 4 hodnotami. První hodnota pole určuje uraženou vzdálenost. Tuto hodnotu získáme zavoláním funkce distance(). Druhá hodnota určuje rychlost pohybu, kterou získáme jako rozdíl 2 vzdáleností dělený rozdílem časů, ve kterých ujeté vzdálenosti zjišťujeme. Třetí hodnota určuje úhel, o který se robot otočil. Tuto hodnotu získáme zavoláním funkce angle(). Poslední hodnota určuje rychlost rotace robota. Obdobným způsobem jako při získání druhé hodnoty v poli, získáme rozdíl dvou úhlů otočení robota, který vydělíme rozdílem časů, ve kterých úhly zjišťujeme. Z výše zmíněných hodnot vytvoříme pole, které funkce vrací.

```
def state(self):
    #vraci [vzdalenost, rychlost, uhel robota, rychlost otaceni robota]
    cas1=self.cas.time()
    vzdalenost1=self.distance()
    uhel1=self.angle()
    wait(50)
    cas2=self.cas.time()
    vzdalenost2=self.distance()
    uhel2=self.angle()
    rychlostT=(vzdalenost2-vzdalenost1)/((cas2-cas1)/1000)
    rychlostR=(uhel2-uhel1)/((cas2-cas1)/1000)
    return [vzdalenost2,rychlostT, uhel2, rychlostR]
```

8.10 funkce reset()

Funkce reset() má za úkol vynulovat uraženou vzdálenost a úhel otočení robota. Jelikož jsou tyto hodnoty vypočítávány z úhlů motoru, stačí vynulovat úhly motorů. Pro vynulování je zavolána funkce reset_angle() se vstupním parametrem 0.

```
def reset(self):
    #vynuluje vzdalenost a rotaci robota
    self.motorLevy.reset_angle(0)
    self.motorPravy.reset_angle(0)
```

8.11 funkce stop()

Funkce stop() má za úkol zastavit robota. Po zastavení lze s motory lehce otáčet, protože na ně působí jen třecí síla. Pro zastavení robota stačí zastavit oba motory, proto je pro tuto příležitost zavolána funkce stop() na oba motory robota.

```
def stop(self):
    self.motorLevy.stop()
    self.motorPravy.stop()
```

8.12 funkce mmtodeg(mm) a funkce degtomm(deg)

Tyto funkce převádí jediný vstupní parametr z rychlosti robota v milimetrech za sekundu na úhlovou rychlost motoru a naopak. Celý převod je postaven na předpokladu, že pohyb robota probíhá bez prokluzu. Z výpočtu obvodu kola získáme, že pokud se kola pohybují rychlostí 360 °/s, tak se robot pohybuje rychlostí o velikost obvodu kola robota. V rovnici (1) nalezneme vztah pro výpočet obvodu kola. Ve vztahu (2) je znázorněn převod z úhlové rychlosti motoru robota na dopřednou rychlost robota. Poměr úhlové rychlosti robota a čísla 360 určuje, kolikrát se kolo za sekundu otočí. Každé otočení kola o poloměru r znamená, že robot urazil vzdálenost stejně velkou, jako je obvod kola robota. Z tohoto důvodu je poměr vynásoben obvodem kola. Úpravou rovnice (2) vyjádříme úhlovou rychlost motoru, čímž získáme vztah (3).

$$o = 2 \cdot \pi \cdot r \tag{1}$$

$$v = \frac{o \cdot \omega}{360} = \frac{2 \cdot \pi \cdot r \cdot \omega}{360} \tag{2}$$

$$\omega = \frac{v \cdot 360}{o} = \frac{360 \cdot v}{2 \cdot \pi \cdot r} \tag{3}$$

```
def mmtodeg(self, rychlost):
    return  (rychlost*360)/self.delkaKolo

def degtomm(self, rychlost):
    return  (rychlost*self.delkaKolo)/360
```

8.13 funkce rottodeg(rot) a degtorot(deg)

Poslední pomocné funkce převádí rychlost rotace robota na úhlovou rychlost motorů a naopak. Při rotaci robota o 360 ° opíšou kola kružnici o průměru vzdálenosti kol. Z tohoto poznatku pomocí obvodu kružnice opsané kolami a poměru rychlosti rotace robota a čísla 360 dokážeme vypočítat, jakou vzdálenost v milimetrech musí kola urazit za jednu sekundu, aby se robot otočil o zadaný úhel. V rovnici (4) lze najít vyjádřenou dráhu kol uraženou za jednu sekundu pomocí proměnné d. Tato vzdálenost musí být stejně velká jako vzdálenost vyjádřena pomocí poloměru kol. V rovnici (5) nalezneme kromě obvodu kružnice i koeficient k, který určuje, kolikrát se kolo otočí o 360 °, aby danou dráhu kolo urazilo. Nakonec koeficient k vynásobíme číslem 360, abychom získali úhlovou rychlost motoru. V rovnici (7) nalezneme vyjádřenou úhlovou rychlost motoru pomocí rychlosti rotace celého robota. V rovnici (8) je pouze ze stejné rovnice vyjádřena rychlost rotace robota pomocí úhlové rychlosti motoru.

$$v = 2 \cdot \pi \cdot \frac{d}{2} \cdot \frac{\omega_r}{360} = \pi \cdot d \cdot \frac{\omega_r}{360} \tag{4}$$

$$\pi \cdot d \cdot \frac{\omega_r}{360} = 2 \cdot \pi \cdot r \cdot k \tag{5}$$

$$k = \frac{\omega_r \cdot d}{360 \cdot 2 \cdot r} \tag{6}$$

$$\omega_m = k \cdot 360 = \frac{\omega_r \cdot d}{2 \cdot r} \tag{7}$$

$$\omega_r = \frac{2 \cdot r \cdot \omega_r}{d} \tag{8}$$

```
def rottodeg(self,rychlost):
    return (self.rozvor*rychlost)/(2*self.polomer)

def degtorot(self,rychlost):
    return (2*self.polomer*rychlost)/(self.rozvor)
```

Praktická část

V této části si rozebereme 3 úlohy, které budou demonstrovat využití funkcí knihovny Pybrics. První úloha je klasické sledování černé čáry. Sledování černé čáry chceme dosáhnout za pomocí P, PD, PI nebo PID regulátoru. Cílem úlohy je demonstrovat využití programovacího jazyka Python na regulační úloze.

Druhá úloha navazuje na první, kde ke sledování černé čáry se připojí problém, jak objet překážku, která se bude nacházet na černé čáře. Tuto překážku musíme objet bez fyzického kontaktu a následně opět začít sledovat černou čáru.

V poslední úloze budeme regulovat plošinu, aby byla ve vodorovné poloze. Na plošina se bude nacházet vozík a úkolem robota je udržet vozík na plošině, mezitím co robot pojede po nerovné dráze. Kdyby nebyla plošina řízena servomotorem, pohybovala by se jako houpačka nebo by byla pevně k robotovi připevněna, což by zapříčinilo spadnutí vozíku z plošiny, poté co by robot najel na nakloněnou rovinu. Pomocí regulátoru řídící servomotor chceme zamezit spadnutí vozíku z plošiny.

1 Sledování černé čáry pomocí PID regulátoru

1.1 Zadání úlohy (11)

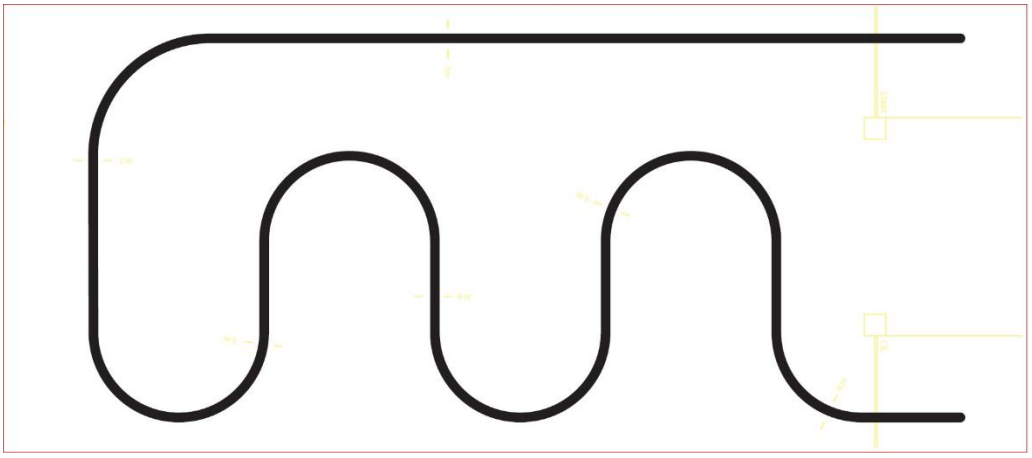
Cílem úlohy je sestavit a naprogramovat robot, který v co nejkratším čase projede dráhu vyznačenou černou čárou. Robot musí černou čáru následovat samostatně, není dovoleno robota ovládat pomocí hlasu, Bluetooth nebo jiné komunikace.

Robot smí být sestaven pouze ze stavebnice LEGO Education EV3 nebo LEGO Education EV3 Doplňková souprava. Při stavbě robota se nesmí používat jiné součástky a pomůcky jako například lepidla, šroubky, pásky či obdobné spojovací materiály. Robot by neměl mít větší půdorys než 40 x 40 cm. Navíc nesmí mít kluzný podvozek. Za podvozek se považuje každá část robota, která se dotýká podložky. Aby podvozek nebyl brán za kluzný, musí se každá část podvozku při pohybu odvalovat, nikoliv klouzat.

Cílem robota je projet vyznačenou dráhu. Pro každou soutěžní dráhu je stanoven maximální čas projetí. Pokud robot v tomto časovém úseku dráhu neprojede, považuje se, že robot ujel 0 m. Pokud robot nedorazí do cíle, výsledkem jízdy je vzdálenost, kterou korektně robot zvládl urazit. Během jízdy mohou členové týmu kdykoliv ukončit jízdu. V danou chvíli bude výsledek stejný, jako by se robot vzdálil od dráhy. Vyhrává tým, který projel celou dráhu za nejkratší čas. Pokud žádný tým neprojel celou dráhu v zadaném čase, vyhrává tým s největší korektně uraženou vzdáleností.

1.2 Soutěžní plán (11)

Celkový rozměr podložky je 250 x 150 cm. Podložku během plnění úkolu nesmí robot opustit. Robot bude jezdit po laminátové desce, na které bude položen bílí papír s černou čárou. Spojitá černá čára o tloušťce 1 až 5 cm a délce 1 až 20 m určuje dráhu, kterou by měl robot projet. Soutěžní čára může mít libovolný tvar, avšak minimální poloměr zatáček je 10 cm a čára musí být vzdálena od nejbližší hrany podložky minimální 25 cm. Na plánu nalezneme žlutou barvou označené dva obdélníky o rozměrech 40 x 40 cm, které označují startovní a cílové pole. Při sledování černé čáry se robot nesmí od čáry vzdálit o více jak 20 cm nebo si zkrátit cestu.



Obr 25: Podoba možného soutěžního plánu (12)

1.3 Sestavení robota

Pro sledování černé čáry je vhodné využít světelný senzor. Pro pohyb robota použijeme 2 servomotory, které budou pohánět stejně velká kola. V zadní části bude umístěno volné kolečko, aby robot byl stabilní.

Světelný senzor vysílá červené záření, které podle počtu vyslaných a přijatých paprsků vrací hodnotu v procentech. V ideálním případě by se od bílého podkladu měly odrážet všechny vyslané paprsky, a proto by senzor měl vracet hodnotu 100 %. Od černého pozadí by se v ideální případě neměl odrážet

žádný světelný paprsek, a proto by senzor měl vracet hodnotu 0 %. Senzor pracuje v pulsním režimu, což znamená, že ve velmi krátkých časových intervalech střída dva pracovní módy. V prvním módu senzor vysílá i přijímá světelné paprsky. V druhém módu žádné paprsky nevysílá, ale pouze přijímá paprsky z okolního prostředí. Porovnáváním těchto dvou režimů získáme pouze ty paprsky, které byly vyslány senzorem, a proto okolní osvětlení nemá znatelné rušivé účinky.

Pokud senzor nevrací ideální hodnoty, je možné, že podklad není dostatečně černý nebo bílý. Dalším možným důvodem, proč senzor nepracuje ideálně je, že senzor není správně upevněn na kostru robota. V ideálním případě by senzor měl směřovat kolmo na podklad, protože pokud by senzor svíral jiný než pravý úhel, dojde k tomu, že část paprsků se bude odrážet mimo snímanou plochu senzorem. Z toho vyplývá, že při špatném naklopení senzoru není možné, aby senzor vracel hodnotu 100 % za předpokladu, že se robot pohybu je po rovné podkladu. Dalším pravidlem, jak správně upevnit senzor, je že nesmí být příliš blízko ani příliš daleko od podložky. Experimentálně jsem dospěl k závěru, že k optimálnímu chování senzor dosahuje, pokud je umístěn 1,5 cm nad podložkou.

Pro jednodušší řízení by robot měl být souměrný, použitá hnací kola by měla být stejná. Pokud by robot nebyl osově souměrný, znamenalo by to, že váha na jednom kole je větší, a proto když oběma motorům nastavíme stejnou rychlost otáčení, jeden z motorů bude mít větší vliv na pohyb a robot nepojede rovně.

Z výše popsaných důvodů jsem využil základ robota popsaný v návodě (příloha 2). Tento robot je symetrický, proto bude jednodušší jet s robotem rovně. Jelikož se budeme snažit sledovat černou čáru, budeme potřebovat, aby robot reagoval co nejrychleji, proto nebudeme používat ozubená kolečka, která zvyšují maximální rychlost robota, ale je mnohem obtížnější robota ovládat. Horší ovládání plyne z faktu, že mezi ozubenými koly je volnost. Nakonec dopředu připevníme světelný senzor, díky kterému budeme snímat, kde se černá čára nachází.

1.4 Software pro sledování černé čáry pomocí třídy Drivebase

Nejdříve si musíme inicializovat třídy, díky kterým budeme pracovat s oběma motory i se světelným senzorem. Následně vytvoříme i třídu DriveBase, pomocí které budeme pracovat s robotem.

Než dále budeme psát kód, musíme zjistit, jaké hodnoty světelný senzor vrací na bílém a černém podkladu soutěžního plánu. Rozsah hodnot senzoru zjistíme tak, že v menu klikneme na „device“ --> „senzors“-->“color sensor“-->“watch values“ a následně robota postavíme tak, aby senzor snímal černou nebo bílou část podložky. Tyto hodnoty zaneseme do programu jako proměnné cerna a bila. Protože budeme provádět normalizaci, střední hodnota, které se robot bude chtít držet, je 50. Následně vytvoříme proměnné P, I a D, které označují koeficienty PID regulátoru. V neposlední řadě musíme určit rychlost v milimetrech za sekundu, jakou se má robot pohybovat. Nesmíme zapomenout na to, že čím vyšší bude rychlost robota, tím hůře se bude robot řídit. Je to způsobeno tím, že za stejný čas mezi získáním hodnot světelného senzoru robot urazí větší vzdálenost a tím je náchylnější pro přejetí černé čáry. Přejetí černé řáry zapříčiní, že robot už nebude schopen se vrátit a následovat hranu čáry, na kterou je programován.

Následující příkazy se budou vykonávat v nekonečném while cyklu. Nejdříve ze světelného senzoru získáme hodnotu a tu normalizujeme, aby hodnoty byly rozprostřeny v rozsahu 0 až 100. To nám zaručí, že pokud robota pustíme na jiné trati, kde senzor vrací jiné hodnoty bílého a černého podkladu, stačí pouze změnit proměnné bila a cerna, ale pro stejné chování, nebude nutné měnit koeficienty PID regulátoru. Nejdříve zjistíme odchylku od ideální hodnoty určující chybu proporcionální složky. Následně po odečtení dvou po sobě jdoucích proporcionálních chyb získáme derivační chybu. Integrační chybu vypočítáme tak, že přičteme k dosavadní integrační chybě současnou proporcionální

chybu. Nakonec každou chybu vynásobíme s příslušným koeficientem a výsledné hodnoty sečteme a tím dostaneme číslo, které nám určuje rotaci robota, kterou následně zadáváme jako druhý vstupní parametr funkce drive(). První parametr této funkce je konstantní a je určen na začátku programu.

Konstanty PID regulátoru získáme Ziegler-Nicholsovou metodou. Ta se odvíjí od kritické hodnoty proporcionálního regulátoru. Následně určíme čas jednoho průběhu while cyklem a periodu kmitání robota kolem černé čáry na rovném úseku.

1.4.1 Určení kritické hodnoty a periody kmitu robota

V první řadě určíme kritickou hodnotu proporcionální složky regulátoru. Kritická hodnota se projeví tak, že robot na rovném úseku černé čáry bude kmitat, ale při kmitání nesmí robot přestat sledovat černou čáru. Ke kritické hodnotě jsme se dostali tím způsobem, že jsme zvyšovali koeficient proporcionální složky a sledovali, jak robot čím dál víc kmitá kolem černé čáry. Nakonec se dostaneme k hodnotě, kde robot kmitá natolik, že ztratí černou čáru. Kritickou hodnotu určíme jako nejvyšší hodnotu proporcionálního koeficientu, kdy ještě regulátor dokázal sledovat černou čáru.

Po určení kritické hodnoty proporcionálního regulátoru pustíme robota ještě jednou na rovném úseku černé čáry. Následně musíme určit periodu kmitu robota. Aby změřená perioda byla přesnější, nesledujeme pouze jediný kmit robota, ale je vhodné změřit čas, který trvá robotovi udělat 20 kmitů. Změřený čas následně vydělíme počtem kmitů a získáme průměrnou periodu jednoho kmitu. Čím více kmitů naměří, tím delší bude časový úsek, a tím přesněji bude určená perioda kmitu.

Při měření jsme dospěli ke kritické hodnotě Kc=1,5. Následně jsme změřili čas, za který robot udělal určitý počet kmitů. Osm kmitů robotovi trvalo 3,15 sekundy, tedy perioda jednoho kmitu je 0,39375 sekundy. Nakonec do kódu přidáme proměnnou k, která bude fungovat jako čítač průběhů while cyklem. Před while cyklem inicializujeme třídu StopWatch a v dalším příkazu zavoláme funkci time(). V posledním kroku přidáme na začátek while cyklu inkrementaci proměnné k a podmínku, zda se proměnná k se rovná 10 000. V podmínce se opět zavolá funkce time(). Odebrané časy se od sebe odečtou a vydělí se 10 000. Výsledek je čas v milisekundách určující dobu jednoho průběhu while cyklem, proto nesmíme zapomenou hodnotu převést na sekundy.

Následně z těchto hodnot podle Zigler-Nicholsonovy metody určíme koeficienty pro proporcionální, proporcionálně integrační, proporcionálně derivační a proporcionálně integračně derivační regulátor. Výpočet jednotlivých koeficientů nalezneme v Tab 1.

Tab 1: Výpočet Ziegler-Nicholsových koeficientů pro jednotlivé regulátory (13)

Druh regulátoru	Proporcionální koeficient K_p	Integrační koeficient K_i	Derivační koeficient K_d
P	$0{,}5 \cdot K_c$	0	0
PI	$0{,}45 \cdot K_c$	$\frac{1{,}2 \cdot K_p \cdot dT}{P_c}$	0
PD	$0{,}8 \cdot K_c$	0	$\frac{K_p \cdot P_c}{8 \cdot dT}$
PID	$0{,}6 \cdot K_c$	$\frac{2 \cdot K_p \cdot dT}{P_c}$	$\frac{K_p \cdot P_c}{8 \cdot dT}$

1.4.2 P regulátor

V první řadě jsme určili koeficient proporcionálního regulátoru za pomoci kritické hodnoty. Tuto hodnotu jsme vynásobili číslem 0,5, tedy proporcionální koeficienty regulátoru byl 0,75.

$$K_p = 0{,}5 \cdot K_c = 0{,}5 \cdot 1{,}5 = 0{,}75 \quad (9)$$

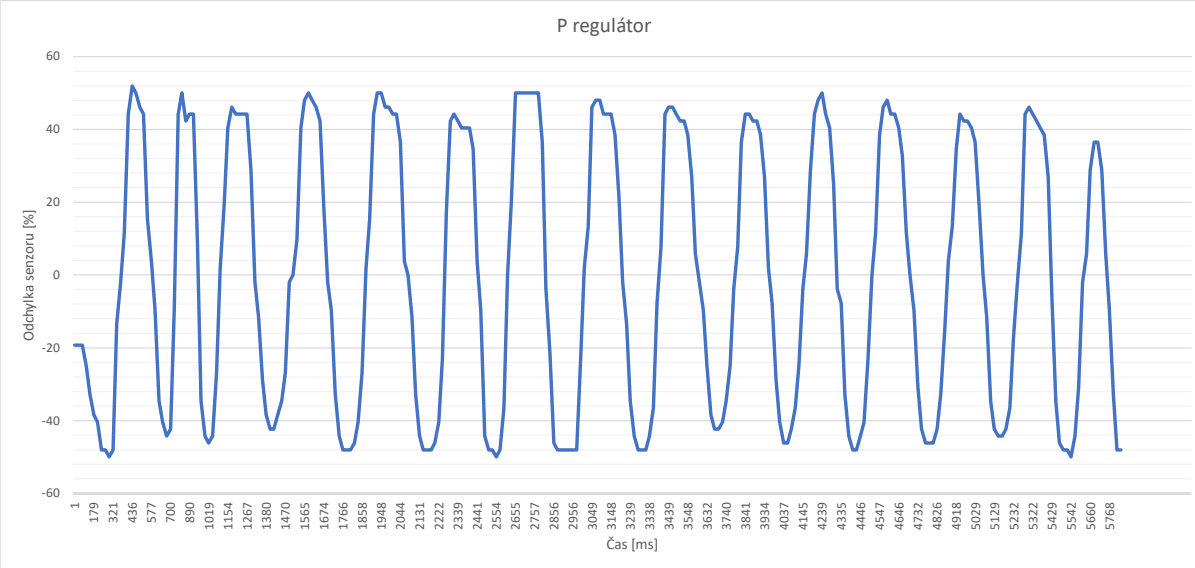
S tímto proporcionálním koeficientem jsme na zkušební dráze dosáhli maximální rychlosti 120 mm/s. Robot sledoval čáru precizně bez značných překmitů, avšak nízká hodnota proporcionálního koeficientu způsobovala nedostatečné dotáčení robota v zatáčkách, což mělo za příčinu ztráty sledované čáry. Z tohoto chování plyne nízká maximální rychlost robota.

Pokud bychom se neřídili Ziegler-Nicholsovou metodou a vypočítané hodnoty bychom brali jako doporučené, změnili bychom proporcionální koeficient dle vztahu (10). S touto hodnotou koeficientu jsme dosáhli maximální rychlosti 160 mm/s. Projetí dráhy vyšší rychlostí způsobilo překmity na rovných úsecích, avšak při rychlosti 160 mm/s ještě nedocházelo ke ztrátě černé čáry.

$$K_p = 0{,}7 \cdot K_c = 0{,}7 \cdot 1{,}5 = 1{,}05 \quad (10)$$

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 120
P =1.5*0.5
I=0
D=0
zatoc=0
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    zatoc = - P * chybaP
    rob.drive(rychlost, zatoc)
```



Obr 26: Hodnoty světelného senzoru P regulátoru na rovném úseku černé čáry

Pokud bychom chtěli zlepšit sledování černé čáry bez úpravy rychlosti a proporcionálního koeficientu regulátoru, mohli bychom vynásobit rychlost rotace robota konstantou, pokud robot bude zatáčet doprava. Je to způsobeno tím, že v levotočivých zatáčkách není tak zásadní nedotáčivost robota, protože po konečném čase robot nalezne černou čáru a bude pokračovat v jejím sledování. Při nedotáčivost robota v pravotočivých zatáčkách dochází k tomu, že robot zatočí nedostatečně a dostane se na levou stranu černé čáry a začne snímat bílé pozadí. Pokud se světelné paprsky odráží od bílého podkladu, robot bude zatáčet doleva a oddalovat se od černé čáry. Tato situace je popsána pro sledování pravé hrany černé čáry. Pokud robot sleduje levou hranu černé čáry, problémové zatáčky pro P regulátor jsou levotočivé.

Abychom docílili projetí robota dráhou bez snížení rychlosti a změny koeficientu, přidali jsme do kódu podmínku, zda robot bude zatáčet doprava. Pokud je podmínka splněna, rychlost rotace zdvojnásobíme, aby robot byl schopen projet ostré pravotočivé zatáčky.

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 160
P =1.5*0.5
I=0
D=0
zatoc=0
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    zatoc = -P * chybaP
    if zatoc>0:
        zatoc *=2
    rob.drive(rychlost, zatoc)
```

1.4.3 PI regulátor

Přidáním integrační složky docílíme, že při sledování rovné černé čáry v nekonečném čase bude mít regulátor nulovou odchylku. Integrační složka napomáhá projet táhlé zatáčky, protože při projíždění těchto zatáček se nasčítá proporcionální chyby. Jejich suma nakonec zajistí zvyšující se akční zásah, kterého by bez integrační složky bylo velmi obtížné dosáhnout.

Proporcionální složka je dána jako kritická hodnota vynásobena hodnotou 0,45. Integrační koeficient je dán jako součin čísla 1,2, proporcionální složky a periody while cyklu, který je vydělen dobou kmitu robota. Pokud dosadíme do vzorců, zjistíme, že proporcionální koeficient je roven 0,675 a integrační koeficient je roven 0,00192. Integrační koeficient je velmi malý, protože běh programu je velmi rychlý a pokud by byl koeficient větší, docházelo k windupu nebo velkým výkyvům na rovné úseku černé čáry.

$$K_p = 0{,}45 \cdot K_c = 0{,}45 \cdot 1{,}5 = 0{,}675 \tag{11}$$

$$K_i = \frac{1{,}2 \cdot K_p \cdot dt}{pc} = \frac{1{,}2 \cdot 0{,}675 \cdot 0{,}0009351}{0{,}39375} = 0{,}00192 \tag{12}$$

S těmito koeficienty jsme na testovací dráze dosáhli maximální rychlosti 200 mm/s. Abychom této rychlosti dosáhli, museli jsme implementovat funkci anti-windup a určili jsme maximální hodnotu

integrační složky. Při splnění podmínky funkce Anti-windup se sníží integrační chyba na tři desetiny své původního hodnoty. V podmínce zjišťujeme, zda dochází k překmitu zapříčiněnému integrační chybou.

Abychom předcházeli překmitům způsobené integrační složkou, implementovali jsme maximální chybu integrační složky. Pokud by integrační chyba byla větší než maximální povolená hodnota, změní se hodnota integrační chyby na předem definovanou konstantu maximální integrační chyby.

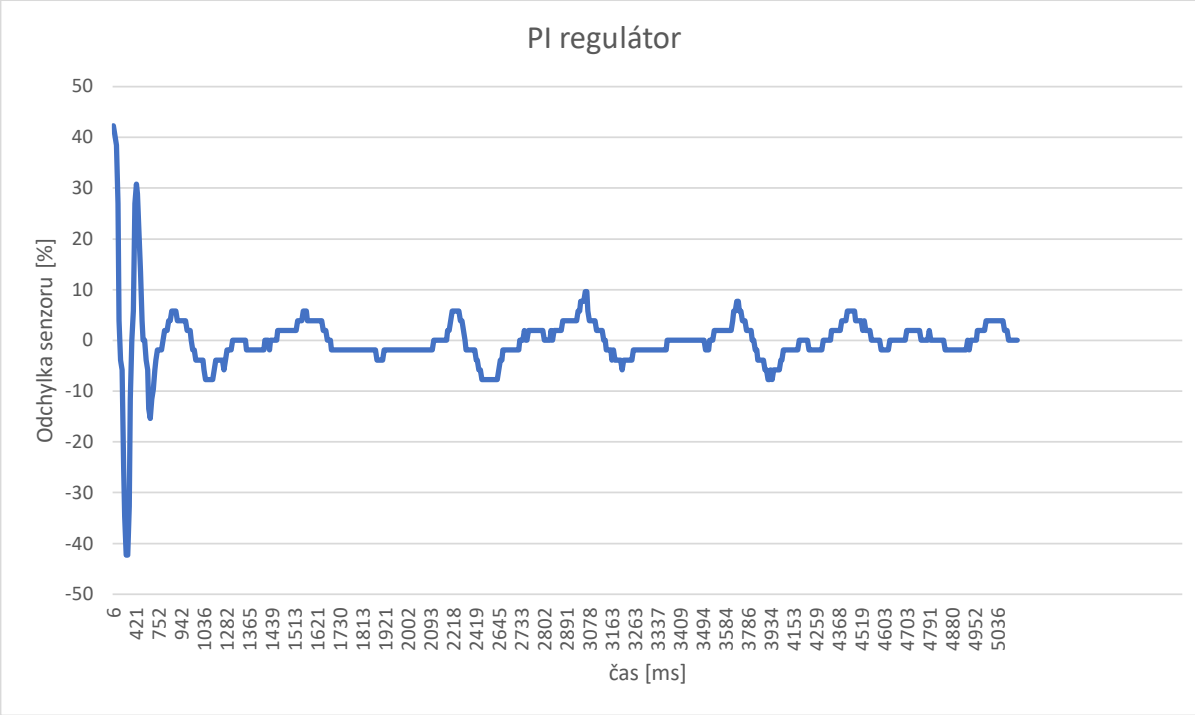
Po úpravě koeficientů jsme dosáhli maximální rychlosti 240 mm/s. Použité koeficienty nalezneme ve vztahu (13) a (14). Robot při projíždění dráhy sledoval čáru precizně a rychlost 240 mm/s byla nejvyšší dosažená rychlost na této dráze.

$$K_p = 0{,}5 \cdot K_c = 0{,}5 \cdot 1{,}5 = 0{,}75 \tag{13}$$

$$K_i = \frac{1{,}2 \cdot K_p \cdot dt}{pc} = 1{,}65 \cdot \frac{1{,}2 \cdot 0{,}75 \cdot 0{,}0009351}{0{,}39375} = 0{,}00353 \tag{14}$$

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 200
P =1.5*0.5
I=0,00353
D=0
chybaI=0
zatoc=0
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    chybaI=chybaI + chybaP
    if (chybaI>0 and chybaP<-47 and k==0) or (chybaI<0 and chybaP>47 and k==0):
        chybaI*=0.3
        k=1
    if abs(chybaP)<5:
        k=0
    if chybaI>12000:
        chybaI=12000
    elif chybaI <-12000:
        chybaI=-12000
    zatoc = -( P * chybaP + I * chybaI)
    rob.drive(rychlost, zatoc)
```



Obr 27: Hodnoty světelného senzoru PI regulátoru na rovném úseku černé čáry

1.4.4 PD regulátor

Přidáním derivační složky k proporcionálnímu regulátoru docílíme toho, že regulátor začne reagovat na změnu dvou po sobě jdoucích hodnot světelného senzoru. Derivační složka napomáhá zvýšit dotáčivost v zatáčkách. Největší akční zásah se tvoří především na začátku a na konci zatáčky, kde je rozdíl hodnot největší.

Proporcionální koeficient PD regulátoru je určen jako součin kritické hodnoty a čísla 0,8. Derivační koeficient regulátoru je dán jako součin proporcionálního koeficientu a periody kmitu robota, který vydělíme osminásobkem času průběhu while cyklem. V rovnicích (15) a (16) nalezneme číselné vyjádření koeficientů.

$$K_p = 0{,}8 \cdot K_c = 0{,}8 \cdot 1{,}5 = 1{,}2 \tag{15}$$

$$K_d = \frac{K_p \cdot pc}{8 \cdot dt} = \frac{1{,}2 \cdot 0{,}39375}{8 \cdot 0{,}0009351} = 63{,}1617 \tag{16}$$

Při přidání derivační složky regulátoru jsme dosáhli maximální rychlost 180 mm/s. Robot na dráze více kmital než při použití proporcionálního regulátoru. Je to způsobeno vyšší rychlostí a vyšší hodnotou proporcionální složky v PD regulátoru.

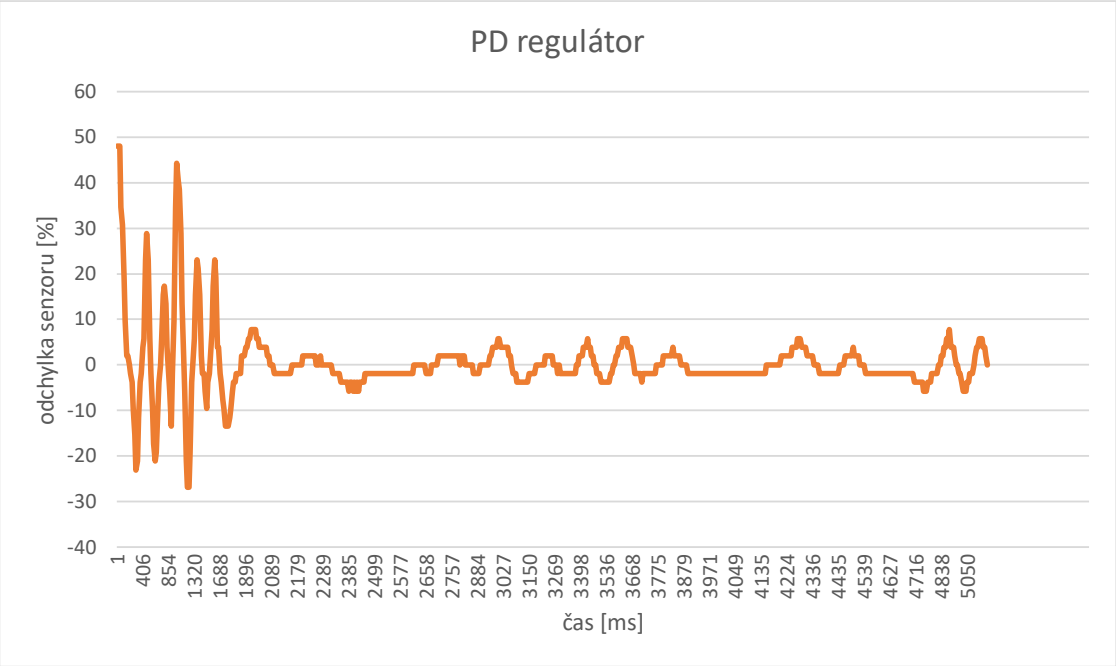
Následně jsme upravili koeficienty dle Tab 2, abychom dosáhli co nejvyšší rychlosti. Při testování jsme dosáhli maximální rychlosti 210 mm/s s koeficienty rozepsané ve vztahu (17) a (18).

$$K_p = 0{,}9 \cdot K_c = 0{,}9 \cdot 1{,}5 = 1{,}35 \tag{17}$$

$$K_d = 1{,}3 \cdot \frac{K_p \cdot pc}{8 \cdot dt} = 1{,}3 \cdot \frac{1{,}35 \cdot 0{,}39375}{8 \cdot 0{,}0009351} = 92{,}374 \tag{18}$$

```
motR=Motor(Port.B, Direction.CLOCKWISE)
motL=Motor(Port.C, Direction.CLOCKWISE)
senzorL = ColorSensor(Port.S2)
rob=DriveBase(motL, motR, 68.7, 169)

cerna = 3
bila = 58
stred = 50
rychlost = 180
P =1.5*0.8
I=0
D =63.1617
staryD=0
novyD=0
zatoc=0
while True:
    chybaP = (senzorL.reflection() *(100/(bila-cerna+1))-3)-stred
    staryD=novyD
    novyD=chybaP
    chybaD=novyD-staryD
    zatoc = - (P * chybaP + D*chybaD)
    rob.drive(rychlost, zatoc)
```



Obr 28: Hodnoty světelného senzoru PD regulátoru na rovném úseku černé čáry

1.4.5 PID regulátor

Posledním regulátorem je PID regulátor. Proporcionální koeficient regulátoru je určen jako součin kritické hodnoty a čísla 0,6. Integrační složka je určena jako zlomek, kde v čitateli nalezneme součin čísla 2, proporcionálního koeficientu a času průběhu while cyklem. Ve jmenovateli nalezneme periodu kmitu robota. Diferenciální složka je určena jako zlomek, kde ve jmenovateli nalezneme osminásobek času průběhu while cyklem a v čitateli nalezneme součin proporcionálního koeficientu a periody kmitu robota. Níže nalezneme číselné vyjádření koeficientů.

$$K_p = 0{,}6 \cdot K_c = 0{,}6 \cdot 1{,}5 = 0{,}9 \tag{19}$$

$$K_i = \frac{2 \cdot K_p \cdot dt}{pc} = \frac{2 \cdot 0{,}9 \cdot 0{,}0009351}{0{,}39375} = 0{,}00427 \tag{20}$$

$$K_d = \frac{K_p \cdot pc}{8 \cdot dt} = \frac{0{,}9 \cdot 0{,}39375}{8 \cdot 0{,}0009351} = 47{,}371 \tag{21}$$

S těmito koeficienty bylo nutné snížit rychlost natolik, že rychlost robota byla nižší než s implementovaným PD regulátorem. Bylo to způsobeno tím, že integrační chyba způsobovala velké překmity, které zapříčinily ztrátu sledované černé čáry.

V dlouhých zatáčkách, které robot nestíhá dokonale sledovat, dochází k windupu. Windup se projeví na konci dlouhé zatáčky, při které nebyla dokonale sledována černá čára. Během zatáčky se integrační složka zvětšuje až do bodu, kdy robot opět nalezne černou čáru. Pokud je integrační složka vysoká, dochází k velkému překmitu, který vede ke ztrátě černé čáry. Tomuto chování jde předejít implementací anti-windupu.

Pro implementaci anti-windupu vytvoříme podmínku, zda je integrační chyba kladná a aktuální proporcionální chyba je menší než daná konstanta. Pokud je podmínka splněna, snížíme nebo vynulujeme integrační chybu, abychom zamezili překmitu na konci zatáčky způsobeného integrační složkou. V tuto chvíli jsme zamezili překmitu u levotočivých zatáček, pokud sledujeme pravou hranu černé čáry. Abychom zamezili překmit v pravotočivých zatáčkách, musíme přidat podmínku, zda integrační chyba je záporná a zároveň proporcionální chyba je větší než zadaná konstanta. Při splnění podmínky snížíme nebo vynulujeme hodnotu integrační chyby. Následně můžeme tyto dvě podmínky spojit do jedné, aby byl kód přehlednější.

Po implementaci anti-windupu jsme dosáhli maximální rychlosti 180 mm/s. Tato rychlosti je shodná s maximální rychlostí PD regulátoru, tedy jsme nedosáhli žádného zlepšení. Proto jsme umístili senzor o jednu pozici výš a znovu vypočítali kritickou hodnotu, čas while cyklu a periodu kmitu robota. Dostali jsme se ke koeficientům, které jsou vyjádřeny ve vztazích (22), (23) a (24). Když jsme světelný senzor umístili senzor do výšky 1,5 cm nad podložku, hodnoty ze senzoru neměly takový rozsah, ale to nezměnilo chování regulátoru, protože získané hodnoty se normalizují. Navíc hrana čáry není tolik strmá z pohledu senzoru, což zapříčiní lepší chování integrační složky, ale sníží akční zásah derivační složky. Pomocí toho PID regulátoru s implementovaným anti-windupem jsme dosáhli rychlosti 210 mm/s.

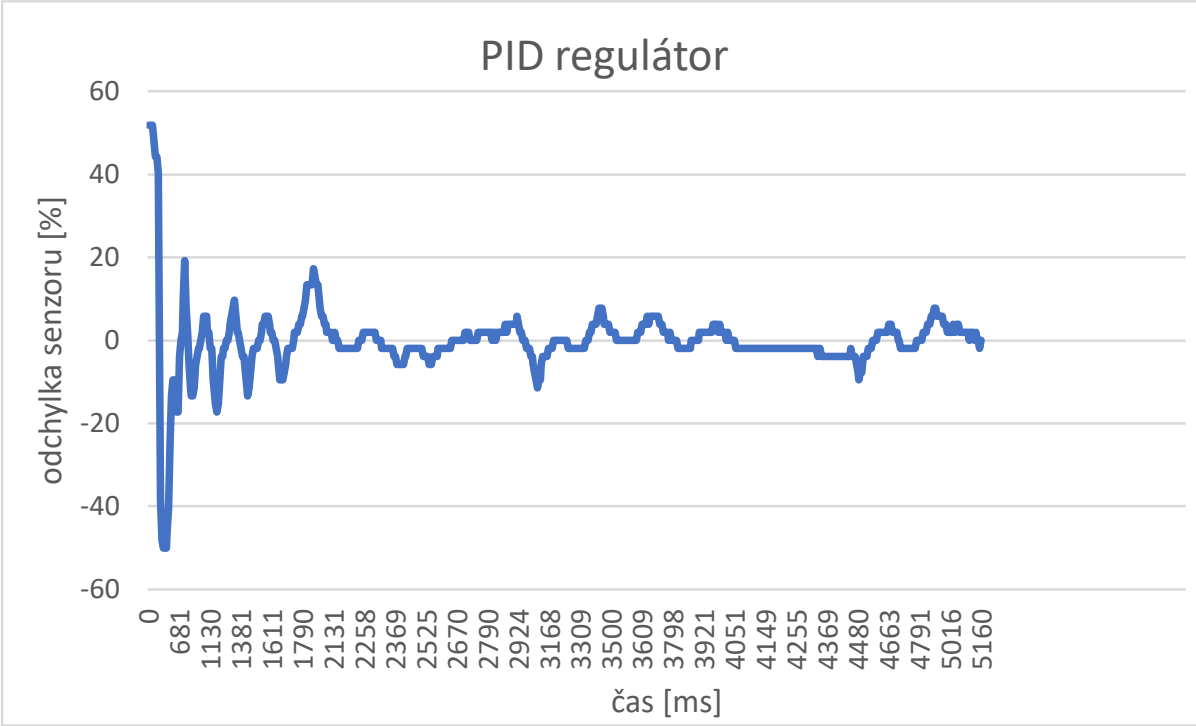
$$K_p = 0{,}6 \cdot K_c = 0{,}6 \cdot 1{,}6 = 0{,}96 \tag{22}$$

$$K_i = \frac{2 \cdot K_p \cdot dt}{pc} = \frac{2 \cdot 0{,}96 \cdot 0{,}0009351}{0{,}3308} = 0{,}00543 \tag{23}$$

$$K_d = \frac{K_p \cdot pc}{8 \cdot dt} = \frac{0{,}96 \cdot 0{,}3308}{8 \cdot 0{,}0009351} = 42{,}451 \tag{24}$$

```
cerna = 3
bila = 58
stred = 50
rychlost = 250
P=0.96
I=0.00543
D=42.451

chybaI=0
staryD=0
novyD=0
zatoc=0
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    chybaI=chybaI + chybaP
    if (chybaI>0 and chybaP<-47 and k==0) or (chybaI<0 and chybaP>47 and k==0):
        chybaI*=0.3
        k=1
    if abs(chybaP)<5:
        k=0
    staryD=novyD
    novyD=chybaP
    chybaD=novyD-staryD
    zatoc = strana*(P * chybaP + D*chybaD + I * chybaI)
        rob.drive(rychlost, zatoc)
```



Obr 29: Hodnoty světelného senzoru PID regulátoru na rovném úseku černé čáry

Maximální rychlost, které jsme dosáhli, je 240 mm/s. Této rychlosti jsme dosáhli pomocí PI regulátoru s implementovanou funkcí anti-windup. Koeficienty jednotlivých složek vznikly úpravou hodnot

koeficientů určených pomocí Ziegler-Nicholsovou metodou. Tyto hodnoty jsou rozepsány ve vztahu (32).

$$K_p = 0{,}75, \qquad K_i = 0{,}00353 \tag{25}$$

1.4.6 PID regulátor se změnou strany

Poslední implementace je podřízena tomu, aby robot mohl jet co nejvyšší rychlostí a nedocházelo k situacím, že robot černou čáru ztratí. Robot černou čáru ztratí v situaci, kdy se dostane na druhou stranu černé čáry, než na kterou je naprogramován. K přejetí černé čáry dochází v ostrých zatáčkách, kde robot nedostatečně zatočí a ocitne se na druhé straně, kde při správné implementaci regulátoru se robot bude od čáry oddalovat. Aby k těmto situacím nedocházelo, nejjednodušší možností je snížit rychlost robota.

Složitější možnost, jak tomu zamezit, je implementovat dvoustavový automat, kde v prvním stavu robot sleduje pravou hranu černé čáry. V druhém stavu bude robot sledovat naopak levou hranu černé čáry. Na začátku implementace se musíme rozhodnout, ve kterém stavu bude robot začínat a musíme ho i položit na patřičnou stanu černé čáry. Aby došlo ke změně stavu v automatu, musí robot přejet přes černou čáru. To je v implementaci kódu zaručeno tak, že světelný senzor vrací nejnižší možnou hodnotu, kterou máme definovanou jako proměnnou cerna. Následně se změní stav, ale nastává problém, protože při dalším průběhu while cyklem s největší pravděpodobností světelný senzor stále bude vracet nejnižší možnou hodnotu senzoru, a proto by měl svůj stav změnit na původní.

Z tohoto důvodu změna stavu je podmíněna proměnnou flag, která musí být rovna nule. Proměnná flag je změněna na hodnotu 0 v případě, že světelný senzor vrací hodnotu, která označuje bílý podklad. Z toho vyplývá, že pro změnu sledované strany musí nejdříve robot opustit černou čáru a až následovně může opět změnit sledovanou stranu černé čáry. Aby byla implementace správná, musíme při změně stavu změnit proměnnou flag na nenulovou hodnotu.

Při změně sledované strany černé čáry také musíme vynulovat integrační chybu. Pokud bychom integrační chybu nevynulovali, mohlo by nastat, že hodnota, která je uložená v proměnné chybaI by měla velký vliv na regulátor a zbytečně by oddálila robota od černé čáry.

Implementace prvního stavu automatu by bylo možné provést jako vnitřek podmínky if a druhý stav by byl vnitřek podmínky else. Toto provedení by bylo funkční, ale většina kódu by se v obou blocích opakovala. Z tohoto důvodu je nutné si uvědomit, že jediná změna nastává uvnitř výpočtu úhlové rychlosti, kterou se robot otáčí. Dokonce se nezmění ani velikost toho úhlu, ale pouze se změní jeho orientace, protože při sledování černé čáry na jedné straně bude úhel kladný, ale na druhé straně bude úhel záporný. Proto při změně stavu automatu se mění proměnná strana, která je vynásobena číslem -1. Pokud proměnnou strana vynásobíme výpočet úhlové rychlosti, získáme správnou hodnotu úhlové rychlosti a nemusíme více zasahovat do původního kódu.

```
cerna = 3
bila = 58
stred = 50
rychlost = 250
kp=2.2
dt=0.00176
pc=8.99/20
P=0.3*kp
I=1.2*2*P*dt/pc
D=P*pc/(3*dt)

chybaI=0
staryD=0
novyD=0
zatoc=0
flag=0
strana=1# 1=L, -1=P
while True:
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    if (chybaP <-49) and flag==0:
        flag=1
        strana=strana*(-1)
        chybaI=0
    if chybaP>49:
        flag=0
    chybaI=chybaI + chybaP
    staryD=novyD
    novyD=chybaP
    chybaD=novyD-staryD
    zatoc = strana* (P * chybaP + D*chybaD + I * chybaI)
    rob.drive(rychlost, zatoc)
```

2 Vyhnutí se překážky během sledování černé čáry

2.1 Zadání úlohy (14)

Druhá úloha vyplývá ze sledování černé čáry (kapitola 1.1), a proto i zadání jsou stejná až na dvě výjimky. První změna je, že čas na projetí dráhy byl zvýšen. Hlavní rozdíl mezi úlohami je, že se na dráze bude vyskytovat překážka, kterou robot musí samostatně objet bez fyzického kontaktu mezi překážkou a robotem a následně pokračovat ve sledování černé čáry. Všechna ostatní pravidla jsou stejná s úlohou sledování černé čáry.

Pro zjištění přítomnosti předmětu před robotem byl k robotovi připevněn ultrazvukový senzor. Pomocí funkce distance() získáme hodnotu z ultrazvukového senzoru, která nám udává vzdálenost senzoru od objektu v milimetrech. Maximální hodnota, kterou senzor vrací je 2550 milimetrů, takže robot dokáže zjistit přítomnost předmětu, pokud je předmět blíže než 2,55 metru před ultrazvukovým senzorem.

2.2 Objetí po „slepé“ trase

Nejjednodušší způsob, jak objet překážku je vytvořit funkci vyhniNaslepo(), která se provede po zaznamenání překážky před robotem. Funkce provede předem definované kroky. Po provedení těchto příkazů bude chtít robot opět najít černou čáru a začít ji sledovat. Protože příkazy ve funkci jsou předem dané, není možné zajistit, že nenastane fyzický kontakt mezi robotem a překážkou.

První provedení je takové, že při zaznamenání překážky v předem dané vzdálenosti se robot otočí o 45 ° od černé čáry a urazí určenou vzdálenost rovně. V tuto chvíli by měl být robot na úrovni překážky. Následně by se robot měl otočit o 90 ° k černé čáře a jet rovně, dokud nenarazí na černou čáru.

Tato implementace je problematická, protože vzdálenost, ve které je překážka zaznamenána, určuje, jaká je maximální velikost překážky, kterou lze bez fyzické interakce objet. Protože ultrazvukový senzor se dále nevyužívá, není možné určit, jestli robot při otáčení nedošlo k prokluzu kol a robot se neotočil o menší úhel, než který mu byl zadán. Pokud by to nastalo, robot by narazil do překážky a tím nesplnil požadovaný úkol. Poslední nevýhoda spočívá v tom, že předpokládáme, že překážka je umístěna na rovném úseku černé čáry. Pokud robot bude před překážkou sledovat pravou stranu černé čáry a překážka by byla umístěna v levotočivé zatáčce, pak by robot nebyl schopen najít a opět sledovat černou čáru.

```
def vyhniNaslepo():
    vzdal=senzorU.distance()
    r=70
    turn(45)
    straight((vzdal**2+r**2)**0.5)
    turn(-90)
    rob.drive(150,0)
    light=100
    while(light>40):
        light=((senzorL.reflection()-cerna) *(100/(bila-cerna)))
    rob.stop()
```

2.3 Objetí po kružnici

Abychom při objíždění překážky vždy našli černou čáru, musíme implementovat takový kód, který zaručí, že robot bude kroužit kolem překážky. Nejvhodnější trajektorie pro kroužení kolem překážky je kružnice. Aby se robot pohyboval konstantní rychlostí po kružnici o poloměru r, budeme používat funkci drive() z třídy DriveBase. Když robot objede celou kružnici, urazí vzdálenost s, která lze vyjádřit

jako obvod kružnice. Toto vyjádření nalezneme v rovnici (26). Protože známe délku trajektorie a rychlost, kterou se robot pohybuje, můžeme spočítat, za jaký čas robot celou trasu urazí. Věc, kterou potřebujeme zjistit, abychom mohli zavolat funkci drive() je rychlost rotace celého robota.

$$s = 2 \cdot \pi \cdot r \tag{26}$$

Úhlová rychlost se vyjádří jako úhel, který se urazí za daný čas. Tento čas je vyjádřen v rovnici (27). Pokud do rovnice (28) dosadíme vyjádření času z rovnice (27), získáme vyjádření úhlové rychlosti robota, protože úhel, o který se robot otočí, je 360 °.

$$t = \frac{s}{v} = \frac{2 \cdot \pi \cdot r}{v} \tag{27}$$

$$\omega = \frac{\phi}{t} = \frac{\phi}{\frac{2 \cdot \pi \cdot r}{v}} = \frac{\phi \cdot v}{2 \cdot \pi \cdot r} \tag{28}$$

Nyní máme vyjádřenou i úhlovou rychlost robota, která je společně s translační rychlostí vstupním parametrem funkce drive().

Opět při sledování černé čáry budeme kontrolovat, jestli ultrazvukový senzor nenabývá hodnot, které by označovaly přítomnost překážky před robotem. Pokud je překážka blíže než hraniční hodnota, robot se otočí o 90 °od černé čáry. Následně zavoláme funkci drive(). První vstupní parametr určuje translační rychlost robota, která je konstantní a je určena na začátku programu. Druhým vstupním parametrem je rychlost rotace celého robota, která je vyjádřena v rovnici (28). Ze vztahu (28) je zřejmé, že úhlová rychlost závisí na poloměru kružnice. Poloměr kružnice můžeme určit jako konstantu nebo jako vstupní parametr funkce pro vyhnutí se překážky. Podle poloměru je nutné i měnit hraniční vzdálenost, ve které se bude robot otáčet o 90 ° před překážkou. Čím větší bude poloměr kružnice, tím větší bude vzdálenost, ve které robot musí zareagovat na překážku, aby se robot vyskytoval na trajektorii vypočítané kružnice.

Preferovaná možnost je, že poloměr r bude vstupní parametr funkce vyhniKruznice(). Oproti funkci vyhniNaslepo() je možné, aby robot se vyhnul větším předmětům jen díky změně jediného vstupního parametru. Z toho plyne, že funkce vyhniKruznice() je mnohem univerzálnější než funkce vyhniNaslepo(), protože při změně překážky stačí změnit pouze jedinou konstantu. Protože se robot pohybuje po kružnici kolem překážky, v ideálním případě udržuje od překážky konstantní vzdálenost. I když se jedná o „slepou“ implementaci, chyba způsobená prokluzem kol nebude mít až takový vliv jako v předchozím případě. Posledním vylepšením od funkce vyhniNaslepo() je, že robot nalezne čáru vždy, protože dokáže objet celou překážku kolem dokola. Z toho plyne, že si tato implementace dokáže poradit nejen s překážkou na rovném úseku černé čáry, ale i pokud je překážka umístěna v libovolné zatáčce.

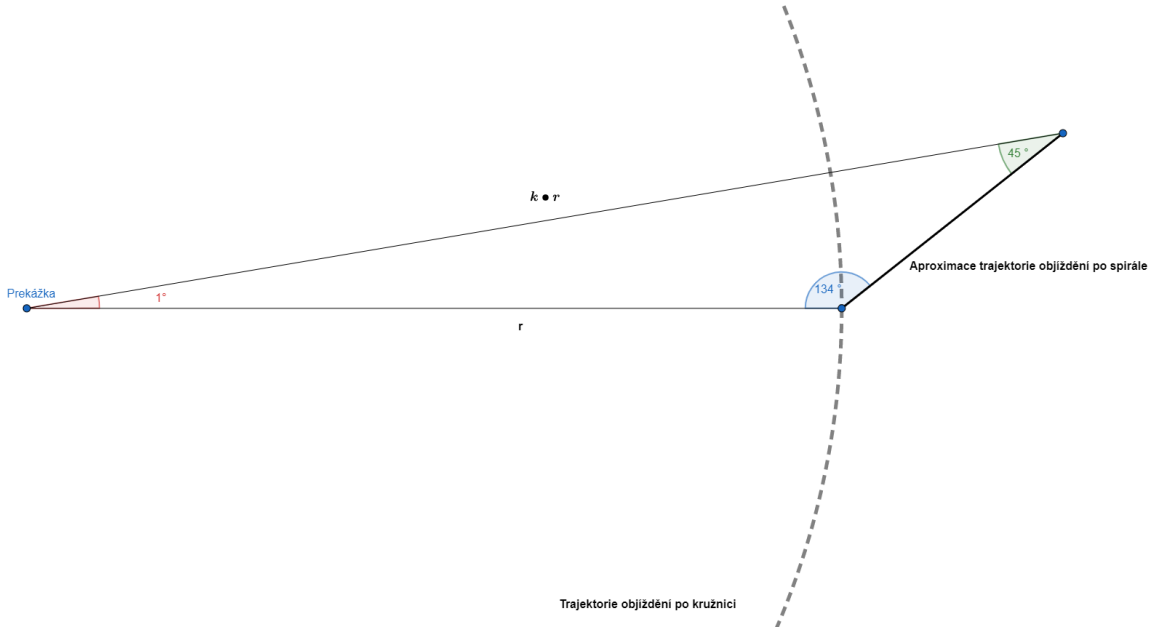
Nevýhodou této implementace je, že ve většině případů robot narazí na černou čáru kolmo. Většinu případů myslíme všechny případy, kde zatáčku si představíme jako 2 polopřímky, které tvoří úhel a ve vrcholu úhlu je umístěna překážka. Z toho plyne, že robot bude muset velmi zpomalit, aby následně černou čáru neztratil. Poté, co robot narazí na černou čáru, zastaví a bude se pomalu otáčet od černé čáry, dokud nenalezne její hranici. V tuto chvíli funkce končí a robot opět sleduje černou čáru.

```
def vyhnise2():
    uhel1=rob.angle()
    r2=160
    rychlost=100
    turn(90)
    rychlost=200
    zatoc=-(360*rychlost)/(2*5.65*math.pi*r2)
    rob.drive(rychlost,zatoc)
    light=100
    while(light>40):
        light=((senzorL.reflection()-cerna) *(100/(bila-cerna)))
    rob.stop()
```

2.4 Objetí po spirále

Abychom omezili vliv poslední nevýhody, můžeme implementovat takovou funkci, aby robot objel překážku po spirále. Spirála nám pomůže, aby robot nenarazil na černou čáru kolmo, ale pod předem určeným úhlem. Určitý úhel zvolíme jako 45 °, protože nejrychleji nalezneme černou čáru, když se budeme pohybovat po kružnici, ale narazíme na černou čáru pod úhlem 90 °. V ideálním případě bychom chtěli na černou čáru narazit pod úhlem 0 °, ale to by znamenalo, že se pohybujeme po přímce, která je tečna černé čáry. Tato možnost je velmi nepravděpodobná, a proto budeme považovat, že bychom černou čáru našli v čase blížící se k nekonečnu. Proto musíme udělat kompromis mezi časem nalezení a úhlem, pod kterým pravděpodobně nalezneme černou čáru. Kompromis zkonstruujeme pomocí průměrné hodnoty těchto 2 extrémních případů a dojdeme k výsledku, že bychom chtěli černou čáru nalézt pod úhlem 45 °.

Abychom došli k nějakému vyjádření, musíme si určit s jakou přesností budeme chtít čáru nalézt pod úhlem 45 °. Pro výpočet níže bylo vybráno, že pro změnu středového úhlu o 1 ° bude platit, že rameno úhlu bude protnuto pod úhlem 45 °. Tento případ je zobrazen na Obr 30. Z faktu, že součet všech vnitřních úhlů trojúhelníku je roven 180 ° znamená, že velikost posledního vnitřního úhlu je 134 °. Následně pomocí sinusové věty, která je zobrazena ve vztahu (29), určíme poměr mezi 2 délkami stran, který určuje změnu poloměru při změně středového úhlu o 1 °. Pomocí sinusové věty jsme vyjádřili, že tento poměr je roven 1,0173. Z toho plyne, že při změně středového úhlu o 360 °se poloměr zvětší 480,459 krát.



Obr 30: Náčrt výpočtu změny poloměru při změně úhlu o 1 °

$$\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} \tag{29}$$

$$\frac{\sin 45\,^\circ}{r} = \frac{\sin 134\,^\circ}{k \cdot r} \tag{30}$$

$$k = \frac{\sin 134\,^\circ}{\sin 45\,^\circ} = 1{,}0173 \tag{31}$$

V dalším kroku chceme spočítat délku spirály, po které se bude robot pohybovat. V první řadě musíme parametrizovat křivku, aby výpočet křivkového integrálu byl přehlednější. Parametrizace křivky vychází z parametrizace kružnice, kterou můžeme vidět v rovnicích (32) a (33), avšak v tuto chvíli není poloměr konstantní, ale bude se zvětšovat podle proměnné φ.

$$x = r \cdot \cos\varphi \tag{32}$$

$$y = r \cdot \sin\varphi \tag{33}$$

V rovnicích (34) a (35) si můžeme všimnout, že hlavní rozdíl mezi parametrizací kružnice a naší křivky je takový, že naše křivka nemá konstantní poloměr, ale poloměr je závislý na úhlu φ. Koeficient k určuje poměr z rovnice (31). Zlomek $\frac{180}{\pi}$ určuje převodní konstantu úhlu z radiánů na stupně.

$$x = \left(r \cdot k^{\varphi} \cdot \frac{180}{\pi}\right) \cdot \cos\varphi \tag{34}$$

$$y = \left(r \cdot k^{\varphi} \cdot \frac{180}{\pi}\right) \cdot \sin\varphi \tag{35}$$

Parametrizaci křivky v rovnicích (34) a (35) dosadíme do vzorce pro křivkový integrál. Abychom z křivkového integrálu získali délku křivky, funkce f se musí rovnat 1. Z toho vyplývá, že nás zajímá pouze velikost derivace parametrizace křivky. Tuto velikost následně dosadíme do integrálu a integrál vypočítáme.

$$l = \int_a^b f\big(\varphi(t)\big) \cdot \|\dot{\varphi}(t)\| \tag{36}$$

$$l = \int_a^b 1 \cdot \|\dot{\varphi}(t)\| \tag{37}$$

$$l = \int_0^{2\pi} \sqrt{\left(r \cdot 1{,}0173^{\varphi} \cdot \frac{180}{\pi} \cdot sin\,\varphi\right)^2 + \left(r \cdot 1{,}0173^{\varphi} \cdot \frac{180}{\pi} \cdot cos\,\varphi\right)^2}\; d\varphi \tag{38}$$

$$l = \int_0^{2\pi} \sqrt{\left(r \cdot 1{,}0173^{\varphi} \cdot \frac{180}{\pi}\right)^2 \cdot (sin^2\,\varphi + cos^2\,\varphi)}\; d\varphi \tag{39}$$

$$l = \int_0^{2\pi} \sqrt{\left(r \cdot 1{,}0173^{\varphi} \cdot \frac{180}{\pi}\right)^2 \cdot 1}\; d\varphi = \int_0^{2\pi} r \cdot 1{,}0173^{\varphi} \cdot \frac{180}{\pi}\; d\varphi \tag{40}$$

$$l = r \cdot \frac{180}{\pi} \cdot \int_0^{2\pi} 1{,}0173^{\varphi}\, d\varphi = r \cdot \frac{180}{\pi} \cdot \left[\frac{1{,}0173^{2\pi} - 1{,}0173^{0}}{\ln 1{,}0173}\right] \tag{41}$$

$$l = r \cdot \frac{180}{\pi} \cdot \cong 380{,}115 \cdot r \tag{42}$$

Nyní jako v minulém případě známe délku, kterou robot musí urazit. Robot se pohybuje konstantní rychlostí a úhel, o který se celý robot otočí, je 315 stupňů, protože robot začíná kolmo k černé čáře, ale po objetí celého kola svírá s černou čárou úhel 45, z toho plyne, že se robot neotočil o celých 360 °, ale pouze o 315 °. Stejným způsobem jako po vyhýbání se po kružnici, dopočítáme úhlovou rychlost robota, abychom mohli použít funkci drive(). V rovnici (43) nalezneme vyjádření úhlové rychlosti robota.

$$\omega = \frac{\varphi}{t} = \frac{\varphi}{\frac{s}{v}} = \frac{\varphi}{\frac{380{,}115 \cdot r}{v}} = \frac{\varphi \cdot v}{380{,}115 \cdot r} \tag{43}$$

Nyní máme vše vyjádřeno k zavolání funkce drive(). Proto si shrneme implementaci, jak se bude robot chovat. Při sledování černé čáry robot kontroluje pomocí ultrazvukového senzoru, zda se před robotem nevyskytuje překážka. Pokud před robotem je zaznamenána překážka ve vzdálenosti počátečního poloměru spirály sečtené s polovinou vzdálenosti kol a ultrazvukového senzoru, robot se zastaví a otočí se o 90 °od černé čáry. Následně je zavolána funkce drive(). První vstupní parametr je konstantní rychlost, která je určena na začátku kódu. Druhý vstupní parametr určuje úhlovou rychlost robota, která je vyjádřena v rovnici (43). Proměnná r určuje počáteční poloměr spirály a je určena jako konstanta na začátku kódu. Následně robot bude kroužit po spirále, dokud nenarazí na černou čáru. Po nalezení černé čáry je funkce ukončena a robot opět sleduje černou čáru.

Výhodou této implementace je fakt, že se stále robot vzdaluje od překážky, proto je méně pravděpodobně, že se s překážkou srazí. Navíc při nalezení černé čáry bude nejspíše robot s čárou svírat menší úhel než při objíždění překážky po kružnici. Největší nevýhoda této implementace je, že robot zvyšuje svůj poloměr příliš rychle, a proto není vhodné tuto implementaci používat, protože pro vyhnutí překážky je potřeba příliš mnoho prostoru.

```
def vyhnise():
    rob.turn(90)
    rychlost=200
    zatoc=-(360*rychlost)/(2*math.pi*300)
    rob.drive(rychlost,zatoc)
    light=100
    while(light>50):
        light=((senzorL.reflection()-cerna) *(100/(bila-cerna)))
    rob.stop()
    P=0.3
    while(abs(light)<3):
        light=(senzorL.reflection()-cerna)*(100/(bila-cerna))-50
        rob.drive(0, -P*light)
    rob.stop()
```

2.5 Objetí s otočným ultrazvukovým senzorem

Další implementaci je možné použít po přestavbě robota. Hlavní změnou konstrukce robota je využití středního servomotoru, který bude otáčet s ultrazvukovým senzorem, aby robot mohl reagovat i na nepravidelnou překážku. V první konstrukci budeme vycházet z návodu (příloha 2), ale před střední servomotor přidáme hřídel, která bude kolmo k podložce a bude otáčena servomotorem. Na tuto hřídel přidělame ozubené kolo, na kterém je upevněn ultrazvukový senzor. V posledním kroku je střední servomotor přichycen ke kostře robota. Detailnější popis konstrukce nalezneme v obrázkovém návodu v příloze 3.

Po dokončení konstrukce můžeme změnit implementaci funkce, abychom využili otočného ultrazvukového senzoru. Po zaznamenání překážky se ultrazvukový senzor bude otáčet tak, aby ultrazvukový senzor snímal prostor vzdálenější od černé čáry. Úhel, při kterém už ultrazvukový senzor nezaznamená překážku před ultrazvukovým senzorem, označíme jako krajní úhel. Po zaznamenání krajního úhlu opět vycentrujeme ultrazvukový senzor a robot se na místě otočí o krajní úhel od černé čáry. Následně robota vyšleme rovně pomocí funkce drive().

V druhé části této implementace jede robot rovně a ultrazvukovým senzorem kontroluje, kde se překážka nachází. Při zaznamenání překážky vypočítáme pomocí Pythagorovy věty, jaká bude minimální vzdálenost od překážky. Pomocí referenční hodnoty, která určuje vzdálenost, kterou chceme udržovat od překážky, zjistíme, zda tuto vzdálenost od překážky udržujeme, nebo může nastat kolize s překážkou. Udržování vzdálenosti od překážky řídí P regulátor, který je vydělen vzdáleností, aby při velké blízkosti k překážce měl regulátor větší vliv a překážce se vyhnul. Pokud ultrazvukový senzor ztratil překážku, otáčí se ultrazvukový senzor k černé čáře, aby překážku opět nalezl. Tento postup je opakován, dokud senzor není otočen o 90 ° k černé čáře.

V tuto chvíli nastává problém, protože při konstrukci robota byly použity převody, které nejsou přesné, aby se určil úhel otočení ultrazvukového senzoru. Abychom se nemuseli spoléhat na hodnotu úhlu, kterou získáme ze středního servomotoru, přidáme na obě strany EV3 dotykové senzory. Pomocí těchto dotykových senzorů zjistíme, zda se ultrazvuk dostal do mezní pozice a nemůže se otáčet dál.

Abychom nemuseli ultrazvukový senzor před každou jízdou ručně nastavovat, aby mířil před robota, byla vytvořena funkce kalibrace(), ve které se ultrazvukový senzor otáčí doleva, dokud není sepnut levý dotykový senzor. Ve chvíli, kdy je levý dotykový senzor sepnut, zapamatujeme si úhel na servomotoru a začneme ultrazvukovým senzorem otáčet doprava, dokud není sepnut pravý dotykový senzor. Po sepnutí si opět zapamatujeme úhel servomotoru a vypočítáme střední hodnotu ze dvou zapamatovaných úhlů. Následně pomocí příkazu run_target() necháme ultrazvukový senzor dosáhnout úhlu, který jsme vypočítali jako střední hodnotu maximálního a minimálního úhlu. Po

provedení příkazu, by měl ultrazvukový senzor směřovat rovně před robota. Současnou polohu označíme jako nulový úhel pomocí funkce reset_angle().

```
def kalibrace2():
    motC.run(-100)
    while(!(touchL.pressed())):
        uhelL=motC.angle()
    motC.run(100)
    while(!(touchR.pressed())):
        uhelL=motC.angle()
    motC.stop()
    uhelStred=(uhelL+uhelR)/2
    motC.run_target(90, uhelStred, then=Stop.HOLD, wait=True)
    motC.reset_angle()
```

Při experimentální části docházelo k situaci, že ultrazvukový senzor měl ukazovat před robota, ale byl vychýlen až o 70 °. Tento jev byl způsoben využitím převodových součástek se 4 rameny. Tyto součástky jsou vhodnější spíše pro konstantní pohyb než pro zjišťování úhlu, protože při využití těchto součástek dochází k velké vůli. Abychom vůli zmenšili, použili jsme místo 4 ramenných součástek ozubená kola, která by měla mít menší vůli. Po výměně jsme opět použili stejnou kalibrační funkci. V tomto případě byla výchylka snížena, avšak odchylka od očekávaného úhlu byla až 30 °.

Abychom dosáhli přesnějšího získání úhlu, museli jsme použít otočný systém bez převodových součástek. Z této úvahy plyne, že servomotor musel být umístěn do svislé polohy. Aby bylo možné zapojit kabely do portů na EV3, museli jsme střední servomotor umístit o jednu polohu dopředu, čemuž bránil světelný senzor. Abychom měli vycentrovaný ultrazvukový i světelný senzor, museli jsme světelný senzor připevnit ke středovému servomotoru. Pro bližší informace ke konstrukci prosím využijte obrázkový návod v příloze 3.

Při změně konstrukce byly odstraněny dotykové senzory, které bez převodových součástek nejsou potřeba. Abychom získali hraniční úhly pohybu ultrazvukového senzoru, místo sepnutí dotykového senzoru jsme použili funkce run_until_stalled(), která provede otáčení senzorem do chvíle, než bude senzor zastaven vnější silou. Ve chvíli, kdy je senzor zastaven, funkce vrací úhel, který budeme považovat za mezní. Stejným postupem získáme mezní úhel na druhé straně. Funkce run_util_stalled() je možné použít, protože při konstrukci robota byl ultrazvukový senzor vycentrován a protože robot je osově souměrný. Po provedení kalibrace ultrazvukového senzoru jsme získávali odchylku od očekávané polohy 5 °, což je pro naše využití dostatečné.

```
def kalibraceUltrazvuku():
    uhelL=motC.run_until_stalled(-90, then=Stop.COAST, duty_limit=60)
    uhelR=motC.run_until_stalled(90, then=Stop.COAST, duty_limit=60)
    uhelStred=(uhelL+uhelR)/2
    motC.run_target(90, uhelStred, then=Stop.HOLD, wait=True)
    motC.reset_angle(0)
```

Po vyřešení problému s převody můžeme pokračovat v implementaci vyhnutí se překážky. Pro shrnutí si zopakujeme, že při zaznamenání překážky před robotem ultrazvukový senzor zjistí, kam až překážka zasahuje a poté se robot otočí o krajní úhel. Krajní úhel je úhel, o který se robot musí otočit, aby jeho nejmenší vzdálenost mezi přímou trajektorií robota a překážkou byla rovna zadané konstantě. Krajní úhel určíme jako součet úhlu ultrazvukového senzoru a vnitřní úhel pravoúhlého trojúhelníku. První odvěsna má délku určenou jako referenční vzdálenost od překážky, která je určena na začátku kódu. Druhé délka odvěsny je určena jako vzdálenost mezi překážkou a ultrazvukovým senzorem sečtená se vzdáleností nápravy kol a ultrazvukového senzoru. Z výpočtu (44) zjistíme, o jaký úhel se robot otočí.

Následně robot pomocí funkce drive() jede rovně a pomocí ultrazvukového senzoru kontrolujeme vzdálenost mezi robotem a překážkou, dokud ultrazvuk není natočen o úhel 90 ° k černé čáře. Ve chvíli, kde je senzor otočen o 90°, je robot nejblíže k překážce.

$$\varphi_r = \varphi_k + \varphi_{vz} = \varphi_k + arctan\left(\frac{d_{ref}}{d_{ultra}}\right) \tag{44}$$

Nakonec robot ponechá ultrazvukový senzor v úhlu 90 ° a nastává část, kdy se robot chová jako dvoustavový automat. K prvnímu stavu dochází, pokud ultrazvukový senzor zaznamenává překážku. V tomto stavu robot pomocí funkce drive() jede rovně, aby zamezil kolizi s překážkou. Pokud ultrazvukový senzor nezaznamenává překážku, automat se přepne do druhého stavu, kde robot s využitím funkce drive() rotuje po kružnici tak, aby udržoval předem danou vzdálenost od překážky. Stavový automat je ukončen po nalezení černé čáry pomocí světelného senzoru. Nakonec jako při objíždění po kružnici srovnáme robota s černou čárou pomocí P regulátoru tak, aby světelný senzor ukazoval na hranu černé čáry. Následně robot pokračuje ve sledování černé čáry.

```
def vyhnise():
    distance1=senzorU.distance()
    rychlost=100
    P=50
    refVzdal=130
    #otaci ultrazvukem
    motC.run(90)
    distance=269
    while (distance<270):
        distance=senzorU.distance()
    rob.stop()
    motC.stop()
    krajniUhel=motC.angle()
    r=math.tan(math.radians(krajniUhel))*distance1 +55
    motC.run_target(90, 0, then=Stop.COAST, wait=False)
    krajniUhel=krajniUhel+math.degrees(abs(math.atan(refVzdal/distance)))
    rob.turn(krajniUhel)
    rob.drive(rychlost, 0)
    uhelC=motC.angle()
    while (uhelC>-90):
        uhelC=motC.angle()
        distance=senzorU.distance()
        #sledovani objektu
        while(distance>250):
            motC.run(-90)
            distance=senzorU.distance()
            if motC.angle()<-90:
                break
        motC.stop()
        maxVzdal=abs(math.sin(uhelC)*distance)
        zatoc=-P*(maxVzdal-(refVzdal+67))/distance
        rob.drive(rychlost,zatoc)
    rob.drive(rychlost,0)
    zatoc=-(360*rychlost)/(2*math.pi*r)
    while(senzorL.reflection()>6):
        distance=senzorU.distance()
        if distance<300:
            rob.drive(rychlost,0)
        else:
            rob.drive(rychlost,zatoc)
    rob.stop()
    wait(500)
    P=1
    chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
    while(abs(chybaP)>5):
        chybaP = ((senzorL.reflection()-cerna) *(100/(bila-cerna)))-stred
        zatoc = -( P * chybaP )
        rob.drive(0,zatoc)
```

3 Úloha číšník

3.1 Zadání (15)

Cílem úlohy je sestavit a naprogramovat robota s plošinou, na kterou je volně položen vozík. Robot musí dráhu projet samostatně bez hlasového ovládání, Bluetooth či jiné komunikace. Návod na sestavení plošiny vozíku nalezneme v příloze 4. Konstrukci plošiny lze měnit, ale je nutné přesně dodržet materiál a rozměr plochy plošinky pro umístění vozíku. Úkolem robota je projet dráhu tak, aby vozík z plošiny nespadl ani se vozík nedotkl jiné části robota. Dráha je přímočará, avšak na dráze dochází ke stoupání a klesání, které mají vychýlit vozík z plošiny. Robot by neměl mít větší půdorys než 28 x 28 cm a nesmí používat kluzný podvozek.

Na projetí dráhy je stanoven časový limit. Jízda je ukončena dříve, pokud robot opustí dráhu, robot se zasekne, vozík opustí plošinu nebo se dotkne jiného objektu, který není součástí horní plochy plošinky nebo se tým robota dotkne před dokončením úkolu. Cílem je dráhu projet v co nejkratším čase. Pokud se žádnému týmu nepovede projet celou dráhu, vyhrává tým s největší uraženou vzdáleností.

3.2 Soutěžní plocha (15)

Soutěžní plocha má rozměry 2560 x 1720 mm. Podkladem je laminátová deska ohraničená bočními laminátovými lištami. Nerovný povrch bude vytvářen z bloků o půdorysu 28 x 28 cm a výšce 7,5 cm. Na Obr 31 a Obr 32 nalezneme čtvercový a šikmý modul, ze kterého budou vytvářeny dráhy. Možnou dráhu z modulů nalezneme na Obr 33. Pro sestavení dráhy platí, že maximální stoupání a klesání je 15 °. Další podmínkou je, že maximální změna sklonu mezi dvěma po sobě následujících modulů je 30 °.

Obr 31: Čtvercový modul (16)

Obr 32: Šikmý modul (17)

Obr 33: Soutěžní dráha (18)

3.3 Konstrukce robota

Nejdříve jsme sestavili povinné části, které každý robot musí mít. Jedná se o vozík a plošinu, na kterou bude vozík položen. Návody k sestavení nalezneme v příloze 4. Následně je potřeba zajistit, aby robot jel rovně, protože není nutné zatáčet. Další otázkou je zajištění regulace vozíku na plošině. Můžeme se setkat s dvěma nejčastějšími řešeními. První řešení je pasivní a můžeme si ho představit jako sestavení houpačky, kde místo sedátka bude umístěna plošinka s vozítkem. Výhodou tohoto řešení je absence motoru. V této konstrukci dochází k vyrovnání plošiny na nakloněné rovině působením gravitační síly.

Nevýhodou je, že absence motoru způsobuje, že není možné naklánět plošinou. Z toho plyne, že jediná možnost, jak pohnout s plošinou, je pohyb celého robota dopředu nebo dozadu.

Abychom dosahovali rychlejší regulace a nespoléhali se pouze na gravitační sílu, je možné připojit plošinu k motoru, který bude plošinu naklánět. Toto řešení dovoluje aktivně regulovat plošinu, a proto při správném nastavení regulátoru je možné udržet vozík na plošině při vyšší rychlosti robota.

Pokud robot je poháněn dvěma motory, kterým dáme stejný příkaz, aby se motor otáčel konstantní úhlovou rychlostí, zjistíme, že robot ve většině případů nepojede rovně, jak bychom očekávali. Aby robot jel rovně, je možné použít například funkci drive(), nebo kontrolovat úhel obou motorů a regulovat úhly motorů tak, aby byly shodné. Pokud se chceme této implementaci vyhnout, můžeme použít pouze jeden motor, který bude pohánět obě kola. Díky použití jediného motoru na pohon robota zajistíme, že robot pojede opravdu rovně. Navíc můžeme druhý velký motor využít na regulaci plošiny.

Jak jsem již zmínil, díky použití pouze jednoho motoru na pohon robota, můžeme použít druhý velký servomotor na regulaci plošiny. Podmínka pro konstrukci plošiny se servomotorem je, aby motor byl umístěn vertikálně a při otáčení servomotorem docházelo k naklonění roviny ve směru pohybu robota. Druhou podmínkou je schopnost naklopit rovinu v rozmezí minimálně +- 15 °. Tento úhel je svírán s podložkou, na které se robot nachází. Kdyby nebylo možné dosáhnout tohoto rozmezí, je možné, že regulátor by chtěl vyrovnat plošinu při pohybu po nakloněné rovině, ale nebude to z konstrukčního hlediska možné. Při dodržení těchto podmínek lze stvořit robota, který bude aktivně regulovat vozík na plošině bez převodů. V příloze 4 je možné najít návod na konstrukci robota, se kterým budeme pracovat v dalších podkapitolách.

3.4 Inicializace robota

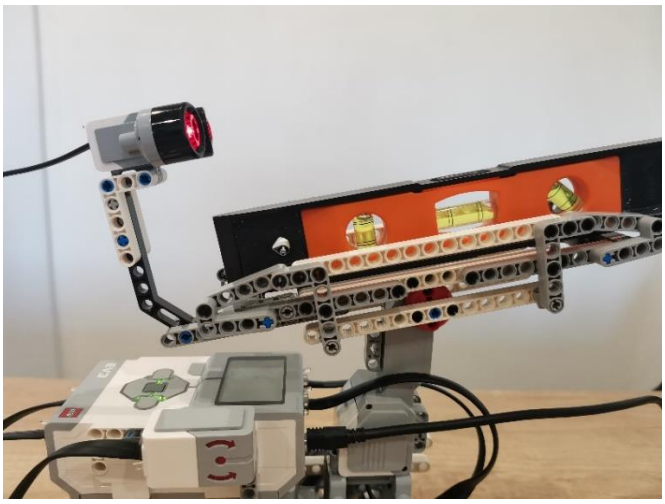
Dle návodu na sestavení robota (příloha 4) lze zjistit, že plošina s robotem je spojena jen pomocí velkého servomotoru. Z konstrukce robota vyplývá, že pokud by motor neovlivňoval polohu plošinky, plošina by nebyla ve vodorovné poloze, ale byla by opřena o kostku EV3. Z toho vyplývá, že před umístěním vozíku na plošinu, musíme nejdříve plošinku vyrovnat do vodorovné polohy.

Z tohoto důvodu jsme se rozhodli před startem robota vytvořit inicializaci. Inicializace spočívá v tom, zavoláme příkaz run_until_stalled(), který zaručí, že motor zastaví až o kostku EV3 a vrátí úhel, při kterém byl zastaven. Nakonec motor změní své natočení na úhel, který je definován jako součet úhlu, který byl vrácen funkcí run_until_stalled() a konstantní hodnotou úhlu. Hodnotu můžeme získat buď experimentálně, nebo můžeme změřit vzdálenost mezi bodem dotyku plošiny s EV3 a motorem držící celou plošinu. Následně změříme i výšku mezi bodem dotyku a osou otáčení motoru. Pomocí goniometrických funkcí získáme, že vnitřní úhel vytvořeného trojúhelníku je 70,9 °. Po přiložení úhloměru k motoru dojdeme ke stejnému úhlu. Z toho plyne, že je nutné změnit úhel o zhruba 20 °. Abychom změnili úhel, použijeme funkci run_target(), kde v argumentu přičteme k meznímu úhlu 20 °. Při provedení programu bylo zjištěno, že pouhým okem se dalo říct, že plošina není ve vodorovné poloze. Je to způsobeno tím, že motor, který má po dosažení požadovaného úhlu udržovat tento úhel, tak než se tak stane, tak plošina klesne. Z toho plyne, že nejlehčí možností, jak určit konstantní úhel bude experimentální metodou.

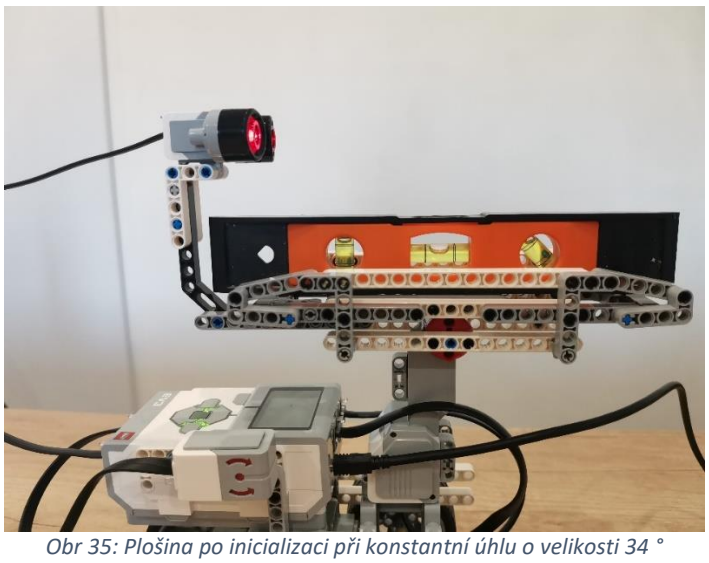
Pro přesnější určení konstantního úhlu budeme potřebovat malou vodováhu. Pro určení úhlu budeme provádět měření, kde v kódu vždy zvětšíme konstantní úhel o 1 °. Po provedení kódu vyčkáme na ustálení plošiny a následně na plošinu položíme malou vodováhu. Vodováhu pokládáme tak, aby těžiště vodováhy bylo co nejblíže k ose otáčení motoru, ale stále musí vodováha být v poloze, aby bylo možné provést měření. Tato podmínka zajistí, že vodováha vytváří nulový moment síly a tedy nevychyluje plošinu z dosavadní polohy. Po provedení měření pro úhly v rozsahu 25 až 35 ° jsme došli k výsledku, že plošina je ve vodorovné poloze pro konstantní úhel o velikosti 31 °.

Po zkušebních jízdách robota bylo zjištěno, že pro správnou regulaci musí být robot plně nabit. Pokud tato podmínka nebyla splněna, motor držící plošinu neměl dostatečnou sílu pro naklápění plošiny o malé úhly. Abychom snížili moment síly, kterým plošina působí na motor, změnili jsme uchycení plošiny motorem, aby se vyskytovalo uprostřed. Aby délka robota nepřesahovala délku 28 cm, gyroskop byl přemístěn na bok kostky EV3. Nový model nalezneme v příloze 5. Změna polohy plošiny navíc umožnila větší naklopení plošiny. Na druhou stranu bylo nutné znova zkalibrovat funkci uvádějící plošinu do vodorovné polohy. Na Obr 34 lze vidět plošinu po inicializaci, pokud jsme ke krajnímu úhlu přičetli vypočtený úhel. Na Obr 35 můžeme vidět plošinu po inicializaci, pokud ke krajnímu úhlu přičteme úhel získaný experimentálně.

Během testování na soutěžní dráze se zjistilo, že pokud by hnací kola byla přední, robot na nakloněné rovině vzhůru kvůli vysoko položenému těžišti většinu váhy přenese na zadní kola. Proto hnací kola nemají dostatečný vliv, a vyjet nakloněnou rovinu je velmi obtížné. Z tohoto důvodu je zadávána záporná rychlost, aby hnací kola by zadní a měly větší účinek při jízdě vzhůru.



Obr 34: Plošina po inicializaci při konstantní úhlu o velikosti 23 °



Obr 35: Plošina po inicializaci při konstantní úhlu o velikosti 34 °

Po provedení inicializace plošiny, kdy plošina se následně nachází ve vodorovné poloze, robot ji udržuje v této pozici a vyčkává, dokud není zmáčknuté jakékoliv tlačítko kromě tlačítka back. Robot vyčkává, abychom mohli na vyrovnanou plošinu umístit vozík. Po umístění vozíku na plošinu vyčkáme

na pokyn startéra a následně zmáčkneme libovolné tlačítko na EV3. Po zmáčknutí tlačítka se bude robot pohybovat dopředu a vyrovnávat plošinu tak, aby vozík zůstal na plošině.

Ukázka kódu:

```
ev3 = EV3Brick()
gyro= GyroSensor(Port.S1)
ultra = UltrasonicSensor(Port.S4) # krajni hodnoty jsou 4 a 13 cm
mot=Motor(Port.A) #kladny jede dopredu
rameno=Motor(Port.D) #kladny je nahoru, zaporny dolu

mot.hold()
uhel0preni=rameno.run_until_stalled(-50, duty_limit=20)
rameno.run_target(100, uhel0preni+31, then=Stop.HOLD)
tlacitka=len(ev3.buttons.pressed())
rameno.reset_angle(0)
while(tlacitka==0):
    tlacitka=len(ev3.buttons.pressed())
```

3.5 Regulace plošiny

V této kapitole si popíšeme, jak vytvořit regulátor, který bude balancovat s vozíkem na plošině. Tato úloha se liší od sledování čáry tím, že vstupní hodnoty v této úloze získáváme ze 2 senzorů. Z toho plyne, že musíme rozdělit, jaký vliv na regulátor bude mít gyroskopický senzor a jaký vliv bude mít ultrazvukový senzor.

Nejdříve si vysvětlíme úlohu gyroskopického senzoru. Pomocí gyroskopického senzoru, který jsme umístili na kostku EV3, lze získat naklonění celého robota. Naklonění celého robota nám pomůže určit úhel nakloněné roviny, po které se aktuálně robot pohybuje a tím zaručit vodorovnou polohu plošiny.

Chování regulátoru si představíme na příkladu. Robot jede po nakloněné rovině, která má úhel stoupání 10 °. Pokud by motor, který reguluje náklon plošiny, nezareagoval na stoupání, došlo by k naklopení plošiny s vozíkem o 10 °, což by zapříčinilo pohyb vozíku, který by z plošiny spadl. Z tohoto důvodu musí motor zareagovat na stoupání tak, že zmenší úhel o 10 °, aby plošina byla opět ve vodorovné poloze. V kódu toho docílíme tak, že získáme hodnotu úhlu z gyroskopu a tuto hodnotu vynásobíme mínus jednou. Výsledek tvoří argument funkce run_target(). Funkce run_target() kromě rychlosti a cíleného úhlu musíme zadat hodnotu False pro nepovinný argument wait. Pokud bychom nevložili argument wait, funkce by vždy čekala, dokud motor nedocílí požadovaného úhlu a až poté by se prováděly další příkazy. Prodleva by nebyla veliká, ale chceme dosáhnout co nejrychlejší odpovědi na změnu naklonění robota, a proto vstupní argument wait má hodnotu False.

Po implementaci kódu získáme regulátor, který po naklonění robota udržuje plošinu ve vodorovné poloze. Toto chování lze vidět na videu v příloze 6.

Ukázka kódu:

```
while(True):
    uhel=gyro.angle()
    rameno.run_target(200, -uhel, then=Stop.HOLD, wait=False)
```

Nyní robot reaguje na nakloněnou rovinu, po které robot jede. Přidáme do regulátoru složku, která bude získávat hodnoty vzdálenosti z ultrazvukového senzoru a regulovat polohu vozíku na plošině, aby

vozík nespadl. V první řadě si musíme určit hodnoty vzdáleností z ultrazvukového senzoru, ve kterých vozík stojí na kraji plošiny. Vozík je v krajních polohách vzdálen od ultrazvukového senzoru 4 cm a 13 cm. V ideálním případě bychom chtěli, aby vozík se po celou dobu jízdy nacházel uprostřed plošiny. Tuto polohu určíme jako střední hodnotu z dvou krajních vzdáleností vozíku. Z toho plyne, že chceme, aby se vozík nacházel po celou dobu jízdy 8,5 cm od ultrazvukového senzoru.

V první řadě zkonstruujeme proporcionální regulátor. Proporcionální regulátor reaguje na vychýlení vozíku z ideální polohy. Pokud vozík bude blíže než 8,5 cm od ultrazvukového senzoru, je nutné zvýšit úhel naklopení, aby se vozík vrátil do žádané polohy. Experimentálně jsme dospěli k proporcionálnímu koeficientu 0,35, který zaručil rychlé chování, které nemělo nestabilní prvky chování jako například rozkmitání celé plošiny.

Ukázkový kód:

```
stred=(40+130)/2
P=0.35
mot.hold()
uhel0preni=rameno.run_until_stalled(-50, duty_limit=20)
rameno.run_target(100, uhel0preni+31, then=Stop.HOLD)
tlacitka=len(ev3.buttons.pressed())
rameno.reset_angle(0)
while(tlacitka==0):
    tlacitka=len(ev3.buttons.pressed())
while(True):
    uhel=gyro.angle()
    chybaP=stred-ultra.distance()
    uhelNaklopeni=-uhel+chybaP*P
    rameno.run_target(200,uhelNaklopeni, then=Stop.HOLD, wait=False)
```

Poté, co jsme určili proporcionální koeficient, můžeme přidat integrační a derivační složku. Při experimentálním zkoušení derivační složky bylo zjištěno, že derivační složka nemá žádný vliv na regulaci vozíku. Je to způsobeno tím, že nečekáme na dosažení úhlu funkcí run_target(). Z toho plyne, že při změně vzdálenosti vozíku od ultrazvukového senzoru se projevuje pouze malá část derivační složky, která není ani pozorovatelná okem. Z toho plyne, že derivační složka by v tomto případě byla zbytečná, a proto ani nebude v regulátoru zastoupena.

Nakonec přidáme integrační složku, která zajistí v nekonečném čase nulovou výchylky vozíku. Opět jako při sledování černé čáry budeme přičítat současnou proporcionální chybu k integrační chybě. Integrační koeficient jsme určili jako konstantu 0,00002. Při této hodnotě integrační složka naklání plošinou tak, aby se vozík opět vrátil do ideální polohy. Pokud implementujeme anti-windup, nebude docházet k překmitům způsobeným integrační složkou. Anti-windup jsme implementovali jako podmínku, zda se vozík nachází maximálně 5 milimetrů od ideální polohy. Pokud je tato podmínka splněna, vynuluje se integrační chyba, čímž zamezíme překmitu způsobeného integrační složkou.

```
stred=(40+130)/2
P=0.35 #0.35
I=0.00002 #0.00002
mot.hold()
uhelOpreni=rameno.run_until_stalled(-50, duty_limit=20)
rameno.run_target(100, uhelOpreni+31, then=Stop.HOLD)
tlacitka=len(ev3.buttons.pressed())
rameno.reset_angle(0)
while(tlacitka==0):
    tlacitka=len(ev3.buttons.pressed())
print("zaciname")
chybaI=0
while(True):
    uhel=gyro.angle()
    chybaP=stred-ultra.distance()
    chybaI=chybaI+chybaP
    if (abs(chybaP) <5):
        chybaI=0
    uhelNaklopeni=-uhel+chybaP*P +chybaD*D+chybaI*I
    rameno.run_target(200,uhelNaklopeni, then=Stop.HOLD, wait=False)
```

3.6 Určení koeficientů

Určení koeficientů probíhalo v této úloze experimentálně, avšak určování koeficientů nebylo nahodilé. Nejdříve jsme určili samostatné chování proporcionální složky. V dalším kroku jsme určili samostatné chování integrační složky. Poté, co jsme určili koeficienty samostatně, spojili jsme chování všech složek regulátoru a sledovali jsme chování celého řízení.

Snižování nebo zvyšování koeficientů probíhalo pomocí sledování čtyř kategorií. První kategorie nám určuje, jak rychle regulátor dosáhne ideální polohy. Při zvyšování integrační a proporcionální složky dochází k rychlejšímu dosažení cílené hodnoty. Druhá vlastnost chování je překmit, který je ve většině využití regulátorů nežádoucí. Při zvyšování integrační a proporcionální složky dochází k vyššímu překmitu, avšak u integrační složky můžeme překmit zmírnit pomocí implementace anti-windupu. Pokud budeme zvyšovat derivační složku, překmit se bude snižovat.

Třetí kategorie určuje dobu ustálení. Tuto kategorii si můžeme představit jako čas, který je potřeba mezi startem regulátoru a dosažení cílené hodnoty s danou výchylkou, avšak po uplynutí času ustálení se nesmí hodnota vychýlit více než je stanovená výchylka. Z toho plyne, že při velkém kmitání kolem cílené hodnoty bude i vyšší čas ustálení. Při zvyšování integrační složky dochází ke zvýšení času ustálení. Pokud bychom chtěli snížit čas ustálení, musíme zvýšit derivační složku. Poslední kategorie určuje chybu v nekonečném čase. Při zvyšování proporcionální složky dochází ke snížení této chyby, ale nikdy nebude chyba nulová. Pokud bychom chtěli nulovou chybu v nekonečném čase, musí v regulátoru být začleněna integrační složka, která toto zajistí. Pro vyšší přehlednost nalezneme v Tab 2 efekt všech zvyšujících se složek regulátoru na všechny čtyři kategorie.

Tab 2: Efekt zvyšujících se složek regulátoru na vlastnosti regulátoru (13)

Složka	Doba dosažení	Překmit	Čas ustálení	Chyba v nekonečném čase
Proporcionální	Snižuje	Zvyšuje	Malý efekt	Snižuje
Integrační	Snižuje	Zvyšuje	Zvyšuje	Eliminuje
Derivační	Malý efekt	Snižuje	Snižuje	Bez efektu

Bibliografie

1. **Watters, Audrey.** Lego Mindstorms: A History of Educational Robots. [Online] 10. duben 2015. [Citace: 27. leden 2022.] http://hackeducation.com/2015/04/10/mindstorms.

2. **šrámek.** Využití robota LEGO Mindstorms EV3 - návrh robota hrajícího na harmoniku pro propagaci FEL. *ČVUT DSpace.* [Online] [Citace: 27. leden 2022.] https://dspace.cvut.cz/handle/10467/83047.

3. **Anton's MINDSTORMS Hacks.** EV3 vs Robot Inventor: LEGO 51515 and 31313 sets compared. *Home – Antons Mindstorms Hacks.* [Online] [Citace: 24. březen 2022.] https://antonsmindstorms.com/2020/10/11/robot-inventor-vs-ev3/.

4. **EDUXE, s.r.o.** 45544-obsah. *EDUXE, distributor učebních pomůcek Brick Soft, DUPLO, LEGO, WeDo a MINDSTORMS.* [Online] [Citace: 28. leden 2022.] https://www.eduxe.cz/files/download/45544-obsah.pdf.

5. **10fb96278755c8f0a051a04156810307--mm2000x2000. [Online] [Citace: 19. březen 2022.] https://im9.cz/iR/importprodukt-orig/10f/10fb96278755c8f0a051a04156810307--mm2000x2000.jpg.**

6. George Robotics Limited. MicroPython - Python for microcontrollers. [Online] [Citace: 27. leden 2022.] https://micropython.org/.

7. The LEGO Group. MINDSTORMS EV3 Support | Everything You Need | LEGO® Education. *Classroom Solutions for STEM and STEAM | LEGO® Education.* [Online] [Citace: 24. leden 2022.] https://education.lego.com/en-us/product-resources/mindstorms-ev3/teacher-resources/python-for-ev3.

8. Pybricks. Pybricks: Python coding with LEGO. [Online] [Citace: 28. leden 2022.] https://pybricks.com/.

9. The LEGO Group. Getting started with LEGO® MINDSTORMS Education EV3 MicroPython — ev3-micropython 2.0.0 documentation. *Pybricks: Python coding with LEGO.* [Online] [Citace: 25. leden 2022.] https://pybricks.com/ev3-micropython/.

10. —. robotics – Robotics — ev3-micropython 2.0.0 documentation. *Pybricks: Python coding with LEGO.* [Online] [Citace: 25. leden 2022.] https://pybricks.com/ev3-micropython/robotics.html.

11. ČVUT FEL. Roboti: Projekt č.1 (Sledování černé čáry). *Kurz: Roboti - B211.* [Online] [Citace: 12. březen 2022.] https://moodle.fel.cvut.cz/mod/page/view.php?id=211905.

12. —. Projekt_c1_cara. *Kurz: Roboti - B211.* [Online] [Citace: 1. duben 2022.] https://moodle.fel.cvut.cz/pluginfile.php/320472/mod_page/content/5/Projekt_c1_cara.jpg.

13. Sluka, James. PID Controller For Lego Mindstorms Robots. [Online] [Citace: 28. únor 2022.] https://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html?fbclid=IwAR0wbv_9Y1URGmZc47RxNknY8SNTSPm-2pxE82hg2c1COLRE88zvBSPx2ZY.

14. ČVUT FEL. Roboti: Projekt č.2 (Sledování černé čáry a vyhýbání se překážce). *Kurz: Roboti - B211.* [Online] [Citace: 12. březen 2022.] https://moodle.fel.cvut.cz/mod/page/view.php?id=211929.

15. —. Cisnik Zadani. *Roboti: Projekt č.5 (ROBO číšník).* [Online] Kurz: Roboti - B211. [Citace: 3. březen 2022.] https://moodle.fel.cvut.cz/mod/page/view.php?id=211943.

16. —. modul_01. *Kurz: Roboti - B211.* [Online] [Citace: 2. duben 2022.] https://moodle.fel.cvut.cz/pluginfile.php/320568/mod_page/content/2/modul_01.jpg.

17. —. Sikmy_modul_01. *Kurz: Roboti - B211.* [Online] [Citace: 3. duben 2022.] https://moodle.fel.cvut.cz/pluginfile.php/320568/mod_page/content/2/Sikmy_modul_01.jpg.

18. —. 02. [Online] [Citace: 1. duben 2022.] https://moodle.fel.cvut.cz/pluginfile.php/320568/mod_page/content/2/02.jpg.

19. The LEGO Group. Getting started with LEGO® MINDSTORMS® Education EV3 MicroPython. [Online] [Citace: 2. duben 2022.] https://education.lego.com/v3/assets/blt293eea581807678a/bltb470b9ea6e38f8d4/5f8802fc4376310c19e33714/getting-started-with-micropython-v2_enus.pdf.

Přílohy

1) PDF moje dokumentace funkcí

2) PDF base návod robota https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/building-instructions Driving base

3) návod na pohyblivý ultrazvuk

4) návod na sestavení plošiny-puvodni

5) návod na plošinu - nový

6) video na gyro