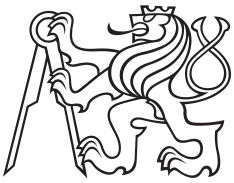
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

Firmware pro Open-Cube (alternativní řídicí LEGO kostku)

Václav Jelínek

Vedoucí: Ing. David Novotný
Obor: Kybernetika a robotika
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jelínek** Jméno: **Václav** Osobní číslo: **499219**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Firmware pro Open-Cube (alternativní řídicí LEGO kostku)

Název bakalářské práce anglicky:

Firmware for Open-Cube (alternative LEGO control brick)

Pokyny pro vypracování:

V rámci bakalářské práce realizujte firmware pro projekt Open-Cube (alternativní řídicí kostka pro LEGO NXT/EV3). Firmware (přesněji řečeno framework) by měl umožnit snazší programování kostky, aby bylo možné její nasazení ve výuce již v prvním ročníku, v předmětu Roboti. Přístup k senzorům a motorům by měl být přímočarý a bez nutných „low-level“ operací a znalostí.
Framework by měl umožňovat přístup ke všem NXT (ideálně i EV3) senzorům, měření interní IMU jednotkou. Dále by mělo být možné ovládat motory, zpracovávat jejich enkodéry a na pozadí frameworku regulovat rychlost/polohu motorů. Další funkcionalitou by měl být obousměrný přenos dat pomocí ESP32 do PC případně mobilu v reálném čase. Nakonec je požadavek na vytvoření uživatelského menu na displeji kostky a ovládání pomocí tlačítek (výběr a spouštění uživatelského kódu). K frameworku bude vypracována důkladná dokumentace všech funkcionalit a kód bude řádně okomentován pro případné budoucí rozšiřování.

Seznam doporučené literatury:

[1] Definitive Guide to ARM (R) Cortex (R)-M0 and Cortex-M0+ Processors, Joseph Yiu, 2015, ISBN: 0128032774
[2] LEGO MINDSTORMS Education EV3 developer kits, available at: https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/developer-kits
[3] Raspberry Pi Pico documentation, available at: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html
[4] Micropython documentation, available at: https://micropython.org/download/rp2-pico/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Novotný katedra měření FEL

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

Datum zadání bakalářské práce: **03.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce:
do konce letního semestru 2023/2024

Ing. David Novotný podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. _____ _____

Datum převzetí zadání Podpis studenta

Poděkování

Děkuji Ing. Davidu Novotnému a Ing. Vojtěchu Petruchovi, Ph.D. za pomoc při zpracování bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 26. května 2023

Abstrakt

Open-Cube je projekt iniciovaný na katedře měření ČVUT FEL, jehož cílem je vyvinutí alternativní řídicí kostky pro ovládání LEGO NXT, EV3 a dalších senzorů a motorů.

První část textu popisuje aktuální stav projektu, tedy první navrženou a sestavenou verzi hardwaru. Dále porovnává EV3, NXT a SPIKE řídicí kostky s vyvíjenou Open-Cube kostkou.

Druhá část textu se zaměřuje na vlastní práci, tedy implementaci firmwaru, jehož součástí je framework (menu s možností procházení a spuštění programů, správa periferií), ovládání komponent kostky a ovládání NXT senzorů a motorů.

Poslední část ověřuje funkčnost hardwaru a aktuálního řešení firmwaru na dvou demonstračních úlohách. První z nich je klasická úloha robotu sledujícího čáru. Druhá, složitější úloha, je robot balancující na dvou kolech (segway).

Řídicí kostka se bude využívat pro výuku studentů prvního ročníku programu Kybernetika a robotika v předmětu Roboti a na Robosoutěži, pořádané ČVUT FEL. Veškeré podklady (HW schémata, firmware, dokumentace, demonstrační úlohy) budou po dokončení vývoje volně přístupné.

Klíčová slova: Open-Cube, řídicí kostka, firmware, framework, RP2040, LEGO, NXT

Vedoucí: Ing. David Novotný
České vysoké učení technické v Praze,
Fakulta elektrotechnická,
Technická 2,
Praha

Abstract

Open-Cube is a project initiated by the Department of Measurement at CTU FEE, which aims to develop an alternative control cube for controlling LEGO NXT, EV3, and other sensors and motors.

The first part of the text describes the current status of the project, i.e. the first designed and built version of the hardware. It also compares the EV3, NXT, and SPIKE control cubes with the Open-Cube under development.

The second part of the text focuses on the actual work, i.e. the implementation of the firmware, which includes the framework (menu with the ability to browse and run programs, peripheral management), control of the cube components, and control of the NXT sensors and motors.

The last section verifies the functionality of the hardware and current firmware solution on two demonstration tasks. The first is a classic line-following robot task. The second, more complex task, is a robot balancing on two wheels (segway).

The control cube will be used for teaching first-year students of the Cybernetics and Robotics program in the Robots course and at the robotic competition Robosoutěž, organized by CTU FEE. All materials (HW schematics, firmware, documentation, demonstration tasks) will be freely available after the development is completed.

Keywords: Open-Cube, control brick, firmware, framework, RP2040, LEGO, NXT

Title translation: Firmware for Open-Cube (alternative LEGO control brick)

Obsah

1 Úvod 1

2 Popis řídicí kostky 3

2.1 Hardware 3

2.1.1 Komponenty 3

2.1.2 Rozhraní 5

2.2 Ochranný kryt 6

2.3 Porovnání řídicích kostek 7

3 Implementace firmwaru 11

3.1 Framework 11

3.1.1 Objekt robot 11

3.1.2 Menu 13

3.1.3 Ochrana před podvybitím baterií 13

3.1.4 Struktura paměti mikrokontroleru 13

3.1.5 Přidávání knihoven a spustitelných programů 14

3.2 Komponenty kostky 14

3.2.1 Tlačítka 14

3.2.2 Piezoelektrický reproduktor . 14

3.2.3 LED 15

3.2.4 Displej 15

3.2.5 Gyroskop a akcelerometr 15

3.2.6 GPIO expander 15

3.2.7 AD převodník 15

3.2.8 ESP32 16

3.3 NXT moduly 16

3.3.1 Světelný senzor 16

3.3.2 Dotykový senzor 19

3.3.3 Ultrazvukový senzor 19

3.3.4 Zvukový senzor 20

3.3.5 Servomotor 21

4 Demonstrační úlohy 25

4.1 Robot sledující čáru 25

4.1.1 Intenzita světla 26

4.1.2 Řízení motorů 26

4.1.3 Komunikace s počítačem 26

4.2 Balancující robot 27

4.2.1 Úhel náklonu 28

4.2.2 Ujetá vzdálenost a rychlost . . 29

4.2.3 Řízení motorů 29

4.2.4 Komunikace s počítačem 30

5 Další práce 33

6 Závěr 35

Literatura 37

A Kód 39

B Videonahrávky demonstračních úloh 41

C Dokumentace Open-Cube 43

Obrázky

2.1 Fotografie sestavené řídicí kostky 3
2.2 Blokový diagram zapojení řídicí kostky 4
2.3 Fotografie řídicí kostky ze stran . . 4
2.4 Odstraněný vrchní kryt pro přístup k bateriím 7
2.5 Rozměry ochranného krytu v milimetrech 7
2.6 NXT řídicí kostka 8
2.7 EV3 řídicí kostka 8
2.8 SPIKE řídicí kostka 8

3.1 NXT světelný senzor 17
3.2 Přechodný děj napětí na měřícím rezistoru světelného senzoru při zapnutí LED 17
3.3 Napětí na měřícím rezistoru světelného senzoru při blikání LED 18
3.4 NXT dotykový senzor 19
3.5 NXT ultrazvukový senzor 19
3.6 I2C žádost ultrazvukovému senzoru o změřenou vzdálenost 20
3.7 NXT zvukový senzor 21
3.8 NXT servomotorr 21
3.9 Odezva regulátoru polohy motoru na skok 22
3.10 Odezva regulátoru rychlosti motoru na skok 23

4.1 Robot sledující čáru 25
4.2 Simulinkový model robotu sledujícího čáru 27
4.3 Graf vlivu PID složek regulátoru na variabilní složku výkonu při sledování čáry 27
4.4 Balancující robot 28
4.5 Graf porovnání úhlu z akcelerometru, gyroskopu a fúzí dat obou senzorů 30
4.6 Schéma propojení regulátorů balancujícího robotu 31
4.7 Simulinkový model balancujícího robotu 31
4.8 Vizualizace stavů balancujícího robotu 32

Tabulky

2.1 Porovnání řídicích kostek 9

Kapitola 1
Úvod

Fakulta elektrotechnická využívá stavebnice LEGO Mindstorms NXT a EV3 k výuce studentů prvního ročníku oboru Kybernetika a robotika v předmětu Roboti a také při pořádání Robosoutěže pro základní a střední školy. Výroba těchto stavebnic byla již ukončena a LEGO nyní prodává novou stavebnici SPIKE. Součástí této stavebnice je nová řídicí kostka, senzory a motory, které nejsou zpětně kompatibilní se zmíněnými předchozími generacemi a neposkytují funkcionality vhodné pro výukové účely. Jelikož Fakulta vlastní desítky NXT a EV3 stavebnic, došlo na katedře měření k rozhodnutí vyvinout alternativní řídicí LEGO kostku s názvem Open-Cube, která by umožňovala připojení klasických LEGO dílů, využití NXT a EV3 senzorů a motorů a poskytovala další funkcionality pro účely výuky studentů.

Na vývoji kostky pracuje skupina vyučujících a studentů Fakulty elektrotechnické. Hardware Open-Cube byl navržen Ing. Davidem Novotným tak, aby kostka byla kompatibilní s klasickými LEGO součástkami, NXT a EV3 senzory a motory. Základní knihovny pro ovládání komponent kostky a NXT senzorů byly z části naprogramovány studenty v rámci předmětu Práce v týmu v magisterské etapě programu Kybernetika a Robotika. Studenti neměli k dispozici hotový hardware pro otestování funkčnosti, a tak bylo nutné většinu této práce předělat. Komunikace a ovládání EV3 senzorů je obsahem bakalářské práce studenta Jakuba Vaňka. Obsahem této bakalářské práce je implementace firmwaru kostky, ovládání komponent kostky, ovládání NXT senzorů a motorů, sjednocení již implementovaných knihoven, otestování funkčnosti kostky na demonstračních úlohách a sepsání dokumentace funkcionalit kostky.

Žáci základních a středních škol a studenti prvního ročníku vysoké školy často nemají zkušenosti s programováním. Python je vhodný programovací jazyk pro začátky a seznámení se se základy programování, algoritmizace a řízení robotů. Je to také první programovací jazyk, se kterým se studenti prvního ročníku programu Kybernetika a robotika setkají v předmětu Algoritmy a programování. Souběžně s tímto předmětem je ve studijním plánu i předmět Roboti, ve kterém budou studenti řídicí kostku využívat. Z těchto důvodů byl vývojový jazyk pro Open-Cube zvolen MicroPython, což je implementace Pythonu 3 optimalizovaná pro běh na mikrokontrolerech.

Kapitola 2
Popis řídicí kostky

Na obrázcích 2.1, 2.3 a 2.4 lze vidět realizaci kostky při pohledu ze shora a ze stran, rozměry kostky jsou na obrázku 2.5.

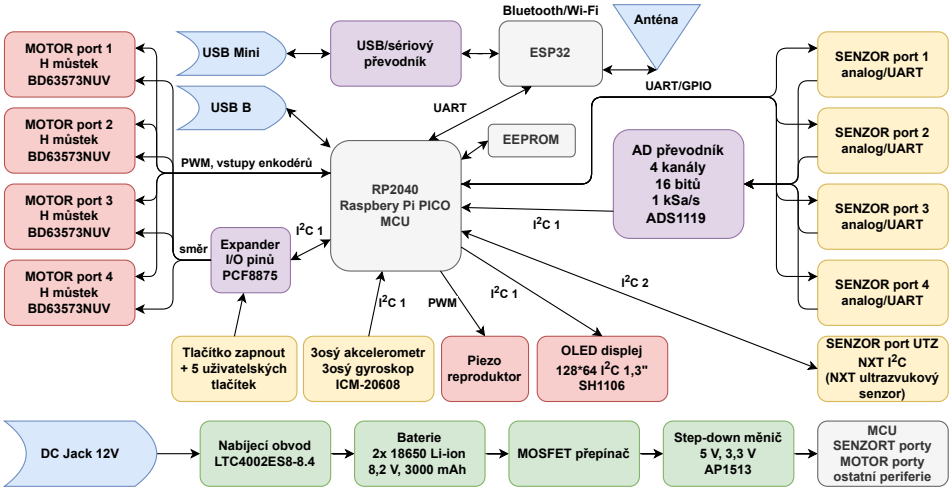
Obrázek 2.1: Fotografie sestavené řídicí kostky.

2.1 Hardware

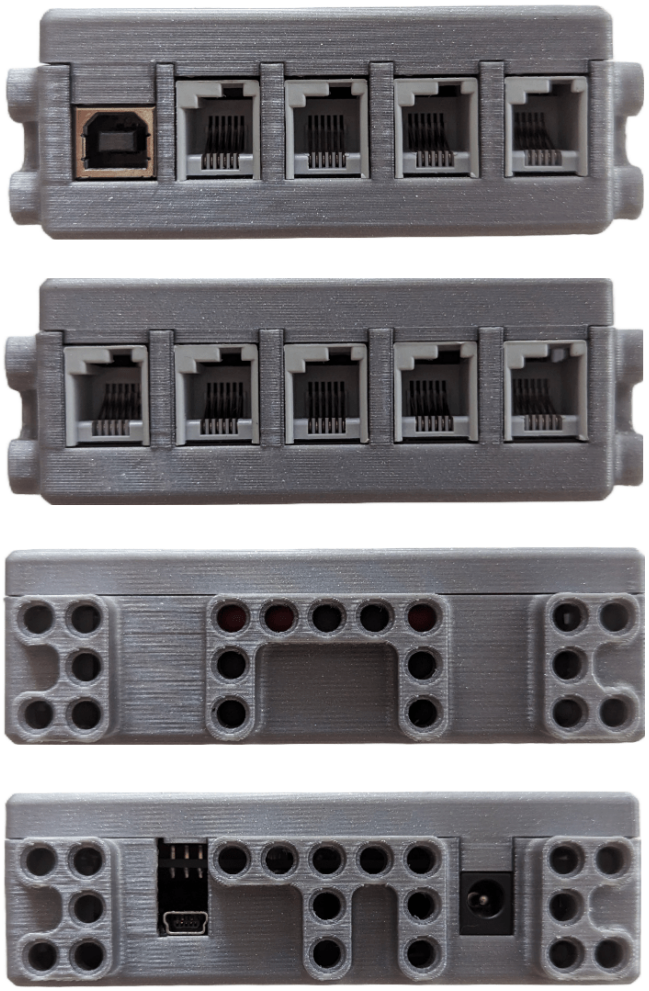
Obrázek 2.2 graficky popisuje propojení hardwaru kostky.

2.1.1 Komponenty

Jádrem řídicí kostky je mikrokontroler Raspberry PI RP2040. Pro bezdrátovou komunikaci (Bluetooth, Wi-Fi) s externím zařízením je kostka vybavena modulem ESP32. Kostka je vybavena tříosým akcelerometrem a gyroskopem ICM-20608-G, piezoelektrickým reproduktorem pro zvukovou signalizaci, AD převodníkem ADS1119 pro získávání dat z analogových senzorů a GPIO expanderem PCF8875



Obrázek 2.2: Blokový diagram zapojení řídicí kostky



Obrázek 2.3: Fotografie řídicí kostky ze stran.

pro rozšíření počtu pinů mikrokontroleru. Napájení je zprostředkováno dvěma 18650 Li-Ion bateriemi.

Mikrokontroler

Raspberry Pi RP2040 je 32bitový dvoujádrový ARM Cortex-M0+@133MHz mikrokontroler s pamětí SRAM o velikosti 264 kB, se 30 GPIO piny, z nichž 4 umožňují analogový vstup, 2 sběrnicemi UART, 2 SPI řadiči, 2 I2C řadiči, 16 PWM kanály a 8 stavovými automaty programovatelných I/O pinů. [14] Oproti ostatním mikrokontrolerům není RP2040 příliš výkonný, ale zato je velmi levný (cena se pohybuje okolo 30 Kč [15])) a v aktuální čipové krizi také relativně dostupný.

Gyroskop a akcelerometr

ICM-20608-G je 3osý gyroskop s programovatelným plným rozsahem ±250, ±500, ±1000, ±2000, deg/s a 3osý akcelerometr s programovatelným plným rozsahem ±2g, ±4g, ±8g, ±16g s integrovanými AD převodníky a digitálním výstupem. Senzor poskytuje programovatelné digitální filtry. [3, 4] Senzor je k mikrokontroleru připojen sběrnicí I2C 1 a pinem přerušení.

GPIO expander

PCF8875 je 16bitový I/O expander umožňující připojení dalších 16 vstupů či výstupu k mikrokontroleru. [13] K Expanderu jsou připojeny vstupy a výstupy, které nevyžadují příliš častou obsluhu, jsou jimi vstupy pro tlačítko zapnutí/vypnutí a uživatelská tlačítka, výstupy pro zapnutí a ovládání H můstku motorů, pro LED na plošném spoji a pro odpojení kostky od napájení. Expander je k mikrokontroleru připojen sběrnicí I2C 1 a pinem přerušení.

AD převodník

ADS1119 je 16bitový 4kanálový převodník se vzorkovací frekvencí 1 kSa/s umožňující měření napětí každého z kanálů proti zemi nebo diferenční měření napětí mezi kanály. [1] Převodník slouží k měření výstupních hodnot připojených analogových senzorů. K mikrokontroleru je připojen sběrnicí I2C 1 a pin přerušení není připojen.

ESP32

ESP32 je bezdrátový komunikační modul umožňující připojení přes Bluetooth nebo Wi-Fi. V řídicí kostce slouží ke komunikaci a výměně dat s externím zařízením (nastavování konstant za běhu programu, ukládání a vizualizace dat o aktuálním stavu kostky). Modul je k mikrokontroleru připojen sběrnicí UART 0.

2.1.2 Rozhraní

Rozhraní kostky tvoří tlačítko pro zapnutí/vypnutí, 5 uživatelských tlačítek pro ovládání, tlačítko boot pro spuštění mikrokontroleru v režimu pro nahrání firmwaru,

OLED displej SH1106, čtyři porty pro připojení motorů, čtyři porty pro připojení NXT, EV3 nebo jiných senzorů a jeden port pro připojení NXT ultrazvukového senzoru. Programování RP2040 je umožněno portem USB B a programování ESP32 portem USB mini. Pro nabíjení baterií slouží 12 V DC Jack.

Tlačítka

Tlačítko pro zapnutí/vypnutí a 5 uživatelský tlačítek pro ovládání kostky (doleva, doprava, OK, nahoru, dolů) jsou k mikrokontroleru připojeny přes I/O expander. K expanderu jsou připojeny přes pull down rezistory.

Displej

SH1106 je černobílý OLED displej s rozlišením 128x64. Displej je k mikrokontroleru připojen sběrnicí I2C 1.

Porty pro připojení motorů

Čtyři 6pinové porty, s upraveným RJ-12 konektorem, pro připojení motorů jsou vybaveny ovladačem motoru s H můstkem BD63573NUV. Porty jsou připojeny na zdroj napětí +5 V a k zemi. Dva piny portu jsou připojeny k ovladači motoru. Další dva piny jsou výstupy z enkodéru a jsou připojeny ke GPIO mikrokontroleru.

Port pro připojení NXT ultrazvukového senzoru

Port je 6pinový s upraveným RJ-12 konektorem. NXT ultrazvukový senzor je digitální senzor, který komunikuje, na rozdíl od ostatních senzorů, po I2C sběrnici, proto je pro něj vyhrazen dedikovaný port. Port je připojen na zdroj napájení +5 V a k zemi. K mikrokontroleru je připojen sběrnicí I2C 2 a je jediným zařízením, které tuto sběrnici využívá.

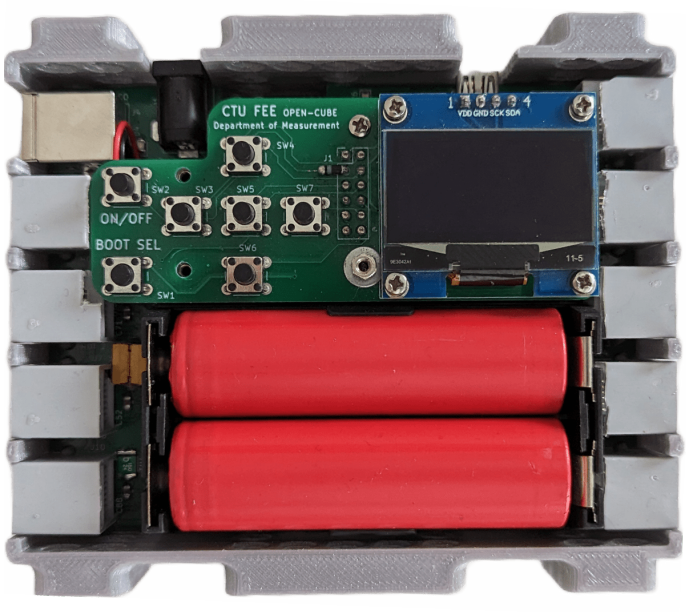
Porty pro připojení ostatních senzorů

Čtyři 6pinové porty, s upraveným RJ-12 konektorem, pro připojení NXT, EV3 nebo jiných senzorů jsou připojeny na zdroj napětí +5 V s omezovačem proudu a k zemi. Každý z portů je připojen k samostatnému kanálů AD převodníku, dva piny portu jsou připojeny ke GPIO mikrokontroleru a poskytují komunikaci přes softwarový nebo hardwarový UART. Porty 2 a 4 umožňují pouze softwarový UART a porty 1 a 3 umožňují softwarový i hardwarový UART sběrnicí mikrokontroleru UART 1.

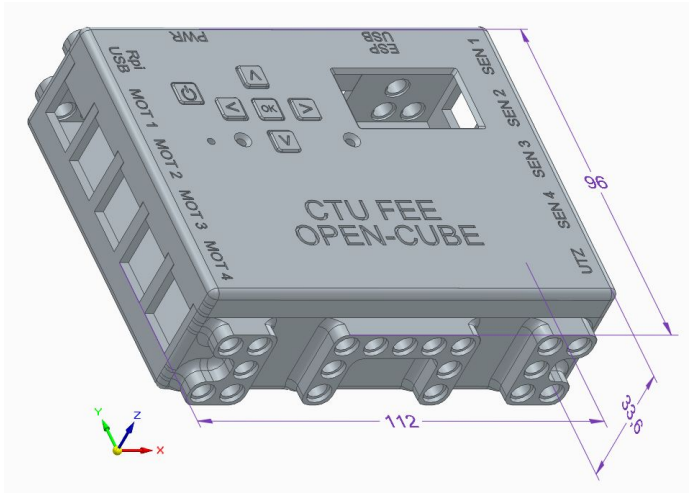
2.2 Ochranný kryt

Kryt se skládá ze spodní a vrchní části. Obě části jsou k plošnému spoji přišroubovány dvěma šrouby. Kryt je vytisknut na klasické FDM 3D tiskárně z materiálu PLA, umožňuje ochranu všech komponent a bezpečný přístup k portům a tlačítkům kostky a je navržený tak, aby k němu bylo možné připevnit klasické LEGO součástky

(obrázek 2.3). Po odšroubování vrchní části krytu je umožněn přístup k bateriím a jejich výměna (obrázek 2.4). Model krytu s popsanými rozměry je na obrázku 2.5.



Obrázek 2.4: Odstraněný vrchní kryt pro přístup k bateriím



Obrázek 2.5: Rozměry ochranného krytu v milimetrech.

2.3 Porovnání řídicích kostek

Řídicí kostky NXT, EV3, SPIKE a Open-Cube jsou na obrázcích 2.6, 2.7, 2.8 a 2.1. Parametry kostek jsou porovnány v tabulce 2.1. Open-Cube může zaujmout především cenou součástek. Dále veškerá hardwarová schémata, firmware, model krytu, dokumentace a manuály budou po dokončení vývoje volně k dispozici.

Obrázek 2.6: NXT řídicí kostka. [10]

Obrázek 2.7: EV3 řídicí kostka. [8]

Obrázek 2.8: SPIKE řídicí kostka. [6]

	NXT	EV3	SPIKE	Open-Cube
Procesor	ARM7	ARM9	ARM Cortex-M4	ARM Cortex-M0+
Paměť	64 KB RAM, 256 KB flash	64 MB RAM, 16 MB flash	320 KB RAM, 1 MB flash	264 KB RAM, 2 MB flash
Operační systém	vlastní	Linux	MicroPython	MicroPython
Vstupní porty	4 RJ-12	4 RJ-12	6 LPF2 (společné)	5 RJ-12
Výstupní porty	3 RJ-12	4 RJ-12	6 LPF2 (společné)	4 RJ-12
SD karta	ne	Micro SD 2-32 GB	ne	ne
Napájení	6xAA/akumulátor	6xAA/akumulátor	akumulátor	2x18650 Li-ion
Displej	100x64 černobílý LCD	178x128 černobílý LCD	5x5 LED	128x64 černobílý OLED
Zvuk	reproduktor 8-bit, 16 KHz	reproduktor PWM, zesilovač zvuku	reproduktor 12-bit, 16 KHz	piezoelektrický reproduktor
Tlačítka	4	6	3	6
IMU	ne	ne	ano	ano
Bluetooth	ano	ano	ano	ano
Wi-Fi	ne	ano	ne	ano
Cena	2 890 Kč (neoficiální) LEGO již neprodává	14 999 Kč (neoficiální) LEGO již neprodává	11 999 Kč (stavebnice)	∼2 600 Kč (součástky)

Tabulka 2.1: Porovnání řídicích kostek. [5, 6, 7, 9, 11, 16]

Kapitola 3
Implementace firmwaru

Hlavním cílem firmwaru je poskytnout uživateli jednoduché rozhraní pro ovládání kostky a zároveň zajistit bezpečnou inicializaci a deinicializaci všech periferií. K dosažení tohoto cíle byl zvolen přístup s vytvořením globálního objektu `robot`, který je inicializován při zapnutí kostky a lze ho využívat ve spuštěném uživatelském programu. Fungování objektu je popsáno v následující sekci.

Původní implementace a zadání úkolu počítalo s implementací firmwaru v jazyce MicroPython. V průběhu implementace a při testování bylo zjištěno, že rychlost běhu kódu je nedostatečná (především pro obsluhu přerušení enkodérů motorů), a tak byla většina knihoven implementována v jazyce C. Tento kód je zkompilovaný a jsou tak zabudovány nové moduly do základního MicroPython firmwaru. Pro zrychlení importování knihoven jsou moduly implementované v jazyce MicroPython předkompilované. MicroPython firmware, moduly v jazyce C a moduly v jazyce MicroPython tvoří Open-Cube firmware.

3.1 Framework

3.1.1 Objekt robot

Globální objekt `robot` poskytuje funkce pro inicializace a deinicializace všech periferií. Objekt se incializuje automaticky při zapnutí kostky. Některé základní periferie (tlačítka, displej, baterie) jsou inicializovány souběžně s objektem `robot`, ostatní periferie mohou být inicializovány a deinicializovány na žádost z uživatelského programu.

Objekt `robot` uchovává a spravuje objekty pro ovládání jednotlivých periferií. Tyto objekty jsou vytvořeny při inicializaci konkrétní periferie a jsou přístupné z objektu `robot`. Při deinicializaci je nejdříve periferie vypnuta (zastavení motorů, vypnutí LED světelného senzoru atd.) a následně se objekt periferie odstraní ze seznamu inicializovaných periferií.

Na objektu můžeme volat metody pro inicializaci periferie: `init_motor`, `init_sensor`, `init_esp_uart`. Analogicky můžeme volat metody pro deinicializace: `deinit_motor`, `deinit_sensor`, `deinit_esp_uart` a `deinit_all`. Poslední zmíněnou metodou se deinicializují všechny uživatelem inicializované periferie. Tato metoda se volá vždy po ukončení programu, čímž je zajištěno správné opětovné spuštění dalšího programu.

Struktura objektu robot je následující:

```
robot
├── battery
├── buttons
├── buzzer
├── display
├── esp_uart
├── led
├── motors[4]
└── sensors
    ├── ultra
    ├── gyro
    ├── touch[4]
    ├── light[4]
    └── sound[4]
```

Následující kód ukazuje jednoduchý uživatelský program, který ovládá periferie kostky za použití globálního objektu `robot`.

Kód 3.1: Použití globálního objektu robot.

```
# Importováni funkce pro uspání programu
from time import sleep
# Importováni konstant tlačítek, portů a senzorů
from lib.robot_consts import Button, Port, Sensor

# Definice programu
def main():
  # Přistoupení ke globalní proměnné robot
  global robot

  # Inicializace motoru na portu motoru 2
  robot.init_motor(Port.M2)
  # Incializace NXT dotykového senzoru na portu senzoru 3
  robot.init_sensor(sensor_type=Sensor.TOUCH, port=Port.S3)

  # Smyčka, čekající na ukončení programu tlačítkem na kostce
  while True:
    # Získání napětí na beteriích
    battery_voltage = robot.battery.voltage()
    # Získání stavu dotykového senzoru
    touch_pressed = robot.sensors.touch[Port.S3].pressed()

    # Pokud je dotykový senzor stisknutý, nastaví motor na plný
    # výkon (100 %), jinak zastaví motor
    if touch_pressed:
      robot.motor[Port.M2].set_power(100)
    else:
      robot.motor[Port.M2].set_power(0)
```

```
    # Vymazání displeje
    robot.display.fill(0)
    # Zobrazení napětí na bateriích a stavu dotykového senzoru
    robot.display.text(
    '{:.2f}V'.format(battery_voltage), 0, 0, 1)
    robot.display.text(str(touch_pressed), 0, 10, 1)
    robot.display.show()

    # Získání stavu tlačítek na kostce
    buttons = robot.buttons.pressed()
    # Ukončení programu, pokud je levé tlačítko stisknuté
    if buttons[Button.LEFT]:
      # Deinicializace všech uživatelem inicializovaných
      # periferií (motoru a tlačítka)
      robot.deinit_all()
      break

    # Uspání programu na 1 sekundu
    sleep(1)

# Spuštění programu
main()
```

3.1.2 Menu

Po zapnutí kostky je na displeji zobrazeno menu se seznamem uživatelských programů a s indikací napětí na beteriích. Seznam programů lze procházet tlačítky nahoru a dolů. Tlačítkem OK je aktuálně vybraný program spuštěn a ovládání displeje je uvolněno. Po ukončení nebo při chybě uživatelského programu se na displeji opět zobrazí menu a lze spustit další program.

3.1.3 Ochrana před podvybitím baterií

Po zapnutí kostky a při běhu jakéhokoliv programu je periodicky měřeno napětí na bateriích. Pokud změřené napětí klesne pod hodnotu 6,5 V, je po čtyřech sekundách kostka vypnuta. V tomto intervalu je vybití baterií signalizováno zvukem piezoelektrického reproduktoru a na displeji je zobrazena varovná hláška.

3.1.4 Struktura paměti mikrokontroleru

V hlavním adresáři se nachází soubor soubor `main.py` a adresáře `lib` a `programs`. Soubor `main.py` je hlavní program, který je ve výchozím nastavení mikrokontroleru spuštěn při zapnutí kostky. Adresář `lib` obsahuje knihovny, které jsou k dispozici pro uživatelské programy. Adresář `programs` obsahuje uživatelské programy. Uživatelský program je buďto soubor s příponou `.py`, nebo adresář obsahující soubor `main.py`.

Ukázka struktury adresáře kostky se vzorovým jednosouborovým programem `vzor1` a programem v adresáři `vzor2`:

```
RP2040 device
├── lib
│   └── ...
├── programs
│   ├── vzor2
│   │   ├── lib
│   │   │   └── vzor2_lib.py
│   │   └── main.py
│   └── vzor1.py
└── main.py
```

3.1.5 Přidávání knihoven a spustitelných programů

Základní knihovny pro ovládání kostky, senzorů a motorů, v jazycích C a MicroPython, jsou zkompilovány a spolu se základním MicroPython firmwarem obsaženy v Open-Cube firmwaru kostky. Další knihovny lze přidat do adresáře `lib`. Uživatelské programy lze přidat do adresáře `programs`. Takovéto programy jsou spustitelné z menu (sekce 3.1.2). Programy a knihovny lze přidávat po připojení kostky k počítači USB kabelem pomocí editoru jako je např. Thonny.

3.2 Komponenty kostky

3.2.1 Tlačítka

Tlačítka (tlačítko zapnutí/vypnutí, 5 uživatelských tlačítek) jsou připojena k mikrokontroleru přes I/O expander, který má pin přerušení připojen k mikrokontroleru. Při vyvolání přerušení od expanderu je aktualizován stav všech tlačítek a uživateli je přístupný poslední zaznamenaný stav. Pokud je kostka vypnutá a uživatel stiskne tlačítko pro zapnutí/vypnutí, dojde k připojení napájení mikrokontroleru, aktivuje se přidržení napájení a na displeji se zobrazí menu (sekce 3.1.2). Pokud je kostka zapnutá a uživatel stiskne toto tlačítko, dojde k deaktivaci přidržení napájení a kostka se vypne. Funkce uživatelských tlačítek není předem definovaná a je ponechaná na uživateli.

Obsluha přerušení je důležitá především proto, že tlačítko zapnutí/vypnutí je připojeno k expanderu a odpojení napájení periferií a mikrokontroleru je řešeno programově. V následující verzi hardwaru kostky je plánované hardwarové odpojení napájení. To zajistí, že se kostka vypne i v případě, že v mikrokontroleru dojde k neošetřené chybě a nebude reagovat na žádost o přerušení od expanderu.

3.2.2 Piezoelektrický reproduktor

Reproduktor je ovládaný PWM signálem mikrokontroleru. Je použit pro signalizaci vybití baterií a z uživatelského programu lze reproduktor ovládat nastavením

frekvence a střídy PWM signálu. Při zapnutí kostky a po ukončení programu je reproduktor vypnut nastavením střídy signálu na 0 %.

3.2.3 LED

LED je umístěna na plošném spoji uvnitř kostky a při přišroubovaném krytu není její zapnutí příliš vidět. Uživatel může LED zapnout nebo vypnout. Při zapnutí kostky a po ukončení uživatelského programu je LED vypnuta.

3.2.4 Displej

Knihovna pro ovládání displeje, jako jediná, zůstala téměř beze změn ve stavu před zpracováním této práce. Na displeji lze zobrazovat, text, jednoduché geometrické útvary a grafy zapisováním do frame bufferu, který je do paměti displeje přenesen a zobrazen na příkaz programu. Při spuštění kostky a po ukončení uživatelského programu je na displeji zobrazeno menu (sekce 3.1.2). Před spuštěním uživatelského programu je frame buffer smazán, na displeji je zobrazena černá obrazovka a ovládání displeje je přenecháno spuštěnému programu.

3.2.5 Gyroskop a akcelerometr

Gyroskop měří úhlové zrychlení v deg/s a akcelerometr měří poměr zrychlení senzoru vůči tíhovému zrychlení. Oba senzory měří hodnoty ve třech osách (x, y, z). Při inicializaci senzoru uživatelem je zapsáním do registrů nastavena konfigurace gyroskopu plného rozsahu $\pm 250\,\mathrm{deg/s}$ a akcelerometru $\pm 1000g$, je nastaveno generování pulzu přerušení při změření nových dat, dále digitální dolnopropustný filtr a dělič vzorkovací rychlosti tak, aby nová data byla změřena každé 0,003 sekundy. [3, 4] Po deinicializaci je senzor vypnut z důvodu šetření energie.

3.2.6 GPIO expander

K expanderu jsou připojeny vstupy tlačítka pro zapnutí/vypnutí a 5 uživatelských tlačítek, 8 výstupů pro ovládání motorů (2 pro každý z motorů – zapnutí a směr), výstup pro ovládání LED a výstup pro podržení napájení. Expander se inicializuje při zapnutí kostky, kdy se zároveň aktivuje výstup pro podržení napájení. Při vypnutí kostky se tento výstup deaktivuje, čímž dojde k přerušení napájení periferií a mikrokontroleru. V paměti expanderu se uchovává poslední zapsaný stav výstupů a aktuální stav vstupů. Komunikace s expanderem je řízena objekty zmíněných vstupních a výstupních periferií bez nutnosti uživatelského zásahu.

3.2.7 AD převodník

Převodník se inicializuje při inicializaci prvního analogového senzoru (NXT světelný, zvukový, dotykový senzor). Je nastavena výchozí konfigurace převodníku: rychlost vzorkování 1 kSa/s a externí napěťová reference (+5 V). Měření napětí probíhá vždy mezi jedním ze čtyř kanálů převodníku (jednotlivé kanály jsou připojeny na senzorové porty) a zemí. Při konverzi napětí je nejdříve změněn kanál pro měření, následně je

převodníku zaslán příkaz pro start konverze a nakonec je přečtena digitální hodnota. Tento proces je řízen objekty analogových senzorů.

3.2.8 ESP32

Modul ESP32 umožňuje bezdrátovou komunikaci kostky s externím zařízením pomocí Wi-Fi nebo Bluetooth. Aktuální řešení používá Bluetooth sériový přenos dat s jednoduchým firmwarem ESP, který je popsán v této sekci. Zkušenější uživatel může do modulu nahrát vlastní firmware a komunikaci si upravit podle potřeb.

Data z externího zařízení jsou přijata ESP a přeposlána mikrokontroleru pomocí UART sběrnice. Obdobně jsou data z mikrokontroleru posílána ESP a z něj následně přeposílána připojenému zařízení. ESP přijatá data nijak neupravuje a přeposílá je okamžitě po přijetí. V mikrokontroleru se přijatá data ukládají do bufferu sběrnice. MicroPython pro Raspberry Pi neumožňuje použití žádosti o přerušení po přijetí dat sběrnicí UART [12], proto jsou data čtena a zpracovávána každých 10 ms použitím časovače.

Data se posílají ve zprávách, což jsou celky znaků nebo čísel patřící k sobě. Data lze posílat v ASCII nebo v binárním režimu.

V ASCII režimu se data posílají ve formátu daném standardem ASCII. Zpráva je ukončena přijetím znaku terminátoru LF nebo CR. Zpráva může mít libovolný počet znaků (omezený pouze pamětí mikrokontroleru a externího zařízení).

V binárním režimu je uživatelem specifikována hlavička zprávy, obsahující nejméně 1 znak o velikosti 8 bitů. Tato hlavička je přidána na začátek odesílané zprávy a je použita pro identifikaci začátku zprávy při příjmu dat. Uživatel dále specifikuje velikost přijímané zprávy v bytech, podle které je určen její konec. Odesílané zprávy mohou mít různou velikost (externí zařízení ale musí zprávu správně interpretovat), přijímané zprávy mají vždy stejnou velikost.

3.3 NXT moduly

3.3.1 Světelný senzor

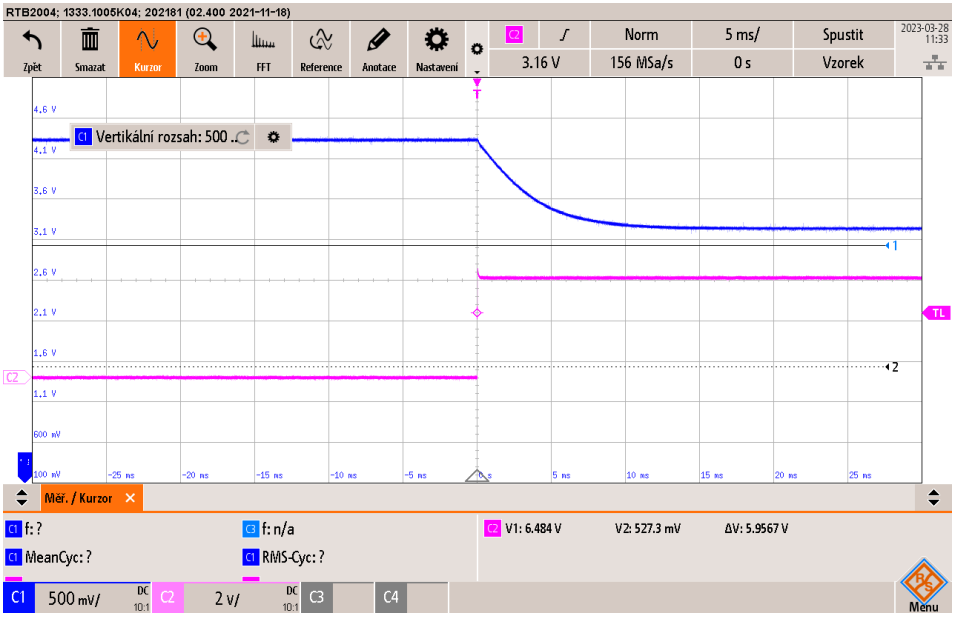
Světelný senzor (obrázek 3.1) je analogový senzor, který měří intenzitu dopadající světla na fototranzistor. Senzor lze připojit na senzorové porty 1–4.

Piny 2 a 3 jsou v senzoru navzájem propojené a v kostce jsou připojeny k zemi, na pin 4 je připojeno napájení +5 V. Pin 5 je připojen k mikrokontroleru a jeho aktivací dojde v senzoru k otevření tranzistoru a rozsvícení vestavěné červené LED. Výstupem senzoru je analogová hodnota napětí na rezistoru a tranzistoru, jehož otevření je určeno otevřením fototranzistoru. Tato hodnota je měřena vestavěným AD převodníkem (sekce 3.2.7) mezi piny 1 a 3. Měřitelný rozsah intenzity světla je v rozmezí daném AD převodníkem 0–32768, kde 0 značí nejvyšší intenzitu světla a 32768 nejmenší (kvůli otevřenému kolektoru tranzistoru a pull-up rezistoru uvnitř senzoru). [1, 11]

Obrázek 3.1: NXT světelný senzor.

Vliv okolních podmínek, jako je např. venkovní světlo a blikání zářivek v místnosti dokážeme eliminovat pomocí modulace světla, kdy změříme intenzitu dopadajícího světla při zapnuté a při vypnuté LED a následně tyto dvě hodnoty odečteme. Tímto procesem získáme přesnější měření očištěné od rušivých elementů.

Měření výstupní analogové hodnoty je ovlivněno rychlostí ustálení napětí na měřeném rezistoru světelného senzoru a vzorkovací rychlostí AD převodníku. Jak můžeme vidět na obrázku 3.2, doba, za kterou se napětí na rezistoru ustálí je přibližně 10 ms. vzorkovací frekvence AD převodníku je 1 kHz. Při zapínání a vypínání LED tedy musíme čekat dobu odpovídající těmto hodnotám, abychom dostali směrodatnou informaci o intenzitě světla.



Obrázek 3.2: Přechodný děj napětí na měřícím rezistoru světelného senzoru při zapnutí LED. Růžově je znázorněno napětí na LED při přechodu z vypnutého do zapnutého stavu. Modře je znázorněno napětí na rezistoru senzoru.

Světelný senzor lze ovládat ve dvou módech – v manuálním a v kontinuálním.

V manuálním módu uživatel zajišťuje ovládání senzoru – nastavuje stav LED a pro zjištění aktuální výstupní hodnoty zažádá o blokující konverzi AD převodníkem.

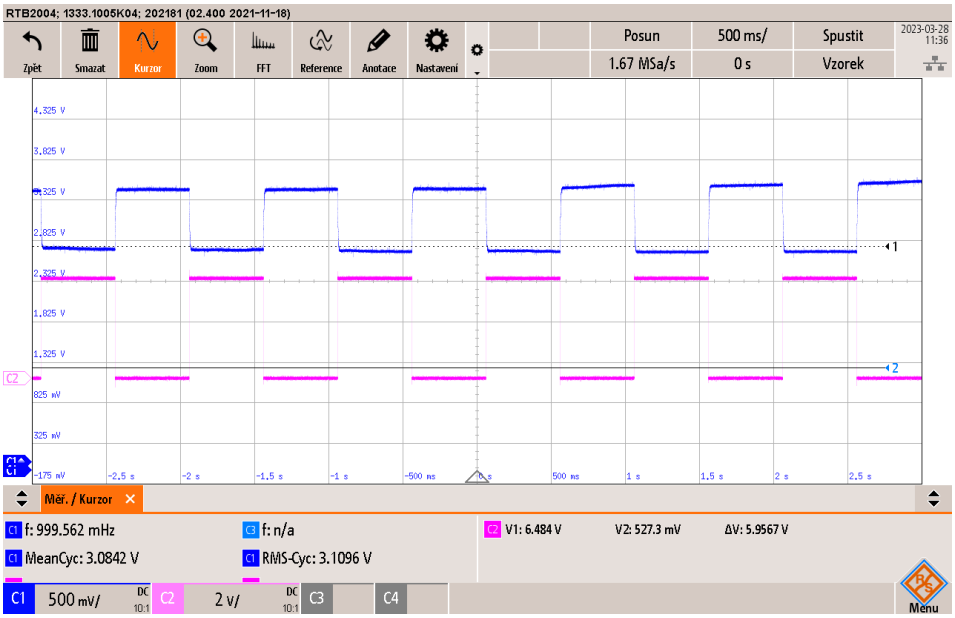
V kontinuálním módu je napětí konvertováno periodicky a výstupní hodnota je uložena do bufferu. Kontinuální mód může dále pracovat v režimech blikající a neblikající. V blikajícím režimu je měřena hodnota ve vypnutém a následně v zapnutém stavu LED. Změřené hodnoty jsou ukládány do bufferu a uživateli je k dispozici vždy poslední změřená hodnota pro vypnutou a pro zapnutou LED. Tyto hodnoty lze použít pro výše popsané měření při modulaci světla. V neblikajícím režimu probíhá měření při uživatelem nastaveném stavu LED.

Doba přechodu fototranzistoru NXT světelného senzoru je poměrně dlouhá, proto je v kontinuálním módu v blikajícím režimu možné nastavit dobu čekání mezi změnou stavu LED a začátkem konverze analogové hodnoty. Výchozí doporučená doba čekání je 10 ms.

Pin přerušení AD převodníku není k mikrokontroleru připojen a není žádoucí mikrokontroler a I2C sběrnici zatěžovat dotazováním se, zda jsou nová data z převodníku k dispozici. Proto je výchozí doporučená doba periody čtení zvolena na 1,5 ms, abychom zajistili čtení aktuální převedené hodnoty z AD převodníku.

Jak lze vidět na obrázku 3.3, napětí na rezistoru je po ustálení stabilní a při dodržení doporučené doby čekání můžeme bezpečně číst výstupní hodnotu senzoru.

Ke kostce je možné připojit až 4 světelné senzory, u každého z nichž lze nastavit manuální, či kontinuální mód a blikající, či neblikající režim.



Obrázek 3.3: Napětí na měřícím rezistoru světelného senzoru při blikání LED. Růžově je znázorněno napětí na LED. Modře je znázorněno napětí na rezistoru senzoru.

3.3.2 Dotykový senzor

Dotykový senzor je analogový senzor, který lze připojit na senzorové porty 1–4. Senzor má skokově měnící se výstup a měřená analogová hodnota AD převodníkem (sekce 3.2.7) tedy nabývá prakticky dvou úrovní – v rozepnutém stavu se blíží 0 a v sepnutém stavu 32768. Při rozhodování, zda-li je senzor sepnutý, je analogová hodnota porovnávána s úrovní mezi těmito dvěma stavy, tedy s úrovní 16384. Výstup senzoru je binární.

Obrázek 3.4: NXT dotykový senzor.

3.3.3 Ultrazvukový senzor

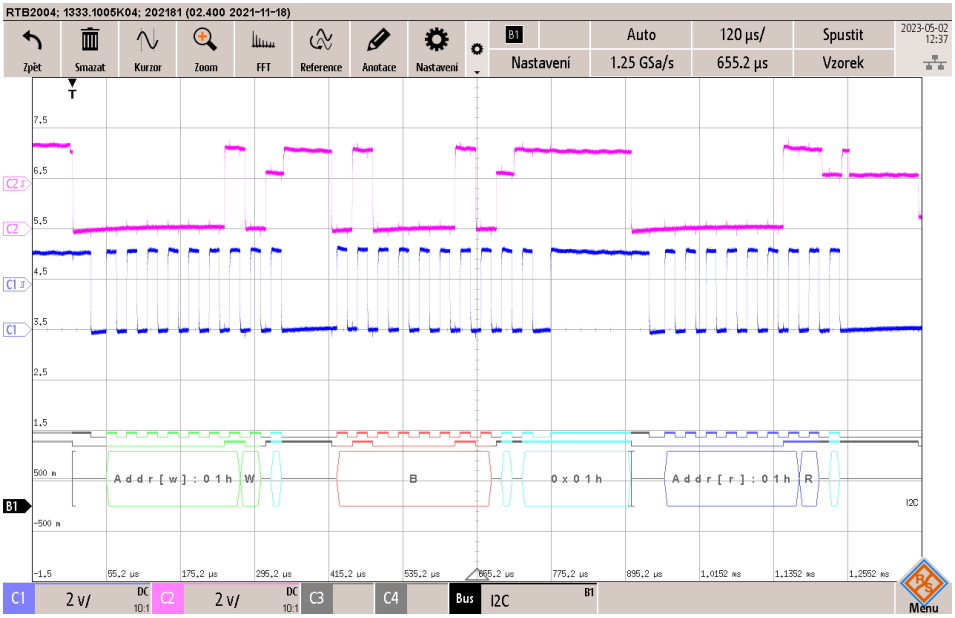
Ultrazvukový senzor je digitální senzor, který lze ke kostce připojit přes dedikovaný port, který umožňuje komunikaci s mikrokontrolerem po sběrnici I2C 2. Senzor vrací digitální hodnotu v rozmezí 0–255 cm, s přesností ±3 cm. Senzor funguje na principu měření doby letu zvukového pulzu a umožňuje měření vzdálenosti jednoho až osmi nejbližších objektů. [11]

Obrázek 3.5: NXT ultrazvukový senzor.

Senzor může pracovat v periodickém režimu, kdy jsou data čtena s nastavitelnou frekvencí, nebo v neperiodickém režimu, kdy jsou data čtena na žádost uživatele. V periodickém režimu jsou data ukládaná do bufferu, který je uživateli přístupný ke čtení.

Dále senzor může pracovat v blokujícím či neblokujícím režimu. V blokujícím režimu je komunikace po I2C sběrnici zprostředkovaná procesorem mikrokontroleru. V případě neblokujícího režimu je komunikace po I2C sběrnici zprostředkovaná stavovými automaty mikrokontroleru. Kvůli nízké komunikační rychlosti 9600 baudů je vhodnější senzor provozovat v neblokujícím režimu.

Ukázka komunikace mikrokontroleru s ultrazvukovým senzorem po I2C sběrnici je znázorněna na obrázku 3.6. Senzor nepoužívá standardizovanou I2C komunikaci, proto je nutné provést úpravu hodinového signálu: nejdříve je do senzoru (adresa 0x02 – v I2C standardu je tato adresa rezervována [2]) zapsán požadavek na čtení změřených dat (byte 0x42), poté je přidán hodinový pulz a následuje žádost o čtení dat. Data zaslaná senzorem na obrázku nejsou zobrazena z důvodu lepší čitelnosti obrázku, protože zde již probíhá komunikace standardně. Dekódování dat osciloskopem neodpovídá skutečnosti, adresa je při I2C komunikaci bitově posunuta doprava, dekódovaný znak B odpovídá bytu 0x42 a kvůli přidanému hodinovému je za znakem B znovu dekódovaná adresa.



Obrázek 3.6: I2C žádost ultrazvukovému senzoru o změřenou vzdálenost.

3.3.4 Zvukový senzor

Zvukový senzor (obrázek 3.7) je analogový senzor, který měří hladinu okolního zvuku. Senzor lze připojit na senzorové porty 1–4.

Piny 2 a 3 jsou v senzoru navzájem propojené a v kostce jsou připojeny k zemi, na pin 4 je připojeno napájení 5 V. Výstupem senzoru je analogová hodnota napětí na rezistoru a tranzistoru, jehož otevření je určeno operačním zesilovačem, kterému předchází obvod pro měření zvuku. Tato hodnota je měřena vestavěným AD převodníkem (sekce 3.2.7) mezi piny 1 a 3. Měřitelný rozsah hladiny okolního zvuku je v rozmezí daném AD převodníkem 0–32768, kde 0 značí nejvyšší intenzitu světla a 32768 nejmenší. [1, 11]

Obrázek 3.7: NXT zvukový senzor.

3.3.5 Servomotor

NXT a EV3 motory se liší pouze vzhledem a vnější konstrukcí, vnitřní parametry a ovládání je stejné. Motor (obrázek 3.8) je ovládán pomocí PWM signálu. Motor má vestavěný rotační enkodér s rozlišením 360 pulzů na otáčku. [9, 11] Enkodéry umožňují získat polohu motoru ve stupních a úhlovou rychlost ve stupních za sekundu. Implementace obsluhy enkodérů s aproximací úhlové rychlosti je prací Jakuba Vaňka a je v této práci použita pro řízení motorů. Ovládání motoru je možné ve 3 módech – řízení výkonu, polohy nebo rychlosti.

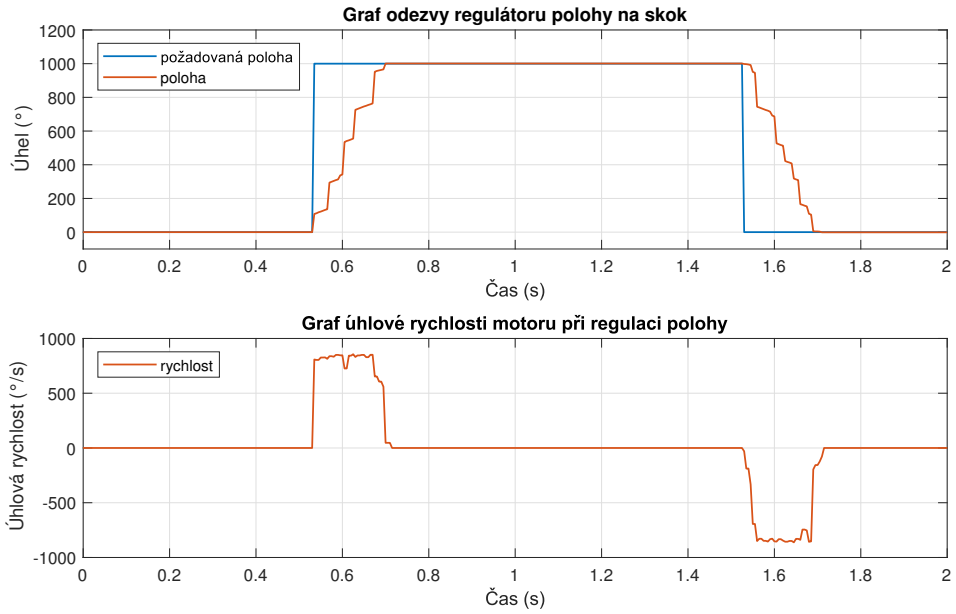
Obrázek 3.8: NXT servomotor.

Požadovaná frekvence PWM pro ovládání motorů je 20 kHz. Výstupy PWM jsou nastaveny v režimu phase correct a hodnota wrap je nastavena na 3124, což umožňuje nastavit výkon na motorech při kladné a záporné polaritě hodnoty výkonu v celkem 6250 různých úrovních. Mikrokontroler a PWM pracují na základní frekvenci 133 MHz, zmíněným nastavením výstupů je dosaženo požadované výstupní frekvence. [14]

V módu řízení polohy lze přímo nastavit výkon na motorech v rozmezí -100–100 %. Nastavená střída PWM odpovídá absolutní hodnotě výkonu a nastavení H můstku

(a tedy směru otáčení motorů) odpovídá polaritě hodnoty výkonu. Kladný směr otáčení motoru je při pohledu na motor jako na obrázku 3.8 po směru hodinových ručiček.

V módu řízení polohy lze nastavit požadované natočení motoru ve stupních. Řízení je zprostředkováno PID regulátorem. Ve výchozím nastavení jsou konstanty regulátoru již nastaveny a experimentálně ověřeny při volnoběhu a s mírnou zátěží, pro specifické použití motorů může uživatel konstanty změnit. Odezva regulátoru na skok je zobrazena na obrázku 3.9.

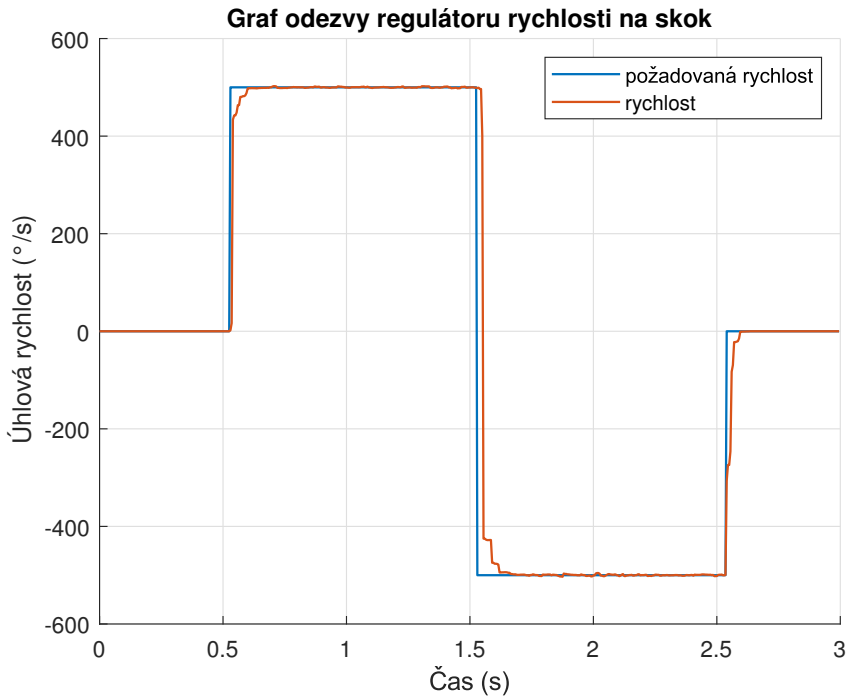


Obrázek 3.9: Odezva regulátoru polohy motoru na skok.

V módu řízení rychlosti lze nastavit požadovanou rychlost otáčení motoru ve stupních za sekundu. Aproximovaná rychlost získaná enkodéry v obsahuje vysokofrekvenční rušení, a proto je filtrována dolnopropustným filtrem s mezní frekvencí 25 Hz. Rovnice pro výpočet n-té hodnoty filtrované rychlosti v^{f} je:

$$v_n^{\mathrm{f}} = 0.854 \cdot v_{n-1}^{\mathrm{f}} + 0.0728 \cdot v_n + 0.0728 \cdot v_{n-1}. \tag{3.1}$$

Řízení je zprostředkováno PID regulátorem. Ve výchozím nastavení jsou konstanty regulátoru nastaveny a experimentálně ověřeny při volnoběhu a s mírnou zátěží, pro specifické použití motorů může uživatel konstanty změnit. Odezva regulátoru na skok je zobrazena na obrázku 3.10.



Obrázek 3.10: Odezva regulátoru rychlosti motoru na skok.

Kapitola 4

Demonstrační úlohy

4.1 Robot sledující čáru

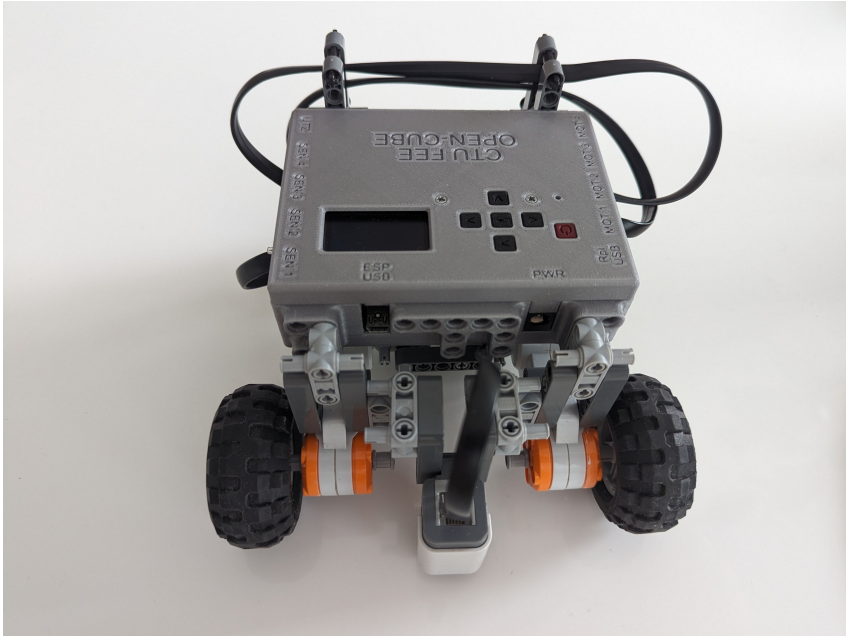
Robot sleduje černou čáru na bílém podkladu.

Konstrukce robotu (obrázek 4.1) se skládá ze dvou motorů se dvěma středně velkými koly vpředu, ze světelného senzoru, který je umístěný mezi těmito koly a z jednoho volně se pohybujícího malého kola vzadu.

Ke sledování čáry je použit regulátor výkonu motorů, jehož vstupní veličinou je normalizovaná intenzita světla ze světelného senzoru.

Při běhu programu je možná oboustranná komunikace s externím počítačem se spuštěným Simulinkovým modelem. Takto lze vizualizovat časové průběhy jednotlivých stavů robotu a vzdáleně nastavovat konstanty regulátoru.

Konstanty regulátorů byly určeny iterativně. S použitím externí komunikace je proces ladění regulátorů relativně rychlý a jednoduchý, protože konstanty regulátorů lze měnit bez nutnosti vypínání programu a nahrávání nového kódu do robotu.



Obrázek 4.1: Robot sledující čáru.

4.1.1 Intenzita světla

Robot se snaží udržet na přechodu mezi černou čarou a bílým podkladem. Změřená intenzita světla určuje v jaké míře se robot nachází nad čarou nebo podkladem.

V počáteční pozici je robot umístěn světelným senzorem nad čarou. Po spuštění programu robot stokrát změří intenzitu světla nad čarou, otočí levým kolem tak, aby se nacházel nad podkladem a provede stejný proces měření. Z hodnot je určena a zaznamenána střední hodnota intenzity světla při výskytu robotu nad čarou ($\bar{I}_\mathrm{c}$) a pokladem ($\bar{I}_\mathrm{p}$). Robot následně otočí levým kolem tak, aby se vyskytoval senzorem na rozhraní čáry a podkladu a rozjede se. Při jízdě je periodicky měřena intenzita světla I_z, která je za použití zaznamenaných středních hodnot normalizována následovně:

$$\tilde{I} = 100 \cdot \min\left\{1, \max\left\{-1, \frac{I_\mathrm{z} - \frac{\bar{I}_\mathrm{c}+\bar{I}_\mathrm{p}}{2}}{\frac{|\bar{I}_\mathrm{c}-\bar{I}_\mathrm{p}|}{2}}\right\}\right\}, \tag{4.1}$$

kde $\tilde{I}$ je normalizovaná intenzita světla v intervalu $(-100, 100)$.

Robustnější řešení měření intenzity při světelné modulaci, popsané v sekci 3.3.1, nebylo možné použít. Důvodem je nedostatečný počet změřených hodnot pro řízení motorů v důsledku dlouhé doby ustálení napětí na senzoru při změně stavu LED.

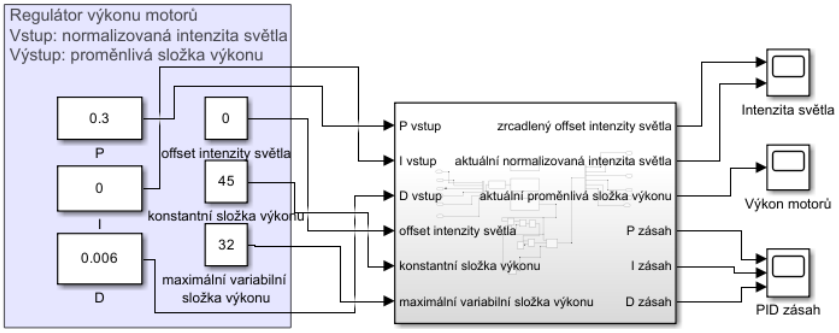
4.1.2 Řízení motorů

Při jízdě je výkon motorů určen součtem konstantní a variabilní složky . Konstantní složka zaručuje pohyb robotu směrem dopředu, variabilní určuje zatáčení směrem k čáře nebo k podkladu. Variabilní složka výkonu je určena PD regulátorem, jehož vstupní veličina je normalizovaná intenzita světla. Pro levý motor je v součtu výkonů použita hodnota variabilní složky a pro pravý motor je použita opačná hodnota variabilní složky.

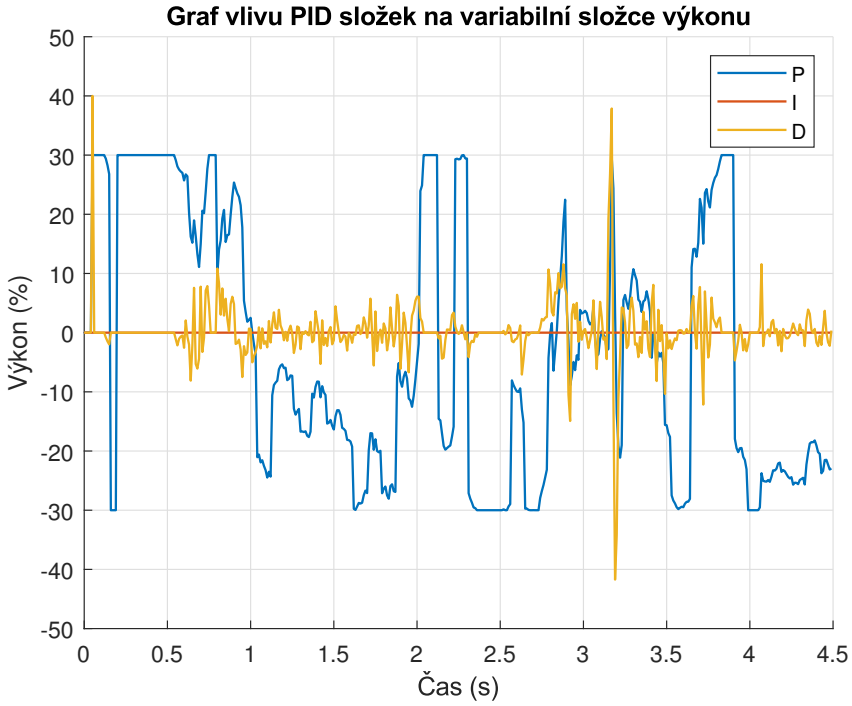
4.1.3 Komunikace s počítačem

Komunikace robotu s počítačem se spuštěným Simulinkovým modelem probíhá sériovým Bluetooth přenosem. Data jsou přenášena ve formátu single o velikosti 4 byty. Data o aktuálním stavu robotu jsou odesílaná každou periodu regulace do počítače, kde jsou v modelu zobrazena v reálném čase na grafech. Z počítače lze změnou hodnot bloků konstant v modelu nastavovat koeficienty regulátoru, tato data se odesílají s periodou $0,2\,\mathrm{s}$. Model je zobrazen na obrázku 4.2. V levé části lze nastavovat konstanty regulátoru, uprostřed je blok zajišťující komunikaci a vpravo lze zobrazit grafy.

Ukázka vizualizace vlivu PID složek regulátoru na variabilní složku výkonu při sledování čáry je na obrázku 4.3. V úloze je použit PD regulátor, proto je vliv integrační složky nulový. V modelu lze nastavit všechny tři konstanty regulátoru (P, I, D) a ke sledování čáry tak použít regulátor jiného typu.



Obrázek 4.2: Simulinkový model robotu sledujícího čáru.



Obrázek 4.3: Graf vlivu PID složek regulátoru na variabilní složku výkonu při sledování čáry.

4.2 Balancující robot

Robot dokáže balancovat na místě, udržovat stálou polohu při vychýlení a na příkaz se pohybovat translačně a rotačně. Doplňující funkcionalitou je udržování vzdálenosti robotu od překážky.

Konstrukce robotu (obrázek 4.4) se skládá ze dvou motorů s velkými koly, na kterých robot balancuje, a z ultrazvukového senzoru, který slouží k určení vzdálenosti překážky. Robot využívá vestavěný gyroskop a akcelerometr pro aproximaci úhlu náklonu.

Ke stabilizaci a pohybu robotu je použita sadu regulátorů, jejichž výsledkem je nastavení výkonu motorů.

Při běhu programu je možná oboustranná komunikace s externím počítačem se spuštěným Simulinkovým modelem. Takto lze vizualizovat časové průběhy jednotlivých stavů robotu, posílat příkazy k pohybu a vzdáleně nastavovat konstanty regulátorů.

Konstanty regulátorů byly určeny iterativně. S použitím externí komunikace je proces ladění regulátorů relativně rychlý a jednoduchý. Lze sledovat odezvy regulátorů na skok a měnit jejich konstanty bez nutnosti vypínání programu a nahrávání nového kódu do robotu.

Obrázek 4.4: Balancující robot.

4.2.1 Úhel náklonu

K aproximaci úhlu náklonu je použita fúze dat z gyroskopu a akcelerometru. Z gyroskopu získáváme úhlové zrychlení kolem osy náklonu (y). Integrací úhlové rychlosti

získáme požadovaný úhel náklonu. Data z gyroskopu jsou málo zarušená, ale integrovaný úhel je zatížen driftem v důsledku offsetu senzoru. Z akcelerometru získáváme zrychlení ve dvou osách (x, y) kolmých na osu náklonu, ze kterých trigonometricky určíme úhel náklonu:

$$\theta^{\mathrm{acc}} = \arctan \frac{a_{\mathrm{z}}}{a_{\mathrm{y}}}, \tag{4.2}$$

kde θ^{acc} je hledaný úhel a a_{y}, a_{z} jsou zrychlení v ose y a z, získaná z akcelerometru. Tento úhel není zatížen driftem, ale může být určen nepřesně při translačním pohybu senzoru a je zatížen vysokofrekvenčním rušením. [17]

Z výše uvedených důvodů používáme k fúzi dat komplementární filtr s dolnopropustným filtrem aplikovaným na data z gyroskopu a hornopropustným filtrem aplikovaným na data z akcelerometru. Komplementární filtr je výpočetně velmi rychlý a zajišťuje dostatečně přesné aproximování úhlu náklonu pro tuto úlohu. [17]

Filtr popíšeme rovnicí

$$\theta_{\mathrm{n}} = \alpha \cdot (\theta_{\mathrm{n}-1} + g_{\mathrm{n}} \cdot \Delta t) + (1 - \alpha) \cdot \theta_{\mathrm{n}}^{\mathrm{acc}}, \tag{4.3}$$

kde θ_{n} je aktuální úhel náklonu, $\theta_{\mathrm{n}-1}$ je předchozí úhel náklonu, α je koeficient filtru, určující mezní frekvenci dolnopropustného a hornopropustného filtru, g je úhlová rychlost otáčení získaná z gyroskopu, Δt je čas mezi vzorky senzorů $\mathrm{n}-1$ a n, $\theta_{\mathrm{n}}^{\mathrm{acc}}$ je aktuální úhel náklonu z rovnice (4.2).

V grafu 4.5 jsou graficky znázorněny výše zmíněné problémy určení úhlu náklonu. V první sekundě průběhu je senzor téměř stabilní. V druhé sekundě se senzor naklání velmi rychle a je zde zřetelná nestabilita úhlu z akcelerometru oproti úhlu z gyroskopu. V čase 2–4 s se senzor naklání pomalu. V poslední sekundě je senzor opět stabilní, zde můžeme vidět vliv driftu na úhlu gyroskopu. Na celém průběhu lze vidět vysokofrekvenční rušení úhlu akcelerometru a postupný drift úhlu gyroskopu v porovnání s úhlem získaným fúzí obou senzorů pomocí komplementárního filtru.

4.2.2 Ujetá vzdálenost a rychlost

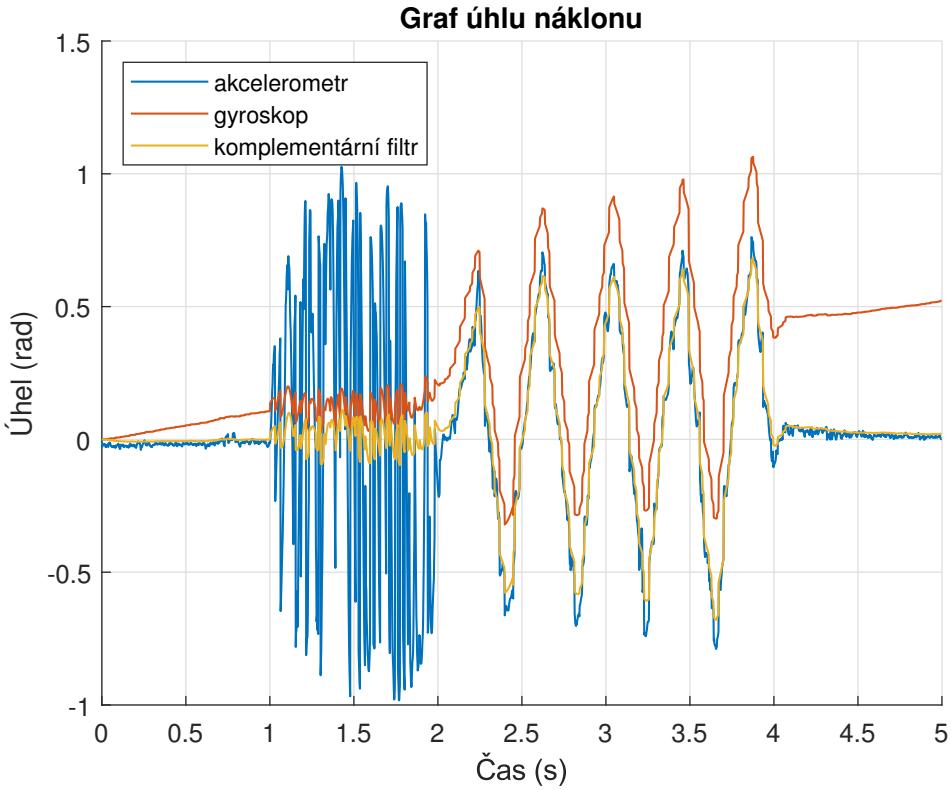
Ujetou vzdálenost a rychlost robotu určujeme sledováním enkodérů obou motorů. To umožňuje plynulejší stabilizaci robotu, udržování pozice a vykonávání příkazů pro translační a rotační pohyb.

4.2.3 Řízení motorů

Pro řízení motorů používáme sadu 5 regulátorů. Nejdůležitější z nich jsou 3 kaskádně propojené regulátory polohy, rychlosti a náklonu.

Regulátor polohy je PI regulátor, který umožňuje translační pohyb, zaručuje udržování pozice robotu a určuje požadovanou rychlost pohybu. Regulátor rychlosti je PD regulátor s filtrovanou derivační složkou, který vyhlazuje pohyb robotu a určuje požadovaný náklon. Regulátor náklonu je PD regulátor s filtrovanou derivační složkou, který zajišťuje požadovaný náklon robotu a určuje balancující složku výkonu motorů.

Čtvrtý regulátor je PI regulátor, který udržuje natočení robotu a jeho rotační pohyb. Regulátor určuje rotační složku výkon motorů.



Obrázek 4.5: Graf porovnání úhlu z akcelerometru, gyroskopu a fúzí dat obou senzorů.

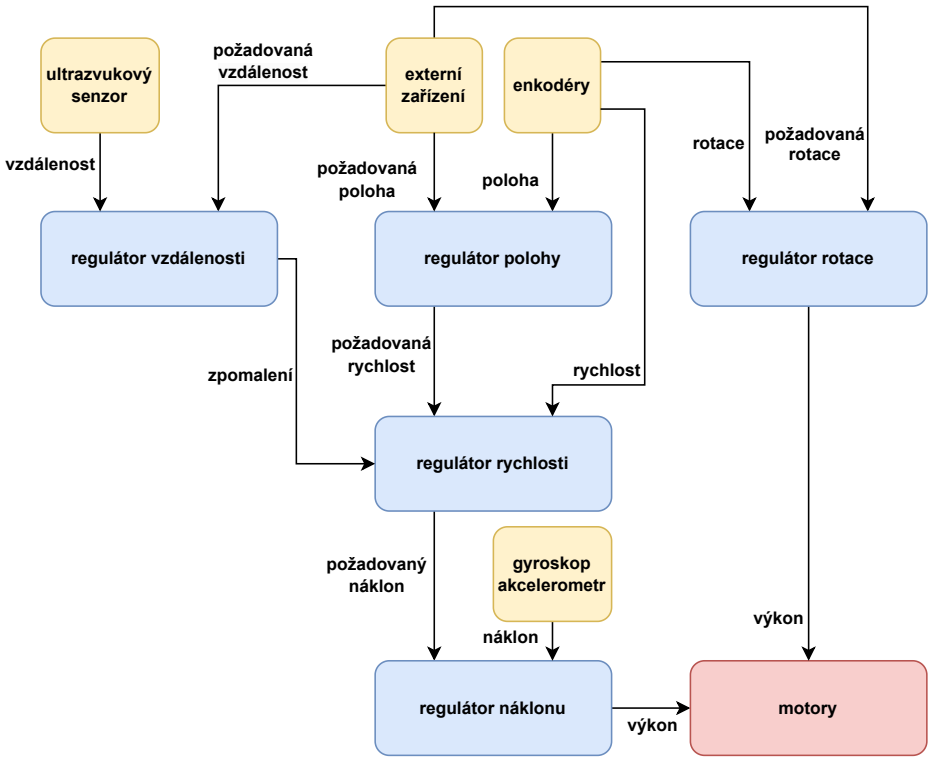
Pátý regulátor zajišťuje zpomalení robotu před překážkou. Určuje zpomalení robotu, které vstupem regulátoru rychlosti.

Perioda regulace je 0,01 s, perioda čtení dat z ultrazvukového senzoru je 0,1 s a enkodéry motorů jsou čteny při žádosti o přerušení. Schéma propojení senzorů, motorů a regulátorů je na obrázku 4.6.

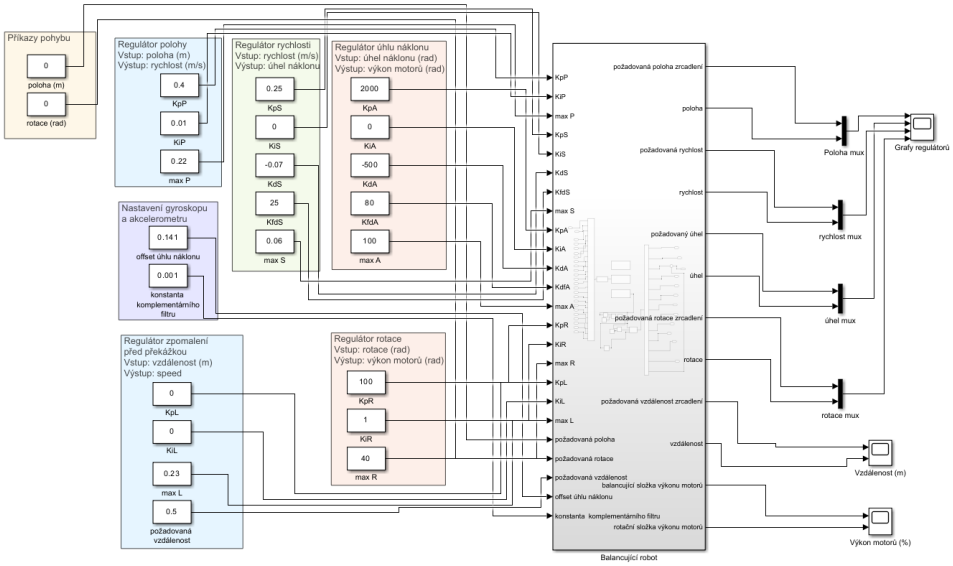
4.2.4 Komunikace s počítačem

Komunikace robotu s počítačem se spuštěným Simulinkovým modelem probíhá sériovým Bluetooth přenosem. Data jsou přenášena ve formátu single o velikosti 4 byty. Data o aktuálním stavu robotu jsou do počítače odesílaná každou periodu regulace. Z počítače se příkazy k pohybu a koeficienty regulátorů odesílají s periodou 0,2 s. Model je zobrazen na obrázku 4.7. V levé části lze nastavovat konstanty regulátorů, gyroskopu a příkazy k pohybu, vpravo lze zobrazit grafy, uprostřed je blok zajišťující komunikaci.

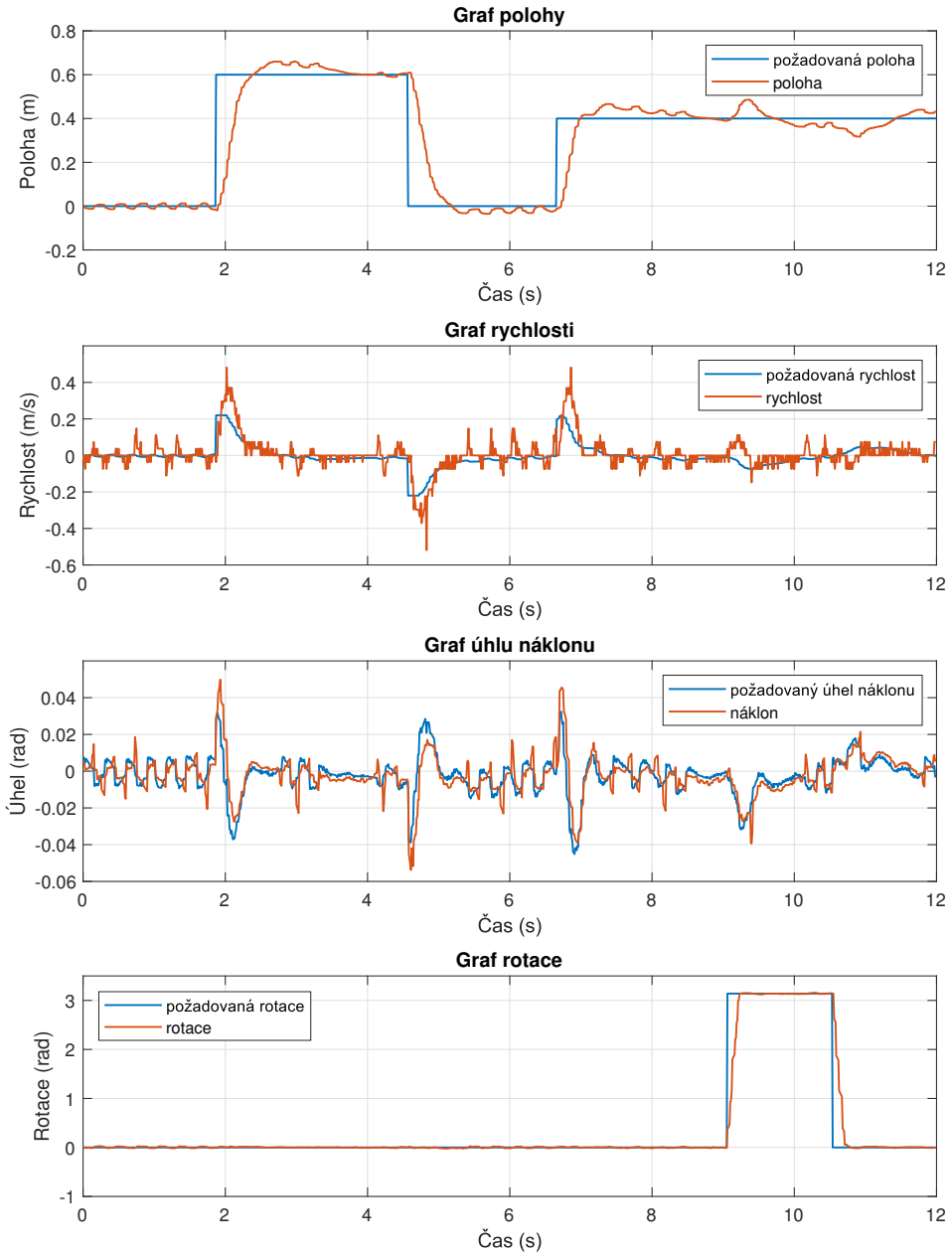
Ukázka vizualizace stavů robotu je na obrázku 4.8. V prvním grafu je aktuální a požadovaná pozice robotu, v druhém je aktuální a požadovaná rychlost, ve třetím je aktuální a požadovaný náklon a ve čtvrtém je aktuální a požadovaná rotace robotu. Startovní pozice robotu byla v poloze 0 m a rotaci 0 rad. V grafu polohy lze vidět požadavky na změnu polohy a odezvu robotu v prvních 8 sekundách průběhu. V grafu rotace lze na konci průběhu vidět požadavek na rotaci o 3,14 rad a poté zpět do startovní pozice rotace.



Obrázek 4.6: Schéma propojení regulátorů balancujícího robotu.



Obrázek 4.7: Simulinkový model balancujícího robotu.



Obrázek 4.8: Vizualizace stavů balancujícího robotu.

Kapitola 5

Další práce

V této fázi projektu je vytvořený základní firmware pro ovládání komponent kostky a NXT senzorů a motorů. Další možná vylepšení a rozšíření projektu jsou

1. Propojení aktuálního firmwaru s knihovnami pro ovládání EV3 senzorů.
2. Vývoj nových senzorů.
3. Vývoj nových motorů.
4. Využití ESP32 pro Wi-Fi komunikaci s mobilní aplikací nebo vytvoření webového rozhraní.
5. Využití ESP32 ke komunikaci a koordinaci s dalšími kostkami.

Kapitola 6

Závěr

V rámci této práce byl implementován a otestován firmware alternativní řídicí LEGO kostky Open-Cube, jehož součástí je framework, knihovny umožňující ovládání komponent kostky a knihovny umožňující ovládání LEGO NXT senzorů a motorů. Při testování hardwaru a předešlých implementovaných knihoven, napsaných v jazyce MicroPython, bylo zjištěno, že běh programů je příliš pomalý, a proto byla většina předešlých a nových knihoven implementována v jazyce C. Firmware Open-Cube tvoří jak knihovny v jazyce MicroPython, tak knihovny v jazyce C zkompilované se standardním kódem MicroPythonu.

Implementovaný framework kostky zaručuje správné nastavení, inicializaci a deinicializaci komponent kostky, senzorů a motorů, umožňuje jednoduché používání a přístup k těmto periferiím. Součástí frameworku je také hlavní program, spouštěný při zapnutí kostky, který provádí prvotní nastavení komponent kostky s ochranou před podvybitím baterií, zobrazuje menu na kostce s možností výběru a spuštění uživatelského programu a zaručuje správné ukončení spuštěného programu.

Další součástí firmwaru jsou knihovny umožňující ovládání a komunikaci mikrokontroleru s komponentami kostky (tlačítko, piezoelektrický reproduktor, LED, displej, gyroskop a akcelerometr, GPIO expander, AD převodník, ESP32). Dále knihovny umožňující ovládání čtyř NXT senzorů (světelný, dotykový, ultrazvukový, zvukový) s nastavitelnými módy měření výstupních hodnot a servomotoru s módy řízení výkonu, polohy a rychlosti, používající PID regulátory s přednastavenými konstantami.

Funkčnost řešení je demonstrována na dvou úlohách. První z nich je klasická úloha robotu sledujícího čáru. Druhá úloha je robot balancující na dvou kolech (segway). V obou úlohách robot bezdrátově komunikuje s počítačem, odkud lze nastavovat konstanty regulátorů a zobrazovat stavy robotu v reálném čase. Díky tomu lze upravovat chování robotu a regulátorů bez nutnosti nahrávání nového kódu do kostky.

Nakonec byla sepsána dokumentace/manuál ke kostce s návody na instalaci firmwaru, nahrávání souborů a programů do kostky, používání implementovaných funkcionalit s ukázkami kódu a propojení kostky s externím zařízením přes Bluetooth.

Literatura

[1] *ADS1119 4-Channel, 1-kSPS, 16-Bit, Delta-Sigma ADC With I2C Interface datasheet (Rev. A)*, https://www.ti.com/lit/ds/symlink/ads1119.pdf, (cit: 6. 4. 2023).

[2] *I2C-bus specification and user manual*, https://www.nxp.com/docs/en/user-guide/UM10204.pdf, (cit: 19. 5. 2023).

[3] *ICM-20608-G Datasheet*, https://invensense.tdk.com/wp-content/uploads/2015/03/DS-000081-v1.01.pdf, (cit: 6. 4. 2023).

[4] *ICM-20608-G Register Maps and Description*, https://invensense.tdk.com/wp-content/uploads/2015/03/RM-000030-v1.0.pdf, (cit: 6. 4. 2023).

[5] *LEDÍLY*, https://www.ledily.cz/lego-9841-1-nxt-intelligent-brick, (cit: 22. 5. 2023).

[6] *LEGO*, https://www.lego.com/, (cit: 22. 5. 2023).

[7] *LEGO Education SPIKE Prime Technical Specifications*, https://docs.rs-online.com/ba8f/A700000007765037.pdf, (cit: 22. 4. 2023).

[8] *LEGO Mindstorms EV3*, https://commons.wikimedia.org/wiki/File:Lego-mindstorms-ev3.jpg, (cit: 22. 5. 2023).

[9] *LEGO MINDSTORMS EV3 Hardware Developer Kit*, https://www.mikrocontroller.net/attachment/338591/hardware_developer_kit.pdf, (cit: 22. 4. 2023).

[10] *LEGO Mindstorms NXT*, https://commons.wikimedia.org/wiki/File:Lego_mindstorms_nxt_main_brick.jpg, (cit: 22. 5. 2023).

[11] *LEGO MINDSTORMS NXT Hardware Developer Kit*, https://www.csd.uoc.gr/~hy428/reading/lego_nxt_hw_dev_kit.pdf, (cit: 22. 4. 2023).

[12] *MicroPython documentation*, https://docs.micropython.org/, (cit: 6. 4. 2023).

[13] *PCA8575 Remote 16-bit I/O expander for I2C-bus with interrupt*, https://www.ti.com/lit/ds/symlink/pcf8575.pdf, (cit: 6. 4. 2023).

[14] *RP2040 Datasheet*, https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf, (cit: 6. 4. 2023).

[15] *RPishop*, https://rpishop.cz/raspberry-pi-rp2040/3669-raspberry-pi-rp2040.html, (cit: 25. 5. 2023).

[16] *Růžovka*, https://ruzovka.cz/cs/2-stupen-zs-vxs/16977-lego-mindstorms-45500-ev3-ridici-jednotka.html, (cit: 22. 5. 2023).

[17] V. Hunter Adams, *Complementary filters*, https://vanhunteradams.com/Pico/ReactionWheel/Complementary_Filters.html, (cit: 17. 4. 2023).

Příloha A

Kód

Kód firmwaru kostky a ESP32, kódy testovacích a demonstračních programů a Simulinkové modely demonstračních programů jsou přiloženy na USB flash disku. Soubory `encoder.c`, `encoder.h`, `encoder_api.c`, `encoder_api.c`, `i2c_locking.c`, `i2c_locking.h` a `i2c_guard.py`, jsou součástí firmwaru, ale byly vytvořeny studentem Jakubem Vaňkem. Kompletní repozitář projektu Open-Cube je dostupný online na adrese `https://gitlab.fel.cvut.cz/open-cube/firmware`. Struktura adresáře s vybranými nejdůležitějšími soubory je následující:

- `kód`
  - `lib` ..... knihovny v jazyce MicroPython
    - `cube` ..... knihovny komponent kostky
    - `hw_defs` ..... definice hardwarových konstant
    - `nxt` ..... knihovny NXT senzorů
    - `robot.py` ..... implementace globálního objektu robot
    - `robot_consts.py` ..... pomocné uživatelské konstanty
  - `micropython` ..... soubory použité pro zkompilování firmwaru
    - `modules` ..... knihovny v jazyce C
      - `lib` ..... pomocné knihovny
      - `nxt_sensors` ..... knihovny NXT senzorů
      - `opencube_brick` ..... knihovny komponent kostky
      - `opencube_motors` ..... knihovny motorů
      - `opencube_pindefs` ..... definice hardwarových konstant
    - `typing` ..... popis knihoven v jazyce Python
  - `programs` ..... testovací a demonstrační programy
  - `simulink` ..... Simulinkové modely pro demonstrační programy
  - `main.py` ..... hlavní program spouštěný při zapnutí kostky
  - `esp_uart.ino` ..... firmware ESP32

Příloha B

Videonahrávky demonstračních úloh

Videonahrávky demonstračních úloh jsou přiloženy na USB flash disku. Nahrávky jsou dostupné také online na následujících adresách:

- robot sledující čáru: `https://youtu.be/EnySkf7ZCDQ`
- balancující robot: `https://youtu.be/b5wHINQK-8Y`

Příloha C

Dokumentace Open-Cube

Dokumentace a manuál k použití Open-Cube řídicí kostky.

Open-Cube

Katedra měření ČVUT FEL

2023

Obsah

1 Instalace firmwaru **4**

2 Editor kódu **4**
2.1 Nastavení editoru . . . 4
2.2 Nahrání kódu do kostky . . . 4

3 Robot **5**
3.1 Tlačítka . . . 7
3.2 LED . . . 8
3.3 Bzučák . . . 8
3.4 Baterie . . . 9
3.5 Displej . . . 9
3.6 Gyroskop a akcelerometr . . . 13
3.7 ESP32 komunikace . . . 14
3.7.1 Bluetooth propojení ESP32 s počítačem . . . 14
3.7.2 Kostka . . . 16

4 NXT senzory **21**
4.1 Tlačítko . . . 21
4.2 Světelný senzor . . . 21
4.3 Ultrazvukový senzor . . . 23
4.4 Zvukový senzor . . . 24

5 Servomotor **24**

6 Parametry a konstanty **27**
6.1 Sensor . . . 27
6.2 Port . . . 27
6.3 Button . . . 28
6.4 Light . . . 28
6.5 GyroAcc . . . 29

Seznam obrázků

1 Spárování počítače s kostkou 15
2 Zjištění čísla COM portu 15
3 Nastavení řešiče Simulinku 18
4 Umístění nastavení Simulation pacing v Simulinku 19
5 Nastavení Simulation Pacing 19
6 Nastavení bloku Serial Configuration v Simulinku 20
7 Příklad použití sériové komunikace v Simulinku 20

Seznam kódů

1 Použití globální třídy robot. 6
2 Zjištění stavu tlačítek. 7
3 Použití displeje. 12
4 Čtení dat z gyroskopu s využitím irq. 13
5 Komunikace mezi mikroprocesorem a ESP32. 16
6 Komunikace s kostkou z Matlabu. 17
7 Použití NXT tlačítka. 21
8 Použití NXT světelného senzoru. 23
9 Použití NXT ultrazvukového senzoru. 23
10 Použití NXT zvukového senzoru. 24
11 Použití motorů. 26

1 Instalace firmwaru

MicroPython firmware spolu se všemi Open-Cube knihovnami popsanými v tomto dokumentu je ve výchozím nastavení nahraný v kostce. Aktuální firmware lze také stáhnout z Open-Cube repozitáře [https://gitlab.fel.cvut.cz/open-cube/firmware/] ze složky micropython. Pro nahrání firmwaru při jeho aktualizaci, poškození či přepsání postupujte následujícími kroky:

1. Stáhněte si firmware [https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/micropython/] (uf2 binární soubor) pro kostku.
2. Připojte kostku k počítači pomocí USB kabelu.
3. Držte tlačítko boot select a stiskněte zapínací tlačítko.
4. V počítači otevřete adresář RPI-RP2 zkopírujte do něj firmware.
5. Po nahrání firmwaru se kostka restartuje a na displeji se zobrazí menu.

2 Editor kódu

Pro úpravu, debugování a nahrávání kódu do kostky doporučujeme použít editor Thonny [https://thonny.org/].

2.1 Nastavení editoru

Po prvním spuštění editoru klikněte na tlačítko v pravém dolním rohu a v záložce `Interpreter` nastavte interpretr na `MicroPython (Raspberry Pi Pico)`. Dále doporučujeme nastavit v záložce `General` možnost `UI mode` na `regular` nebo `expert`. V hlavním okně editoru v záložce `View` zvolte zobrazení lišt `Files` a `Shell`. Lišta `Files` slouží k procházení adresářů v počítači a `Shell` ke komunikaci s kostkou, vypisování informací a chybových hlášek.

2.2 Nahrání kódu do kostky

Po propojení zapnuté kostky USB kabelem k počítači se stisknutím tlačítka `Stop/Restart backend` v editoru připojíte ke kostce. V liště `Shell` se zobrazí informace o připojení a v liště `Files` se zobrazí soubory nahrané v kostce. Soubory i celé adresáře můžete kopírovat mezi kostkou a počítačem pravým kliknutím na daný soubor či adresář. Obdobně můžete mazat či vytvářet nové soubory a adresáře.

Upravovat soubory lze jak v adresáři počítače, tak v adresáři kostky. Pokud si otevřete soubor v kostce a po úpravách ho uložíte, automaticky se nahraje do kostky.

Uživatelské programy nahrávejte do adresáře `programs` v kostce. Programem může být jeden soubor s koncovkou `.py` nebo adresář, ve kterém je uživatelský soubor `main.py`. Takto nahrané programy se zobrazí v menu na kostce. Zobrazené jméno programu je určeno názvem souboru bez koncovky `.py` nebo názvem adresáře.

Po úpravě kódu v editoru můžete stisknutím tlačítka `Run current script` spustit na kostce kód aktuálně zobrazený v editoru. Doporučujeme takto spouštět pouze hlavní program `main.py`, který obsahuje framework kostky s inicializací všech potřebných funkcí

pro ovládání periferií. Po spuštění hlavního programu se na kostce zobrazí menu ovládané tlačítky na kostce s možností spuštění nahraného uživatelského programu. Soubor `main.py` se spustí automaticky při zapnutí kostky.

3 Robot

Po zapnutí kostky je hlavním programem `main.py` inicializovaný globální objekt `robot`, který obsahuje všechny funkce pro inicializaci, deinicializaci a přístup k objektům periferií kostky. Funkce jsou popsány v následujících kapitolách. Struktura proměnných objektu `robot`, kterými lze přistupovat k objektům periferií je následující:

```
robot
├── battery
├── buttons
├── buzzer
├── display
├── esp_uart
├── led
├── motors[4]
└── sensors
    ├── ultra
    ├── gyro
    ├── touch[4]
    ├── light[4]
    └── sound[4]
```

class **Robot**()

Inicializace, deinicializace a správa objektů motorů, senzorů a periferií kostky. Automaticky inicializuje tlačítka, LED, bzučák a měření napětí baterií s ochranou podvybití. Uživatelem lze inicializovat senzory, motory a ESP32 Bluetooth komunikaci.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot`

init_sensor(*sensor_type=None, port=None*)

Inicializace senzorů.

Parametry: **sensor_type** (Sensor) – Typ senzoru.
port (Port) – Port senzoru. Pro typ Sensor.GYRO a Sensor.ULTRA není potřeba specifikovat.

deinit_sensor(*sensor_type=None, port=None*)

Denicializace senzorů.

Parametry: **sensor_type** (Sensor) – Typ senzoru. Pokud není specifikován, deinicializuje senzor na daném portu.
port (Port) – Port senzoru. Pokud není specifikován, deinicializuje všechny senzory daného typu.

init_motor(*port=None*)

Inicializace motorů.

Parametry: **port** (Port) – Port motoru.

deinit_motor(*port=None*)

Denicializace motorů. Vypne regulátor, snímání enkodérů a zastaví motor.

Parametry: **port** (Port) – Port motoru.

init_esp_uart()

Inicializace ESP32 Bluetooth komunikace.

deinit_esp_uart()

Denicializace ESP32 Bluetooth komunikace.

Následující kód ukazuje použití třídy robot na jednoduchém programu pro blikání LED na kostce. Po spuštění tohoto programu z menu se periodicky každou sekundu mění stav LED na kostce. Po stisknutí levého tlačítka se program ukončí, na displeji se opět zobrazí menu a lze spustit další program.

```
# Importováni funkce pro uspáni programu
from time import sleep
# Importováni konstant tlačítek na kostce
from lib.robot_consts import Button

# Definice programu
def main:
    # Přistoupení ke globalní proměnné robot
    global robot

    # Smyčka čekající na ukončení programu
    while True:
        # Změna stavu LED
        robot.led.toggle

        # Získání stavu tlačítek
        buttons = robot.buttons.pressed
        # Ukončení programu, pokud je levé tlačítko stisknuté
        if buttons[Button.LEFT]:
            break

        # Uspání programu na 1 sekundu
        sleep(1)

# Spuštění programu
main
```

Kód 1: Použití globální třídy robot.

3.1 Tlačítka

class Buttons()

Informace o stavu stisknutí tlačítek na kostce.

Inicializace:	Zapnutí kostky.
Deinicializace:	Vypnutí kostky.
Přístup:	`robot.buttons`

pressed()

Navrátí stav tlačítek. Pro zjištění stavu konrétního tlačítka lze použít konstanty tlačítek.

Vrací:	Tuple stavu tlačítek zapnout, doleva, doprava, ok, nahoru, dolů. True pokud je tlačítko stisknuté, False pokud není.
Typ:	(bool, bool, bool, bool, bool, bool)

```
# Importování funkce pro uspání programu
from time import sleep
# Importování konstant tlačítek na kostce

def main:
  global robot
  while True:

    # Získání stavu tlačítek
    buttons = robot.buttons.pressed

    # Zobrazení stavu tlačítek v~terminálu
    print("Tlačítko stisknuté:",
          "zapnout:", buttons[Button.POWER],
          "doleva:",  buttons[Button.LEFT],
          "doprava:", buttons[Button.RIGHT],
          "OK:",      buttons[Button.OK],
          "nahoru:",  buttons[Button.UP],
          "dolů:",    buttons[Button.DOWN])
    # Ukončení programu, pokud je levé tlačítko stisknuté
    if buttons[Button.LEFT]:
      break
    # Uspání programu na 1 sekundu
    sleep(1)

# Spuštění programu
main()
```

Kód 2: Zjištění stavu tlačítek.

3.2 LED

class **Led**()

Zapínání a vypínání LED uvnitř kostky.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.led`

on()

Zapne LED.

off()

Vypne LED.

toggle()

Změní stav LED.

3.3 Bzučák

class **Buzzer**()

Ovládání bzučáku (piezoelektrického reproduktoru) v kostce.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.buzzer`

set_freq_duty(*freq*, *duty*)

Nastaví PWM na bzučáku na požadovanou frekvenci a střídu.

Parametry:
- **freq** (frekvence: Hz) – Frekvence PWM.
- **duty** (střída: %) – Střída PWM.

off()

Vypne bzučák.

3.4 Baterie

class Battery()

Měření napětí na napájecích bateriích. Při inicializaci se spustí časovač s periodou 200 ms, který spouští měření napětí.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.battery`

voltage()

Navrátí poslední změřené napětí na bateriích.

Vrací: Poslední změřené napětí na bateriích.
Typ: float: V

read_voltage()

Změří napětí na bateriích.

Vrací: Poslední změřené napětí na bateriích.
Typ: float: V

deinit()

Vypne periodické měření napětí.

3.5 Displej

class SH1106_I2C()

Zobrazování textu, geometrických tvarů a grafů na displeji. Aktivování jednotlivých pixelů se zaznamenává do frame bufferu, který je příkazem `show` celý přenesen do displeje.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.display`.

show()

Zobrazí aktuální frame buffer na displeji.

fill(*color*)

Vyplní celý frame buffer určenou barvou.

Parametry: **color** (0/1) – Barva – 0 černá, 1 bílá.

pixel(*x, y [, color]*)

Nastaví pixel ve frame bufferu na určenou barvu. Pokud barva není určena, navrátí nastavenou barvu pixelu.

Parametry:
- **x**, **y** – Souřadnice pixelu.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

Vrací: Barva pixelu.

Typ: int (0/1)

hline(*x, y, w, color*)

Nakreslí horizontální linii do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice levého začátku linie.
- **w** – Šířka linie.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

vline(*x, y, h, color*)

Nakreslí horizontální linii do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice horního začátku linie.
- **h** – Výška linie.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

line(*x1, y1, x2, y2, color*)

Nakreslí linii do frame bufferu.

Parametry:
- **x1**, **y1** – Souřadnic začátku linie.
- **x2**, **y2** – Souřadnice konce linie.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

rect(*x, y, w, h, color*)

Nakreslí obdélník do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice levého horního vrcholu obdélníku.
- **w**, **h** – Výška a šířka obdélníku.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

fill_rect(*x, y, w, h, color*)

Nakreslí vyplněný obdélník do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice levého horního vrcholu obdélníku.
- **w**, **h** – Výška a šířka obdélníku.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

ellipse(*x, y, xr, yr, color, f=False, m=1111b*)

Nakreslí elipsu do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice středu elispy.
- **xr**, **yr** – Délka poloos elipsy.
- **color** (0/1) – Barva – 0 černá, 1 bílá.
- **f** (bool) – Vyplnění elipsy pokud je parametr specifikován a má hodnotu True.
- **m** (4 bity) – Omezení vykreslení elipsy do určených kvadrantů. Bit 0 určuje Q1, b1 Q2, b2 Q3 a b3 Q4. Kvadranty jsou číslovány proti směru hodinových ručiček a Q1 je pravý horní kvadrant.

fill_rect(*x1, y1, x2, y2, x3, y3, color*)

Nakreslí vyplněný trojúhelník do frame bufferu.

Parametry:
- **x1**, **y1**, **x2**, **y2**, **x3**, **y3** – Souřadnice vrcholů trojúhelníku.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

text(*s, x, y, color=1*)

Nakreslí text do frame bufferu. Znaky mají rozměr 8x8 pixelů.

Parametry:
- **s** (string) – Text.
- **x**, **y** – Souřadnice levého horního rohu začátku textu.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

draw_bar_chart_v(*value, x, y, w, h, low_lim=0, high_lim=100, no_of_tics=5, label=None, redraw=False*)

Nakreslí vertikální sloupcový graf do frame bufferu.

Vrací: Překreslení grafu pro snížení problikávání grafu.

Typ: bool

Parametry:
- **value** (float) – Zobrazovaná hodnota.
- **x**, **y** (int) – Souřadnice levého dolního rohu začátku grafu.
- **w**, **h** (int) – Šířka a výska grafu.
- **low_lim**, **high_lim** (int) – Dolní a horní omezení grafu.
- **no_of_tics** (int) – Počet rozdělení grafu pro lepší čitelnost. Doporučená maximální hodnota je 5.
- **label** (string) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

draw_bar_chart_h(*value, x, y, w, h, low_lim=0, high_lim=100, no_of_tics=5, label=None, redraw=False*)

Nakreslí horizontální sloupcový graf do frame bufferu.

Vrací: Překreslení grafu pro snížení problikávání grafu.
Typ: bool
Parametry:
- **value** (float) – Zobrazovaná hodnota.
- **x**, **y** (int) – Souřadnice levého dolního rohu začátku grafu.
- **w**, **h** (int)– Šířka a výska grafu.
- **low_lim**, **high_lim** (int) – Dolní a horní omezení grafu.
- **no_of_tics** (int) – Počet rozdělení grafu pro lepší čitelnost. Doporučená maximální hodnota je 5.
- **label** (string) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

draw_dial(*value, x, y, r, loval, hival, no_of_steps, label, redraw*)

Nakreslí kruhový číselník do frame bufferu.

Vrací: Překreslení grafu pro snížení problikávání číselníku.
Typ: bool
Parametry:
- **value** (float) – Zobrazovaná hodnota.
- **x**, **y** (int) – Souřadnice středu číselníku
- **r** (int)– Poloměr číselníku.
- **loval**, **hival** (int) – Dolní a horní omezení číselníku.
- **no_of_steps** (int) – Počet rozdělení číselníku. Doporučená maximální hodnota je 10.
- **label** (string) – Popisek číselníku.
- **redraw** (bool) – Překreslení číselníku.

continuous_graph(*x, y, gx, gy, w, h, xlo, xhi, ylo, yhi, label, redraw*)

Nakreslí průběžný graf do frame bufferu. Funkce si vnitřně udržuje hodnoty grafu. Při vyplnění šířky grafu se předchozí hodnoty odstraní a graf se začne vykreslovat od počátku.

Vrací: Překreslení grafu pro snížení problikávání grafu.
Typ: bool
Parametry:
- **x**, **y** (float) – Zobrazovaná souřadnice v grafu.
- **gx**, **gy** (int) – Souřadnice levého dolního rohu grafu.
- **w**, **h** (int) – Šířka a výska grafu.
- **xlo**, **xhi** (int) – Dolní a horní omezení osy x.
- **ylo**, **yhi** (int) – Dolní a horní omezení osy y.
- **label** (string) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

```
# Vymazání displeje
robot.display.fill(0)
# Zapsání textu do frame bufferu
robot.display.text("test message", 0, 0, 1)
# Zobrazení frame bufferu na displeji
robot.display.show()
```

Kód 3: Použití displeje.

3.6 Gyroskop a akcelerometr

class ICM20608()

Gyroskop a akcelerometr. Měření zrychlení a úhlové rychlosti ve třech osách. Interrupt pin senzoru je připojen na GyroAcc.IRQ_PIN (GPIO 28) mikrokontroleru.

Inicializace:	`robot.init_sensor(sensor_type=Sensor.GYRO)`
Deinicializace:	`robot.deinit_sensor(sensor_type=Sensor.GYRO)`
Přístup:	`robot.sensors.gyro`

read_value()

Přečte ze senzoru a navrátí změřené hodnoty úhlových rychlostí a zrychlení. Pro zjištění konkrétní hodnoty lze použít GyroAcc konstanty gyroskopu a akcelerometru.

Vrací:	Touple zrychlení a úhlových rychlostí v osách x, y, z.
Typ:	(a_x: -, a_y: -, a_z: -, g_x: °/s, g_y: °/s, g_z: °/s)

Příklad čtení dat z gyroskopu s využitím irq:

```
# Importování knihoven
from time import sleep
from machine import Pin
# Importování konstant
from lib.robot_consts import Button, Sensor, GyroAcc

a_g_data = (0,0,0,0,0,0)

def main:
  global robot, a_g_data
  # Inicializace senzoru
  robot.init_sensor(sensor_type=Sensor.GYRO)
  # Nastavení GPIO 28
  icm_irq_pin = Pin(GyroAcc.IRQ_PIN, Pin.IN)
  # Nastavení callback funkce volané na náběžnou hranu irq, pokud jsou
    nová data k dispozici
  icm_irq_pin.irq(trigger=Pin.IRQ_RISING, handler=callback)

  # Smyčka čekající na ukončení programu
  while True:
    # Zobrazení posledních hodnot zrychlení a úhlové rychlosti
    print("Acc:",  a_g_data[GyroAcc.AX],
                   a_g_data[GyroAcc.AY],
                   a_g_data[GyroAcc.AZ],
          "Gyro:", a_g_data[GyroAcc.GX],
                   a_g_data[GyroAcc.GY],
                   a_g_data[GyroAcc.GZ],)
    # Získání stavu tlačítek
    buttons = robot.buttons.pressed
    # Ukončení programu, pokud je levé tlačítko stisknuté
    if buttons[Button.LEFT]:
      break
    # Uspání programu na 1 sekundu
    sleep(1)

  # Zrušení callbacku před deinicializací senzoru
```

```
36   icm_irq_pin.irq(handler=None)
37   # Deinicializace senzoru
38   robot.deinit_sensor(sensor_type=Sensor.GYRO)
39
40 def callback(p):
41   global robot, a_g_data
42   # Přečtení nových hodnot ze senzoru
43   a_g_data = robot.sensors.gyro.read_value
44
45 # Spuštění programu
46 main
```

Kód 4: Čtení dat z gyroskopu s využitím irq.

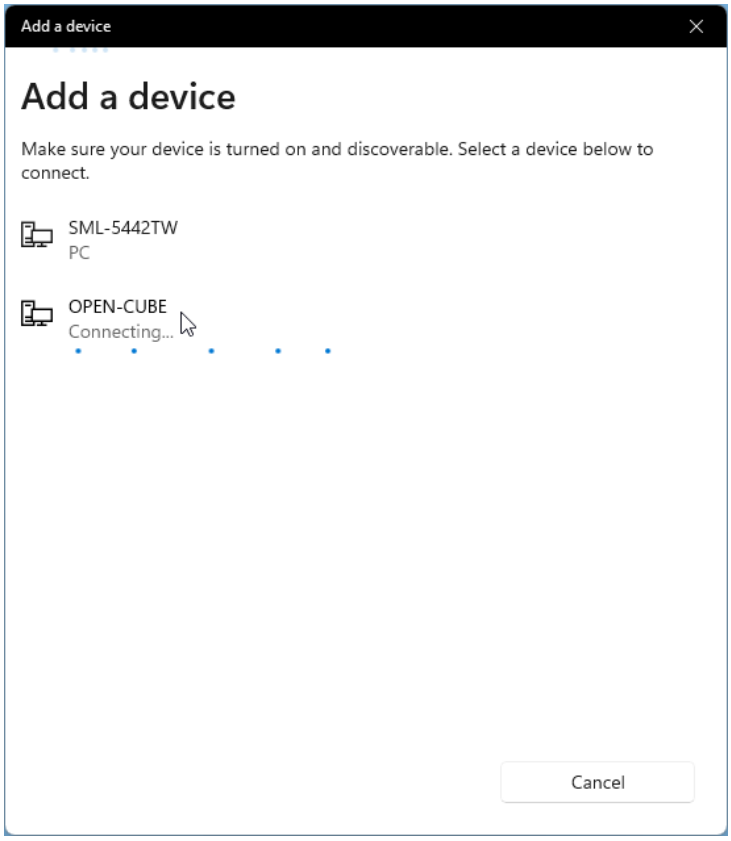
3.7 ESP32 komunikace

ESP32 komunikuje s kostkou pomocí sériové linky. S dalším zařízením může komunikovat pomocí Wi-Fi nebo Bluetooth. ESP je programovatelné po připojení USB kabelem na vrchní straně kostky. Výchozí firmware ESP (dostupný z Open-Cube repozitáře [https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/ESP]) pouze přeposílá data z kostky přes Bluetooth virtuální COM port do počítače a obráceně. Zkušenější uživatel si může naimplementovat vlastní komunikaci.

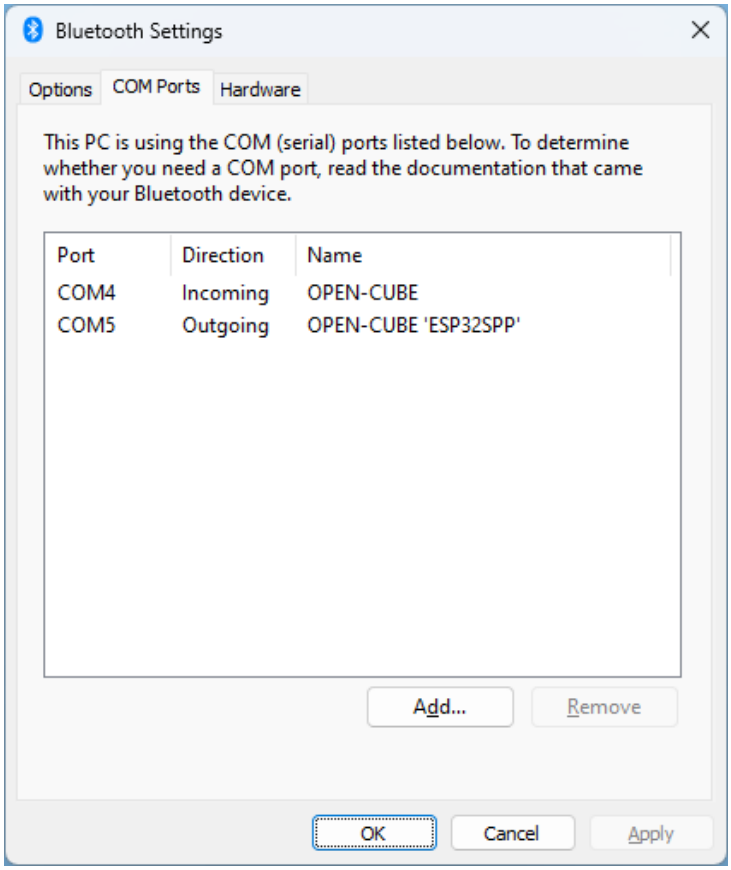
3.7.1 Bluetooth propojení ESP32 s počítačem

Komunikace probíhá sériově přes Bluetooth virtuální COM port a je otestováno pouze pro Windows.

1. Zapněte kostku.
2. V Bluetooth nastavení zvolte Přidat nové zařízení, dále Bluetooth a spárujte počítač s kostkou. Viz obrázek 1.
3. Pro zjištění čísla COM portu pokračujte do pokročilých nastavení Bluetooth. Požadovaný COM port má název obsahující ESP32SPP a směr Outgoing. Viz obrázek 2.
4. Tento COM port lze použít pro přijímání i odesílání dat.



Obrázek 1: Spárování počítače s kostkou.



Obrázek 2: Zjištění čísla COM portu.

3.7.2 Kostka

class **EspUart**()

Oboustranné posílání dat mezi mikroporcesorem a ESP32 pomocí sběrnice UART. Při inicializaci se spustí časovač s periodou 10 ms, při kterém se přijímají a ukládají data posílaná ESP.

Data se posílají po zprávách, což jsou celky dat patřící k sobě. Odesílání a přijímání dat může probíhat v ASCII nebo v binárním režimu.

ASCII režim: Odesílaná zpráva je zakončena terminátorem LF (new line). Při přijímání dat je zpráva k dispozici po přijetí terminátoru LF nebo CR. Data jsou typu string.

Binární režim: Pro komunikaci je nutné specifikovat hlavičku odesílaných a přijímaných dat, aby bylo možné rozeznat začátek zprávy. Aby bylo možné zprávu správně přijmout a určit její konec, musí být předem specifikovaná velikost zprávy v bytech. Data jsou typu bytes a jejich interpretace je ponechána na uživateli.

Inicializace: `robot.init_esp_uart()`
Deinicializace: `robot.deinit_esp_uart()`
Přístup: `robot.esp_uart`

read()

Navrátí poslední přijatou zprávu.

Vrací: Přijatá zpráva. Pokud není nová zpráva k dispozici: None.
Typ:
- ASCII režim: string
- Binární režim: bytes

write(*buff*)

Odešle zprávu.

Parametry: **buff** (buffer: string(ASCII)/bytes(binární)) – Buffer odesílaných dat.

set_binary(*header, data_size*)

Nastaví komunikaci binárního režimu. Pokud je header None, komunikace se nastaví do ACII režimu.

Parametry: **header** (tuple) – Hlavička libovolné délky obsahující čísla 0–255, nebo prvek None.
data_size (int) – Velikost přijímané zprávy v bytech.

Příklad použití komunikace mezi mikroprocesorem a ESP32:

```
# Importování knihovny pro práci s binárními daty
import struct

robot.esp_uart_init()
# Odeslání a přijetí zprávy typu string v režimu ASCII
robot.esp_uart.write("test message")
received_message = robot.esp_uart.read()
```

```
8
9  # Odeslání a přijetí zprávy o velikosti 8 bytů v binárním režimu
10 robot.esp_uart.set_binary((112, 241, 3, 62), 8)
11 # Vytvoření a odeslání zprávy typu bytes se dvěma čísly typu single
12 write_message = struct.pack("<ff", 521.9, -11.821)
13 robot.esp_uart.write(write_message)
14 # Příjem a dekódování zprávy obsahující dvě čísla typu single
15 received_message2 = robot.esp_uart.read()
16 (value1, value2) = struct.unpack("<ff", received_message2)
```

Kód 5: Komunikace mezi mikroprocesorem a ESP32.

3.7.2.1 Matlab

Pro komunikaci pomocí Matlabu je zapotřebí mít nainstalovaný Communications Toolbox. Toto rozhraní používá Bluetooth SPP a umožňuje přijímat a odesílat ASCII a binární data. Při posílání ASCII dat je zpráva zakončena terminátorem. V případě binárních dat zpráva není zakončena terminátorem a kostka i program v Matlabu musí předem znát délku zprávy, aby mohla být správně interpretovaná.

Příklad použití rozhraní je uveden v následujícím kódu. Další informace k použití rozhraní naleznete v oficiální dokumentaci [https://www.mathworks.com/help/matlab/bluetooth-communication.html/].

```
1  % Propojení kostky s Matlabem.
2  cube = bluetooth('OPEN-CUBE')
3  % Nastavená terminátoru LF (new line) pro odesílání a příjem
       ASCII dat.
4  configureTerminator(device, 'LF')
5  %%
6  % Poslání testovací zprávy do kostky. Zpráva je zakončena
       nastaveným terminátorem.
7  writeline(cube, 'test message')
8  % Příjmutí zprávy z kostky. Funkce přijímá data a čeká na
       zakončení nastaveným terminátorem.
9  cube_message = readline(cube)
10 %%
11 % Poslání testovací zprávy do kostky. Zpráva je
       interpretovaná jako formát string a není zakončena
       terminátorem.
12 write(cube, 'test message', 'string')
13 % Příjmutí zprávy z kostky. Zpráva má délku 8 bytů a je
       interpretovaná jako formát float.
14 cube_message2 = read(cube, 8, 'float')
```

Kód 6: Komunikace s kostkou z Matlabu.

3.7.2.2 Simulink

Pro komunikaci pomocí Simulinku je zapotřebí mít nainstalovaný Instrument Control Toolbox, který poskytuje bloky `Serial Configuration`, `Serial Send` a `Serial Receive` pro sériovou komunikaci. Při vytvoření nového modelu je potřeba upravit nastavení řešiče simulace a nastavit parametry sériové komunikace:

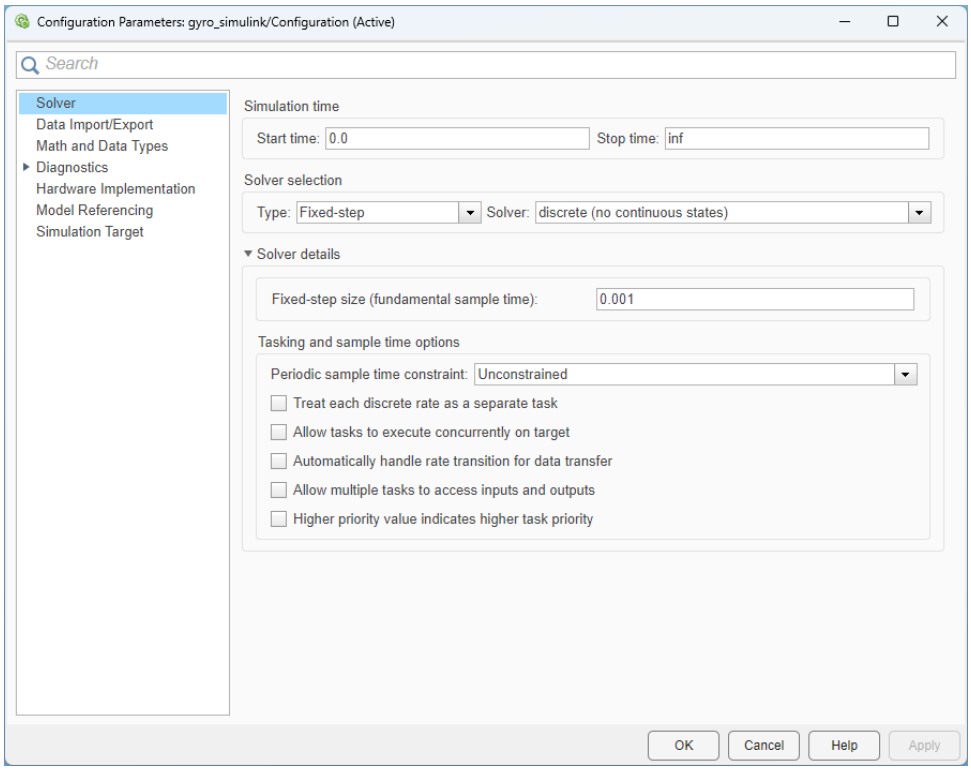
1. Kliknutím na tlačítko v pravém dolním rohu nastavit řešič na `Fixed-Step`, `discrete` a `Fixed-step size` na hodnotu podle frekvence odesílání a přijímání dat (např. 0,001). Viz obrázek 3

2. Kliknutím na šipku pod tlačítkem `Run` zapnout `Simulation Pacing`, viz obrázky 4 a 5.

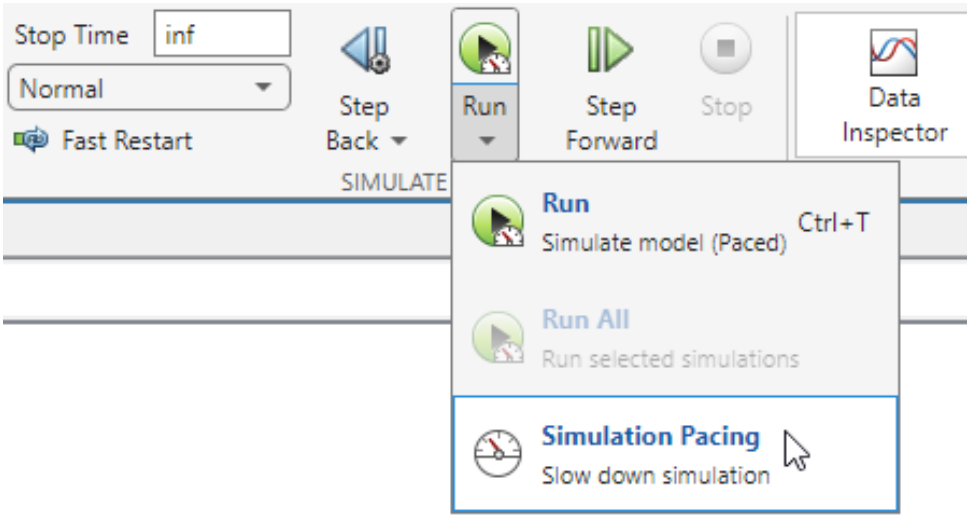
3. V modelu vytvořit blok `Serial Configuration`, ve kterém nastavit COM port, zjištěný v kapitole 3.7.1 a ostatní parametry podle obrázku 6.

Po tomto nastavení je možné použít bloky `Serial Send` a `Serial Receive` pro odesílání a přijímání dat. Informace k použití bloků naleznete v oficiální dokumentaci [https://www.mathworks.com/help/instrument/direct-interface-communication-in-simulink.html].

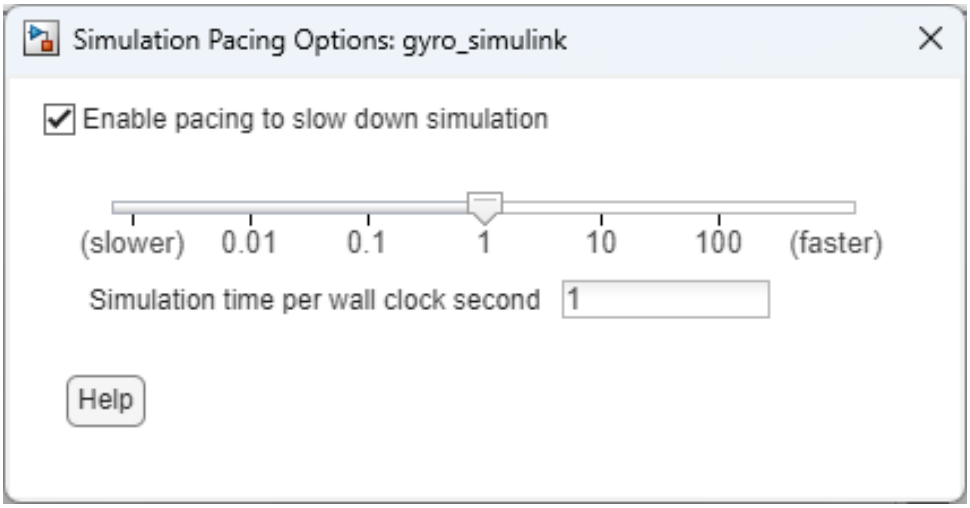
Příklad použití sériové komunikace pro nastavení konstant PID regulátoru v kostce a zobrazení aktuálních složek PID je na obrázku 7. V tomto příkladu se hodnoty odesílají a přijímají v binárním formátu, je nastavena hlavička pro odesílání a přijímaní a data jsou typu single (4 byty). Data se přijímají každé 0,01 sekundy, což je nastaveno v bloku `Serial Send` v kolonce `Block sample time`. Data se odesílají každou sekundu, což je nastaveno v kolonkách `Block sample time` tří bloků `Constant`. Nastavený `Fixed-step size` řešiče by měl být menší než `Block sample time`.



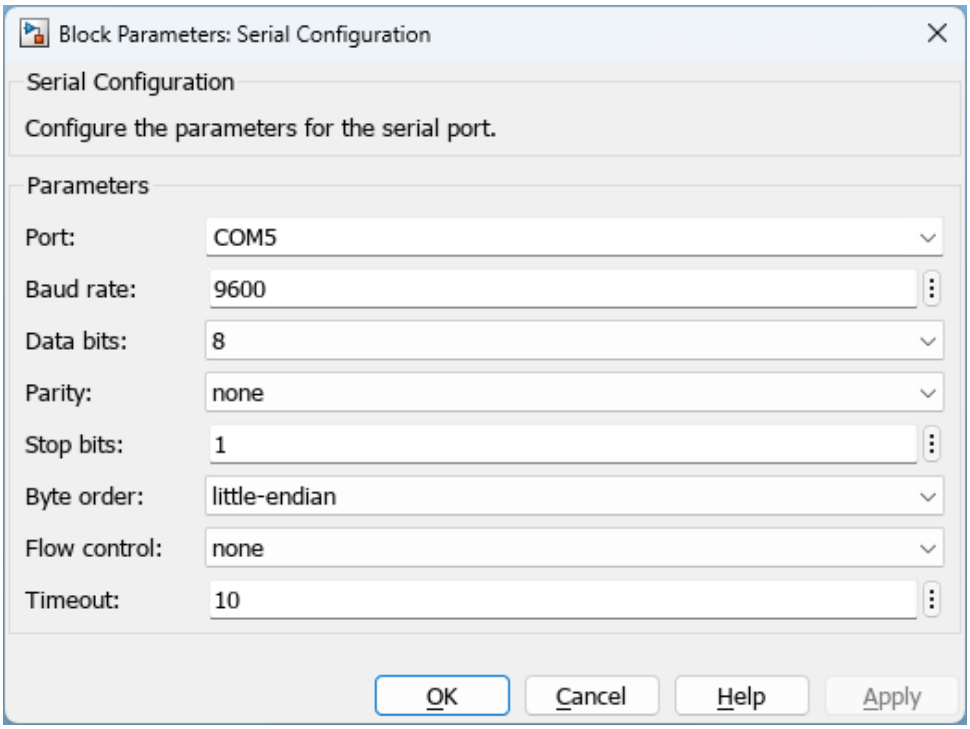
Obrázek 3: Nastavení řešiče Simulinku.



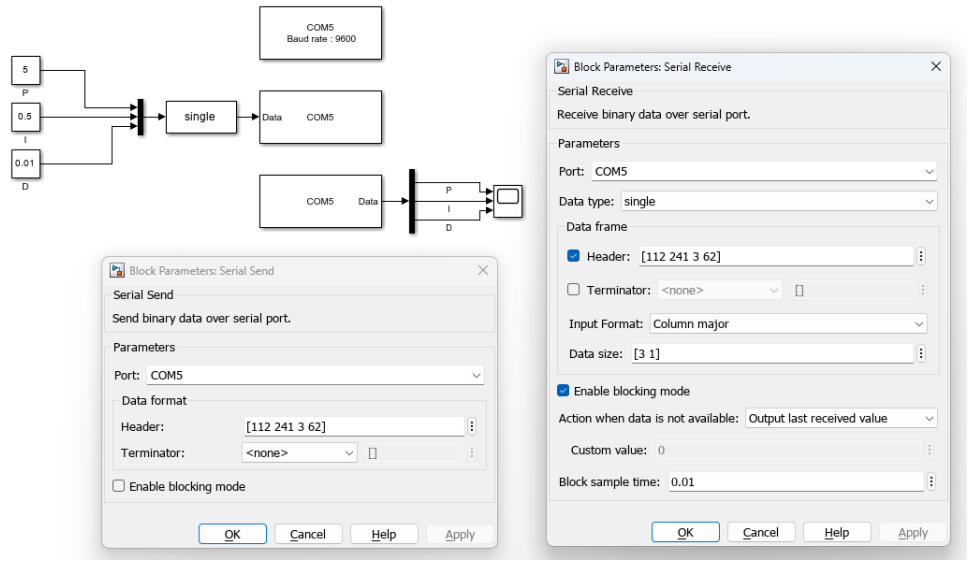
Obrázek 4: Umístění nastavení Simulation pacing v Simulinku.



Obrázek 5: Nastavení Simulation Pacing.



Obrázek 6: Nastavení bloku Serial Configuration v Simulinku.



Obrázek 7: Příklad použití sériové komunikace v Simulinku.

4 NXT senzory

4.1 Tlačítko

class **TouchSensor**(*port*)

Informace o stavu stisknutí NXT tlačítka.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.TOUCH,port=Port)`.
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.TOUCH,port=Port)`.
Přístup: `robot.sensors.touch[Port]`.

pressed()

Navrátí stav tlačítka.

Vrací: True pokud je tlačítko stisknuté, False pokud není.
Typ: bool

```
# Importováni konstant typů senzorů a portů
from lib.robot_consts import Sensor, Port

# Inicializace NXT tlačítka na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.TOUCH, port=Port.S1)

# Zjištění stavu stisknutí tlačítka
touch_pressed = robot.sensors.touch[Port.S1].pressed()
```

Kód 7: Použití NXT tlačítka.

4.2 Světelný senzor

class **LightSensor**(*port*)

Měření intenzity dopadajícího světla NXT světelného senzoru.

Měření je umožněno dvěma módy – manuální, při kterém je intenzita změřena na požádání a kontinuální, při kterém je intenzita měřena periodicky a na požádání je vrácena poslední změřená hodnota. Kontinuální mód dále poskytuje režimy neblikající a blikající. Při nastavení neblikajícího režimu je intenzita měřena ve stavu LED nastaveném uživatelem. V blikajícím režimu je intenzita měřena při vypnutém i zapnutém stavu LED. Změna stavu LED je prováděna automaticky a uživateli je na požádání k dispozici poslední změřená intenzita pro oba stavy LED.

AD převodník vzorkuje signál s periodou 1 ms. Intenzita dopadajícího světla je získána měřením napětí na senzoru. Při změně stavu LED se toto napětí ustálí za přibližně 10 ms.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.LIGHT,port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.LIGHT,port=Port)`
Přístup: `robot.sensors.light[Port]`

on()

Zapne LED.

off()

Vypne LED.

togggle()

Změní stav LED.

intensity(*pin_on*)

V manuálním módu změří intenzitu světla a navrátí ji. V kontinuálním módu navrátí poslední změřenou intenzitu.

Parametry: **pin_on** (bool) – Volba navrácené změřené intenzity dopadajícího světla. V manuálním módu a v kontinuálním módu s neblikajícím režimem na hodnotě nezáleží. V kontinuálním módu s blikajícím režimem navrátí pro True intenzitu změřenou při zapnuté LED a pro False při vypnuté LED.

Vrací: Intenzita dopadajícího světla na senzor. 0 pro největší intenzitu, 32768 pro nejmenší.

Typ: int (0–32768)

set_continuous(*period_us, wait_us*)

Nastaví měření do kontinuálního módu.

Parametry:
- **period_us** (int) – Doba mezi začátkem konverze analogové hodnoty a vyčtením konvertované digitální hodnoty z AD převodníku. Doporučená minimální hodnota je 1100.
- **wait_us** (int) – Doba čekání před měřením po změně stavu LED v mikrosekundách. Doporučená minimální hodnota je 10000.

stop_continuous()

Nastaví měření do manuálního módu.

set_switching()

Nastaví měření kontinuálního módu do blikajícího režimu.

stop_switching()

Nastaví měření kontinuálního módu do neblikajícího režimu.

```
# Importováni konstant typů senzorů a portů
from lib.robot_consts import Sensor, Port

# Inicializace NXT světelného senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.LIGHT, port=Port.S1)

# Zapnutí LED senzoru
robot.sensors.light[Port.S1].on()
# Změření intenzity světla
light_intensity = robot.sensors.light[Port.S1].intensity()
```

Kód 8: Použití NXT světelného senzoru.

4.3 Ultrazvukový senzor

class **UltrasonicSensor**()

Měření vzdálenosti objektu od senzoru v rozmezí 0–255 cm s přesností ±3 cm.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.ULTRA)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.ULTRA)`
Přístup: `robot.sensors.ultra`

distance()

Změří vzdálenost a navrátí ji.

Vrací: Vzdálenost objektu od senzoru.
Typ: int (0-255)

```
# Importováni konstant typů senzorů
from lib.robot_consts import Sensor

# Inicializace NXT ultrazvukového senzoru
robot.init_sensor(sensor_type=Sensor.ULTRA)

# Změření vzdálenosti
distance = robot.sensors.ultra.distance()
```

Kód 9: Použití NXT ultrazvukového senzoru.

4.4 Zvukový senzor

class **SoundSensor**(*port*)

Měření intenzity zvuku NXT zvukového senzoru.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.SOUND,port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.SOUND,port=Port)`
Přístup: `robot.sensors.sound[Port]`

intensity()

Změří intenzitu zvuku a navrátí ji.

Vrací: Intenzita zvuku. 0 pro největší intenzitu, 32768 pro nejmenší.
Typ: int (0–32768)

```
# Importováni konstant typů senzorů a portů
from lib.robot_consts import Sensor, Port

# Inicializace NXT zvukového senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.SOUND, port=Port.S1)

# Změření hladiny zvuku
sound_intensity = robot.sensors.sound[Port.S1].intensity()
```

Kód 10: Použití NXT zvukového senzoru.

5 Servomotor

Motory ze sady NXT a EV3 jsou vnitřními parametry identické, liší se pouze vzhledem a plastovou konstrukcí.

class **Motor**(*port*)

Na motorech se nachází rotační enkodéry umožňující snímání polohy natočení motoru a aproximaci rychlosti otáčení.

Poskytuje tři módy řízení – výkon, poloha, rychlost.

V módu řízení výkonu uživatel přímo nastavuje požadovaný výkon na motoru v rozmezí -100–100 %.

V módu řízení polohy uživatel volí požadovanou polohu natočení motoru ve °. Regulace je zprostředkována PID regulátorem s nastavitelnými konstantami.

V módu řízení rychlosti uživatel volí požadovanou rychlost otáčení motoru ve °/s. Maximální rychlost při nezatíženém motoru je přibližně 9500 °/s. Regulace je zprostředkována PID regulátorem s nastavitelnými konstantami.

Parametry: **port** (Port) – Port, ke kterému je motor připojen.
Inicializace: `robot.init_motor(port=Port)`
Deinicializace: `robot.deinit_motor(port=Port)`
Přístup: `robot.motor[Port]`

Parametry: **port** (Port) – Port, ke kterému je motor připojen.

set_power(*power*)

Nastaví výkon na motoru.

Parametry: **power** (float) – Požadovaný výkon na motoru v rozmezí -100–100 %.

init_encoder()

Inicializuje enkodér s měřením polohy a rychlosti motoru.

deinit_encoder()

Denicializuje enkodér s měřením polohy a rychlosti motoru.

position()

Navrátí polohu motoru.

Vrací: Poloha motoru.

Typ: úhel: °

speed()

Navrátí polohu motoru.

Vrací: Rychlost motoru.

Typ: úhlová rychlost: °/s

init_regulator()

Inicializuje regulátor motoru.

deinit_regulator()

Denicializuje regulátor motoru.

set_regulator_position(*position*)

Nastaví motor do módu řízení polohy.

Parametry: **position** (int) – Požadovaná konečná poloha motoru ve °.

regulator_pos_set_consts(*p, i, d*)

Nastaví PID konstanty regulátoru polohy.

Parametry:
- **p** (float) – Konstanta proporcionální složky.
- **i** (float) – Konstanta integrační složky.
- **d** (float) – Konstanta derivační složky.

set_regulator_speed(*speed*)

Nastaví motor do módu řízení polohy.

Parametry: **speed** (int) – Požadovaná konečná rychlost motoru ve °/s.

regulator_speed_set_consts(*p, i, d*)

Nastaví PID konstanty regulátoru rychlosti.

Parametry:
- **p** (float) – Konstanta proporcionální složky.
- **i** (float) – Konstanta integrační složky.
- **d** (float) – Konstanta derivační složky.

regulator_error()

Navrátí aktuální regulační odchylku.

Vrací: Regulační odchylka.

Typ:
- **mód řízení polohy** – poloha: °
- **mód řízení rychlosti** – úhlová rychlost: °/s

regulator_power()

Navrátí aktuální akční zásah regulátoru.

Vrací: Akční zásah.

Typ: výkon (-100–100 %)

```
# Importováni konstant portů
from lib.robot_consts import Port

# Inicializace motorů na motorových portech 1 a 2
robot.init_motor(Port.M1)
robot.init_motor(Port.M2)

# Nastavení výkonu motorů
robot.motors[Port.M1].set_power(50)
robot.motors[Port.M2].set_power(-20)

# Zjištění aktuální pozice a rychlosti motorů
pos1 = robot.motors[Port.M1].position()
pos2 = robot.motors[Port.M2].position()
speed1 = robot.motors[Port.M1].speed()
speed2 = robot.motors[Port.M2].speed()
```

Kód 11: Použití motorů.

6 Parametry a konstanty

6.1 Sensor

class **Sensor**

Typy senzorů.

NO_SENSOR

Žádný senzor.

LIGHT

NXT světelný senzor.

ULTRA

NXT ultrazvukový senzor vzdálenosti.

TOUCH

NXT tlačítko.

SOUND

NXT zvukový senzor.

GYRO

ICM20608-G gyroskop a akcelerometr.

Příklad použití motorů je uveden

6.2 Port

class **Port**

Porty motorů a senzorů na kostce.

M1

Port motoru 1.

M2

Port motoru 2.

M3

Port motoru 3.

M4

Port motoru 4.

S1

Port senzoru 1.

S2

Port senzoru 2.

S3

Port senzoru 3.

S4

Port senzoru 4.

6.3 Button

class **Button**

Tlačítka na kostce.

POWER

Tlačítko zapnout.

LEFT

Tlačítko doleva.

RIGHT

Tlačítko doprava.

OK

Tlačítko OK.

UP

Tlačítko nahoru.

DOWN

Tlačítko dolů.

6.4 Light

class **Light**

Změřená hodnota při vypnuté a zapnuté LED světelného senzoru.

OFF

Změřená hodnota při vypnuté LED.

ON

Změřená hodnota při zapnuté LED.

6.5 GyroAcc

class **GyroAcc**

Změřená hodnota gyroskopu a akcelerometru ICM20608-G v osách x, y, z a pin přerušení senzoru.

AX

Zrychlení z akcelerometru v ose x.

AY

Zrychlení z akcelerometru v ose y.

AZ

Zrychlení z akcelerometru v ose z.

GX

Úhlová rychlost z gyroskopu v ose x.

GY

Úhlová rychlost z gyroskopu v ose y.

GZ

Úhlová rychlost z gyroskopu v ose z.

IRQ_PIN

Pin přerušení senzoru ICM20608-G.