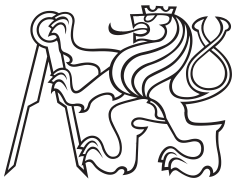
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řídící techniky

Knihovna pro senzory a akční členy nové FEL Open - Cube

Jakub Vaněk

Vedoucí: Ing. Martin Hlinovský, Ph.D.
Studijní program: Kybernetika a robotika
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vaněk** Jméno: **Jakub** Osobní číslo: **499014**

Fakulta/ústav: **Fakulta elektrotechnická**

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

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Knihovna pro senzory a akční členy nové FEL Open - Cube

Název bakalářské práce anglicky:

The library for sensors and actuators of the new FEL Open - Cube

Pokyny pro vypracování:

1. Seznamte se s možnostmi nové FEL Open - Cube (z hlediska HW a SW vybavení) realizované jako náhrada kostky LEGO Mindstorms EV3.
2. Vytvořte knihovnu pro připojení základních senzorů a akčních členů ze stavebnice LEGO Mindstorms EV3.
3. Vyzkoušejte novou FEL Open - Cube s použitím vytvořené knihovny na dvou úlohách pro ROBOSOUTĚŽ

Seznam doporučené literatury:

[1] https://education.lego.com/v3/assets/blt293eea581807678a/blt471320cb14ed0291/5f880386631d5a2165df4144/lego-mindstorms-ev3-hardware-developer-kit.zi
[2] https://www.lego.com/cdn/cs/set/assets/blt86e79bb287a0d0b1/Appendix_LEGO_MINDSTORMS_EV3_programmable_brick_main_hardware_schematics.pdf
[3] https://gitlab.fel.cvut.cz/open-cube/hardware
[4] https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf
[5] https://docs.micropython.org/en/latest/library/index.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

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

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

Datum zadání bakalářské práce: **10.01.2023** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **22.09.2024**

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

Rád bych zde poděkoval svému vedoucímu, panu Ing. Martinu Hlinovskému, Ph.D., za veškerou poskytnutou podporu. Dále bych chtěl poděkovat Ing. Davidu Novotnému za pomoc při oživování kostky Open-Cube. Oběma bych rád poděkoval za možnost podílet se na jejím vývoji.

Prohlášení

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

V Praze, 26. května 2023

Abstrakt

Bakalářská práce se zabývá přidáním podpory pro senzory a motory ze stavebnice LEGO Mindstorms EV3 do nové řídící kostky Open-Cube. Krátce je zde popsána Open-Cube i sada EV3. Následně jsou vytvořeny ovladače pro MicroPython pro oba typy periferií. Pro řízení motorů je dále navržen regulátor zajišťující plynulé zrychlování motorů. Vzniklý kód je následně otestován na dvou úlohách pro Robosoutěž, každoroční soutěž robotů pořádanou na FEL ČVUT. Na závěr je na platformě GitLab automatizováno sestavování celkového MicroPython firmwaru.

Klíčová slova: LEGO, Mindstorms, EV3, Open-Cube, RP2040, MicroPython, GitLab

Vedoucí: Ing. Martin Hlinovský, Ph.D.
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky
Karlovo náměstí 13
120 00 Praha 2

Abstract

This bachelor thesis deals with adding support for sensors and motors from the LEGO Mindstorms EV3 kit to the new Open-Cube programmable brick. The Open-Cube and the EV3 kit are briefly described. Subsequently, new MicroPython drivers are created for both types of peripherals. Furthermore, a feedback controller is designed to make smooth acceleration of the motors possible. The resulting code is then tested on two tasks from Robocontest, an annual robotics competition hosted at FEE CTU. Finally, the build process of the MicroPython firmware is automated on the GitLab platform.

Keywords: LEGO, Mindstorms, EV3, Open-Cube, RP2040, MicroPython, GitLab

Title translation: The library for sensors and actuators of the new FEL Open - Cube

Obsah

Obrázky

Kapitola 1

Úvod

Robotické stavebnice jsou jednou z používaných pomůcek při výuce programování na středních i vysokých školách. Tyto stavebnice využívá i Fakulta elektrotechnická na jí pravidelně pořádané Robosoutěži [1]. Cílem účastníků je obvykle sestavit autonomního mobilního robota ze stavebnice LEGO Mindstorms NXT nebo novější LEGO Mindstorms EV3. Tito roboti musí co nejlépe a nejrychleji vyřešit předem zadanou úlohu [1].

Fakultou doposud používané stavebnice ale již začaly zastarávat. V létě 2021 ukončila firma LEGO prodej sad Mindstorms EV3 [2]. Tím se výrazně zhoršila dostupnost těchto stavebnic. Ty jsou přitom běžně zapůjčovány účastníkům Robosoutěže – mnohé střední a základní školy nemají stavebnice jinak k dispozici.

Jako pravděpodobnou náhradu za řadu EV3 uvedla společnost LEGO stavebnice řady SPIKE PRIME [2]. Z diskuzí s organizátory Robosoutěže ale vyplynulo, že tato stavebnice z mnoha důvodů není dobrou náhradou současných setů.

Katedra měření FEL proto začala vyvíjet alternativní řídící kostku – Open-Cube. Ta poskytuje vstupy a výstupy kompatibilní s oběma sety od firmy LEGO. Eventuálním cílem je vyvinout i nové senzory a motory, prozatím se ale počítá s využitím periferií ze starších stavebnic.

Cílem této práce je rozšířit řídící software kostky Open-Cube tak, aby dokázal spolupracovat se senzory a motory ze stavebnice Mindstorms EV3.

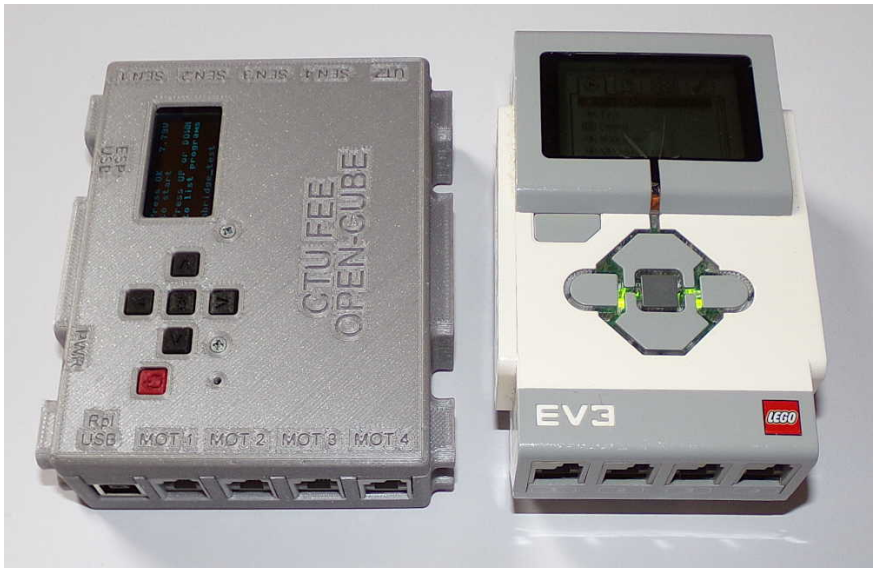
Kapitola 2

Úvodní přehled

2.1 Kostka Open-Cube

2.1.1 Základní informace

Open-Cube již byla krátce představena v úvodu. Na obrázku 2.1 je nová kostka porovnaná s řídící jednotkou sady EV3. Obě kostky mají podobný přístup k ovládání a lze je připevnit k robotům postaveným z LEGO Technic.



Obrázek 2.1: Nová kostka Open-Cube (vlevo) a starší jednotka EV3 (vpravo).

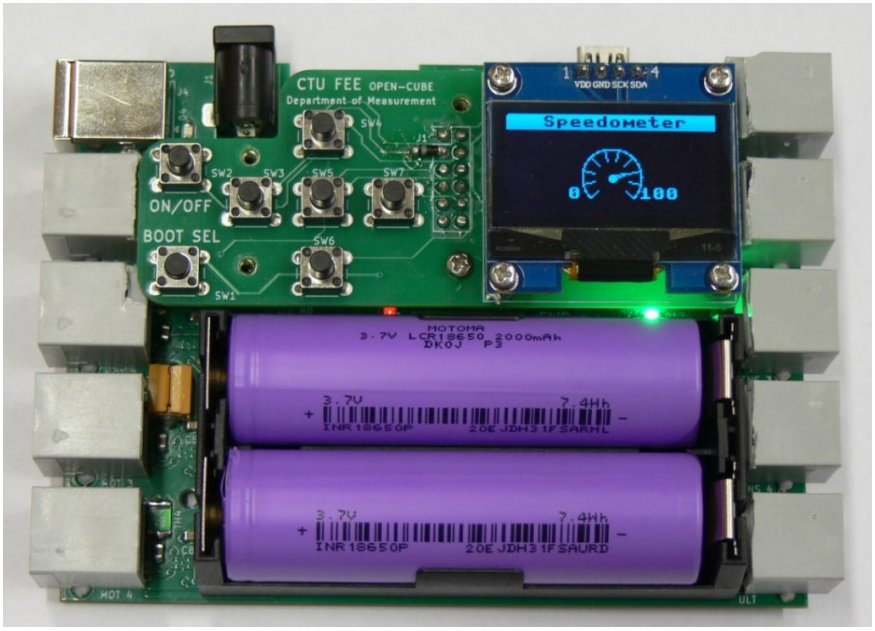
Pro připojení k dalším zařízením poskytuje Open-Cube tato rozhraní:

- Čtyři porty pro připojení NXT a EV3 motorů.
- Čtyři porty pro připojení analogových NXT senzorů a libovolných EV3 senzorů.

- Jeden port pro připojení NXT ultrazvukového senzoru.
- USB port pro připojení hlavního procesoru kostky k PC.
- Mini USB port pro programování zabudovaného modulu pro bezdrátovou komunikaci.

Bezdrátovou komunikaci Open-Cube podporuje přes Bluetooth a v budoucnu potenciálně též přes Wi-Fi. Pro tento účel má kostka zabudovaný modul ESP32. [3]

Oproti jednotkám NXT a EV3 má Open-Cube jinak řešené napájení. Dřívější řídící kostky závisely na vyměnitelných bateriích nebo externích Li-ion akumulátorech. Nová kostka má akumulátor i s nabíjecími obvody zabudovaný pod krytem, viz obrázek [4]. [3]



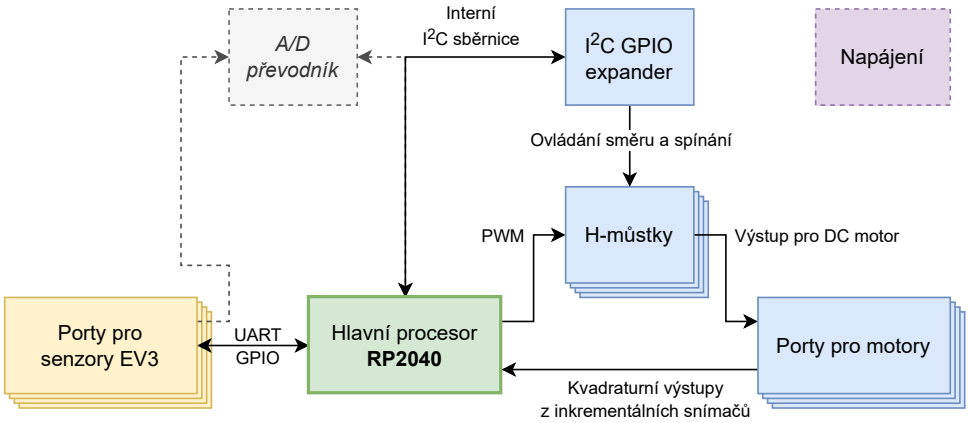
Obrázek 2.2: Pohled na část elektroniky pod krytem Open-Cube. [3]

2.1.2 Hardware

Open-Cube je postavená na mikrořadiči RP2040 od firmy Raspberry Pi [3]. Tento čip v sobě obsahuje dvě jádra ARM Cortex-M0+ na výchozí frekvenci 125 MHz. Z pohledu paměti nabízí 264 kB interní RAM a rychlé rozhraní pro externí Flash paměť. Dále poskytuje 30 obecných pinů a řadiče pro několik komunikačních protokolů, jako jsou například UART a I^2C. Další rozhraní je možné implementovat pomocí tzv. PIO stavových automatů blíže popsaných v sekci 2.1.2. [5]

Připojení procesoru k portům

Propojení mezi procesorem Open-Cube a vstupy/výstupy kostky naznačuje obrázek 2.3.



Obrázek 2.3: Zjednodušené blokové schéma Open-Cube zachycující propojení s jejími porty. Založeno na podrobnějším blokovém schématu z [3].

Na senzorové porty má kostka z procesoru přivedené obousměrné sériové linky. Ty jsou určené pro komunikaci s EV3 senzory, které jsou blíže popsány v kapitole 2.3.2. Jeden pin portů je také připojený k analogově-číslicovému převodníku, ten se ale pro EV3 senzory nevyužívá. [4]

Ovládání motorů závisí na zabudovaných H-můstcích, které spínají vinutí motorů. Můstky lze částečně řídit přímo z RP2040, pro některé funkce je ale nutné využít I^2C pin expandér. Pomocí něj autoři Open-Cube rozšířili počet pinů, které dokáže RP2040 ovládat. [4]

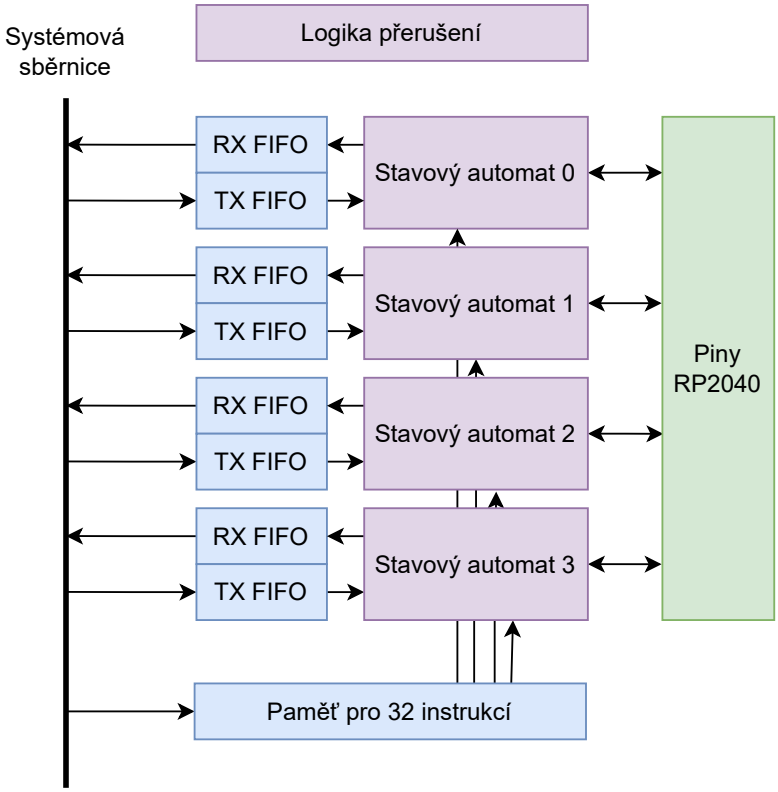
Pro zjišťování polohy motorů se počítá s použitím kvadraturních inkrementárních čidel polohy. Blíže jsou popsaná v sekci 2.3.3. Jejich odrušený výstup vede přímo do RP2040 [4]. Při další práci bude nezbytné naprogramovat jejich softwarové dekódování, protože RP2040 k tomu neposkytuje jinou vhodnou hardwarovou periferii.

Bloky programovatelného I/O

Předchozí kapitola zmiňuje, že do RP2040 vedou od portů čtyři sériové linky. RP2040 přitom ale obsahuje pouze dvě hardwarové UART periferie a jenom jedna z nich je připojitelná k senzorovému portu [4][6, s. 13].

S touto komplikací se počítalo již při návrhu Open-Cube. Řešením je použít tzv. PIO (programmable I/O) periferii uvnitř RP2040. Na tu lze pohlížet jako na sadu programovatelných posuvných registrů. Jedním z jejich cílů je právě flexibilní implementace různých sériových nebo paralelních rozhraní [6].

Blokové schéma jednoho PIO bloku ukazuje obrázek 2.3. RP2040 obsahuje dva nezávislé PIO bloky (PIO 0 a PIO 1) [6, s. 309].

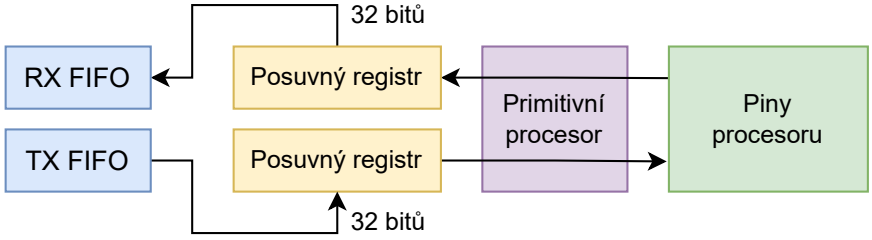


Obrázek 2.4: Blokové schéma periferie programovatelného I/O. Založeno na blokovém schématu z datasheetu RP2040 [6, s. 309].

Každý PIO blok v sobě obsahuje čtyři nezávislé „stavové automaty“, které ovládají přenos dat. Komunikovat se zbytkem systému mohou přes přijímací a vysílací fronty. Mimo to mohou automaty také vyvolat přerušení na některém z ARM jader. [6, s. 310]

Vnitřní strukturu automatů naznačuje obrázek 2.5. Každá jednotka sestává z přijímacího posuvného registru, vysílacího posuvného registru a z primitivního procesoru, který řídí přenosy dat mezi I/O frontami, posuvnými registry a obecnými vstupně-výstupními piny procesoru. [6, s. 310]

Programy pro tyto periferie je možné psát ve speciálním textovém jazyce, který se pak přeloží do instrukčního kódu [6, s. 311]. Ukázky užitečných programů je možné nalézt v dokumentaci RP2040. Pomocí dvou stavových automatů jde například implementovat obousměrný UART [6, s. 348], pomocí jednoho automatu lze implementovat I^2C [6, s. 359] nebo SPI [6, s. 342].



Obrázek 2.5: Schéma jednoho stavového automatu. Založeno na blokovém schématu z datasheetu RP2040 [6, s. 310].

2.1.3 Software

Řídící software Open-Cube je v současnosti založený na MicroPythonu – interpreteru jazyka Python pro různé vestavné systémy [7, s. 3]. Pro něj již skupina studentů vytvořila základní ovladače [8]. Umožňují například ovládání motorů, čtení analogových NXT senzorů a snímání stavu tlačítek. Další student vytvořil menu program, pomocí kterého lze kostku jednoduše ovládat.

Tato práce se proto bude zabývat rozšířením MicroPythonu o knihovny, které umožní a zjednoduší práci s EV3 motory a senzory.

Procesor RP2040 je možné programovat také v C/C++ pomocí oficiálního Pico SDK [9]. Nad ním je postavená i výše zmíněná podpora v MicroPythonu. Toto SDK bylo využito při vytváření části knihoven.

2.2 MicroPython

MicroPython do určité míry plní roli operačního systému Open-Cube. Poskytuje mimo jiné tyto základní služby:

- **Přístup přes terminál.** MicroPython vytváří na USB virtuální sériový port, na kterém poskytuje běžnou Python konzoli [7, s. 7]. Přes ni je možné řídící kostku ovládat a nahrávat na ni programy.
- **Systémové ovladače.** Uživatelé MicroPythonu nemusí znát nízkoúrovňové detaily použitého mikrořadiče. Pro mnoho periferií existují obecná rozhraní, například sériovou linku abstrahuje třída `machine.UART` [10].
- **Souborový systém.** MicroPython dokáže systémovou Flash paměť rozdělit na dvě části. V první se nachází neměnný kód (dále též nazývaný jako firmware). Nad druhou oblastí dokáže vytvořit souborový systém. Sem je pak možné dynamicky ukládat uživatelské programy. [7, s. 12]

2.2.1 Možnosti rozšíření

Při rozšiřování MicroPythonu o další knihovny lze zvolit dva přístupy. [7, s. 3] [11]

- První možností je napsat tyto knihovny v Pythonu a nahrát je do souborového systému v Open-Cube. Kód se tak nepřímo stává součástí uživatelské aplikace. [7, s. 3]

  Výhodou tohoto přístupu je, že MicroPython firmware zůstává nezměněný. Díky tomu lze na Open-Cube použít standardní firmware pro Raspberry Pi Pico. Nevýhodou je omezení programovacího jazyka na Python a případně jednoduché C [12].

  Tuto variantu dříve využívaly všechny ovladače specifické pro Open-Cube.

- Alternativou je zabudovat knihovny do MicroPython firmwaru. Tento přístup by umožnil využití plnohodnotného C a přinesl by větší volnost při tvorbě knihoven. [11][12]

  U této možnosti by bylo nutné najít způsob, jak sestavit MicroPython firmware i s novými knihovnami [11]. Výsledný firmware by následně nahradil ten standardní pro Raspberry Pi Pico.

Při vývoji byl nejprve použit první přístup. Ukázalo se ale, že Python není pro některé operace dostatečně rychlý. Knihovny proto byly přepsány do C s použitím druhého přístupu. K tomu ale bylo nutné zjistit, jak lze MicroPython takto rozšířit.

2.2.2 Vnitřní struktura

MicroPython je už v základu poměrně modulární. Na základě zdroje [13] lze jeho zdrojové kódy rozdělit přibližně takto:

- Společné jádro: interpreter jazyka Python a standardní knihovna.
- Pomocné knihovny nezávislé na platformě, na které MicroPython běží.
- Knihovny specifické pro cílovou platformu, též nazývanou jako *port*.

MicroPython firmware lze dále rozšířit pomocí *uživatelských modulů*. Pomocí nich lze vytvářet vlastní třídy a funkce viditelné v Pythonu, ale implementované v C nebo jiném jazyce. [11]

Výhoda uživatelských modulů oproti možnostem výše spočívá v místě uložení modulů. Jejich kód se nemusí nacházet uvnitř repozitáře MicroPythonu. Místo toho mohou být uložené kdekoliv v souborovém systému počítače, plná cesta k nim se předá až skriptům pro sestavení MicroPythonu. [11]

Uživatelské moduly se zabudovávají do firmware během jeho sestavení. Mimo to ale MicroPython podporuje i načítání nativního kódu za běhu. Tento mechanismus má ale určitá omezení. Uvnitř dynamicky načtené nativní knihovny je dostupná pouze část vnitřních MicroPython funkcí. [12]

U dynamických modulů jsem se také obával problémů s kompatibilitou s RP2040 SDK. Součástí SDK je poměrně složitý sestavovací systém [9, s. 9] a do něj by bylo nutné integrovat nástroje pro vytváření dynamických modulů [12]. Tomu jsem se chtěl vyhnout, proto se práce dále zaměřuje na standardní moduly.

2.2.3 Sestavení pro RP2040

Ze struktury MicroPythonu vyplývá i způsob jeho sestavování. Vždy je nutné sestavit konkrétní port – v případě Open-Cube port `rp2`. Logika uvnitř portu zařídí, že se spolu propojí vše potřebné. [11]

Výstupem procesu je obvykle firmware ve spustitelné podobě. Pro RP2040 vzniknou dva soubory: ELF a UF2. Oba obsahují stejný kód, liší se ale použitím: [7, s. 5]

- UF2 soubor lze využít k jednoduchému nahrání MicroPythonu do Flash paměti. Jedná se o nativní formát podporovaný RP2040 boot ROM. Po připojení Open-Cube v boot režimu stačí tento soubor přetáhnout na zavaděčem emulovaný USB disk. [7, s. 5]
- ELF soubor se využije při ladění MicroPythonu přes debugger. [7, s. 5]

Každý z portů je postavený nad určitým sestavovacím systémem. Pomocí jejich jazyků lze popsat, jaké soubory (a jak) se musí zkompilovat. MicroPython v současnosti podporuje dva systémy: Make nebo CMake [11]. Port pro RP2040 používá CMake [7, s. 5].

Konkrétní postup, jak lze MicroPython sestavit, se liší mezi porty. Pro RP2040 je možné návod nalézt v dokumentaci od Raspberry Pi [7, s. 4].

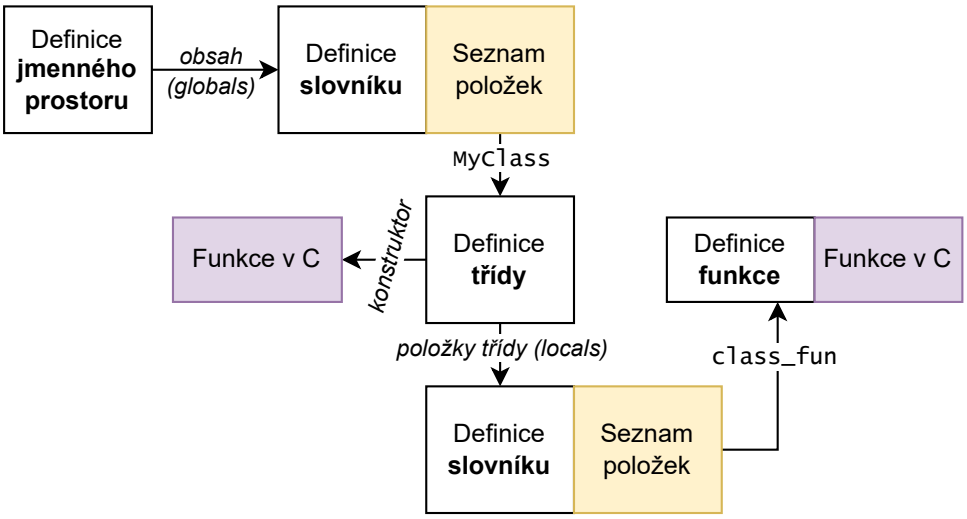
2.2.4 Nativní uživatelské moduly

Uživatelský modul je možné definovat jako skupinu C souborů, která spolu tvoří knihovnu a zabuduje se dovnitř MicroPythonu. [11]

Pro vytvoření uživatelského modulu je potřeba napsat CMake nebo Make popis toho, z jakých souborů modul sestává. Toto je poměrně přímočaré a existují k tomu ukázkové skripty. Ve zdrojových kódech MicroPythonu je k dispozici ukázkový modul [14], ve vývojářské dokumentaci je pak tato ukázka rozebraná [11].

Uvnitř modulu může být definován jeden nebo více jmenných prostorů, které si lze v uživ. programu vyžádat standardní konstrukcí `import module_name`.

Na obrázku 2.6 je naznačený ukázkový jmenný prostor. Uvnitř něj je definovaná jediná třída `MyClass`, která má konstruktor a jednu metodu `class_fun`.



Obrázek 2.6: Příklad struktury jmenného prostoru definovaného v C.

Pro popis struktury lze použít předpřipravená C makra. Například třídu je možné definovat voláním `MP_DEFINE_CONST_OBJ_TYPE()`. Výsledný jmenný prostor lze v MicroPythonu zaregistrovat makrem `MP_REGISTER_MODULE()`, které jej učiní viditelným pro Python kód. [14]

Plný seznam maker lze nalézt ve zdrojových kódech MicroPythonu v souboru `py/obj.h` [15]. Podrobnější návod na vytvoření jmenného prostoru lze najít v dokumentaci MicroPythonu [11][14] a také na stránce od nezávislého přispěvatele [16].

2.3 Senzory a motory EV3

Přehled periferií dostupných ve stavebnici EV3 ukazuje obrázek 2.7.

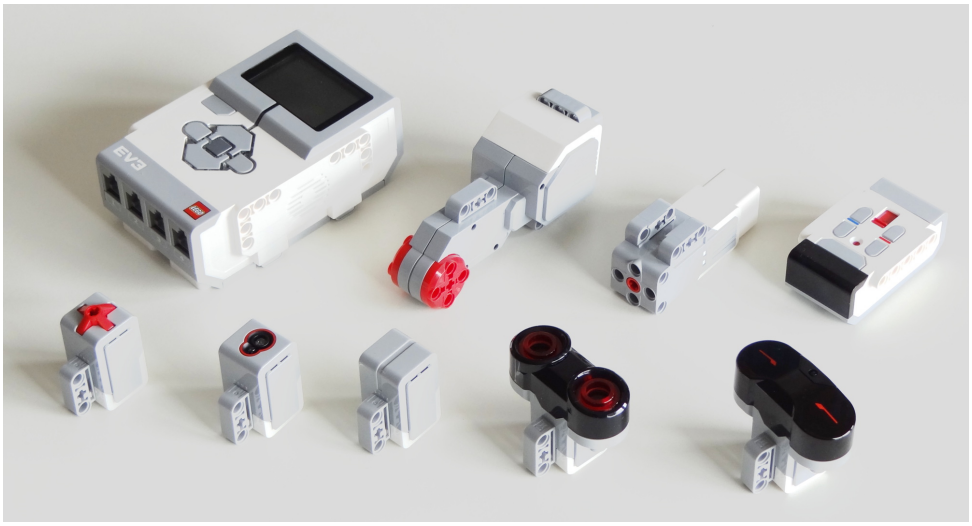
Senzory v setu je možné rozdělit do dvou kategorií – analogové a digitální. Analogový přenos dat používá pouze senzor dotyku („tlačítko“). Zbylé senzory komunikují s řídící kostkou digitálně. Jedná se o ultrazvukový senzor, infračervený senzor, gyroskop a senzor barev. [17]

Sada poskytuje také dva typy motorů – velký a střední. Oba mají jednotné rozhraní a liší se pouze svým vnitřním provedením. [17]

K propojení senzorů a motorů s řídící kostkou používá LEGO vlastní kabely s upravenou koncovkou RJ12. Na konektor je vyvedeno napájení a potřebná

komunikační rozhraní. Podrobnější informace o konektoru jsou dostupné v hardwarové dokumentaci EV3 [17].

-

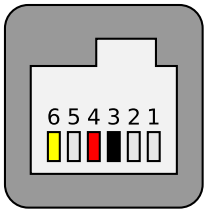


Obrázek 2.7: Senzory a motory stavebnice EV3 [18].

2.3.1 Senzor dotyku

Senzor dotyku je nejjednodušším senzorem ve stavebnici. Neobsahuje žádnou elektroniku, obsahuje pouze jeden spínač a několik rezistorů. [17]

Zapojení portu tlačítka je naznačené na obrázku 2.8. Z něj vyplývá způsob, jak řídící kostka může zjistit stav tlačítka. Stisknuté tlačítko vynutí na pinu 6 vysokou logickou úroveň. Při jeho uvolnění bude pin 6 na tlačítku odpojený. V EV3 kostce je ale na pinu 6 slabý pull-down rezistor, díky kterému zde kostka přečte nízkou logickou úroveň. [17]



Pin 3: Zem

Pin 4: Napájení 5 V

Pin 6: Podle stavu tlačítka:
- stisknuto: 5 V přes 2,2 kΩ
- odepnuto: vysoká impedance (v praxi 0 V kvůli pull-down rezistoru na straně kostky)

Obrázek 2.8: Funkce pinů v konektoru dotykového senzoru. Konektor má stejné pořadí pinů na kostce i na senzoru, schéma je tedy pro obě strany stejné.

2.3.2 Digitální senzory

Zbývající senzory jsou komplexnější. Každý z nich obsahuje mikrořadič řady STM8S, který realizuje většinu funkcí senzoru. Komunikace mezi senzorem a kostkou probíhá po pinech 5 a 6 pomocí protokolu UART, jak ilustruje obrázek 2.9. [17]

Obrázek 2.9: Funkce pinů v konektoru UART senzoru.

Nad tímto kanálem LEGO používá svůj vlastní protokol vyšší úrovně. Ten je společný pro všechny digitální EV3 senzory [17]. Protokol nejspíše nemá oficiální název; lze jej ale dohledat jako „EV3 UART Sensor Protocol“ [19].

Účelem protokolu je zvýšit univerzálnost senzorů. Každý z LEGO senzorů poskytuje několik měřících režimů – u senzoru barev se jedná například o režimy měření odrazivostni nebo okolního světla. EV3 UART protokol jednotně definuje, jak probíhá výměna naměřených dat a jak lze senzor přepnout na jiný měřící režim. [17, s. 8]

Protokol také definuje způsob, jakým senzor dokáže řídící kostku informovat o jeho vlastnostech. Mezi ty patří typ senzoru, podporované režimy měření, přenosové rychlosti a formát naměřených dat. Na základě těchto metadat dokáže řídící jednotka spolupracovat i se senzorem, který nebyl vyrobený firmou LEGO. [17, s. 8]

Průběh komunikace se senzorem lze rozdělit na dvě hlavní fáze: inicializační a datovou. Po svém připojení začne senzor opakovaně posílat řídící kostce metadata o sobě. Přenos dat se zahájí až ve chvíli, kdy kostka senzoru potvrdí, že metadata úspěšně přijala. [17, s. 8]

Protokol obsahuje také pojistku proti kolapsu komunikace. V datové fázi musí řídící jednotka minimálně každých 300 ms poslat senzoru obnovovací zprávu. Pokud ji senzor z libovolného důvodu včas nepřijme, dohlížecí obvod v mikrořadiči způsobí jeho restart. [17, s. 8]

Oficiální dokumentaci protokolu je možné nalézt v hardwarové dokumentaci EV3 [17]. Ta obsahuje ale pouze vysokoúrovňový popis protokolu. Podrobnější popis existuje v dokumentaci alternativního firmware leJOS EV3 [19] nebo ve zdrojových kódech továrního firmware EV3 kostky [20]. Další informace je možné získat z implementace protokolu na straně senzoru, která byla zdokumentována v práci [21].

2.3.3 Motory

Poslední nepopsanou periferií jsou servomotory. Z pohledu komunikace se nacházejí na pomezí mezi digitálním a analogovým přenosem dat.

Ovládání motoru je řešené přímočaře. Uvnitř krytu se nachází stejnosměrný motor [22], jehož vinutí je připojené na piny 1 a 2 příslušného portu [17].

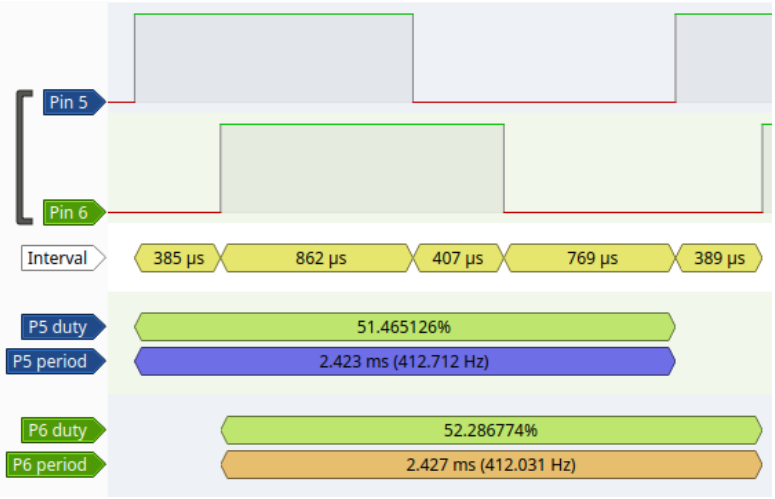
LEGO motory poskytují řídící kostce také informace o jejich otáčení. Na pinech 5 a 6 jsou vyvedené výstupy polohového čidla, viz obr. 2.10. [17]

Obrázek 2.10: Funkce pinů v konektoru motoru.

Signály čidla odpovídají výstupům kvadraturního inkrementálního snímače. Ideálním výstupem jsou dva obdélníkové signály fázově posunuté o 90°. Z jejich průběhu lze určit rychlost a směr otáčení motoru. Počítáním hran signálu lze také určit relativní polohu motoru vůči referenční pozici.

Pro ověření vlastností snímače byly pomocí logického analyzátoru změřeny průběh signálů. Obrázek 2.11 ukazuje výřez průběhů z velkého EV3 motoru, který se otáčel poměrně vysokou rychlostí.

Rozlišení polohového čidla v EV3 motorech je poměrně nízké – na jednu otočku připadá pouze 180 náběžných hran jednoho signálu. Toto potvrzuje i dekódování signálu v EV3 firmwaru [20]. Zároveň čidlo nemá oba výstupy fázově posunuté o přesně 90°, což může komplikovat výpočet rychlosti motoru.



Obrázek 2.11: Průběh signálů z polohového snímače.

Kapitola 3

Podpora EV3 senzorů

V této kapitole bude popsán postup, jakým byla do Open-Cube přidána podpora pro senzory ze stavebnice EV3.

Úlohu lze rozdělit na tři podproblémy:

- Zprovoznění obecné UART komunikace na portech pro senzory.
- Implementace EV3 UART protokolu.
- Vytvoření uživatelsky přívětivého rozhraní pro všechny senzory.

3.1 Postup a programovací jazyk

Při programování potřebného kódu byl z mé strany zvolen specifický postup:

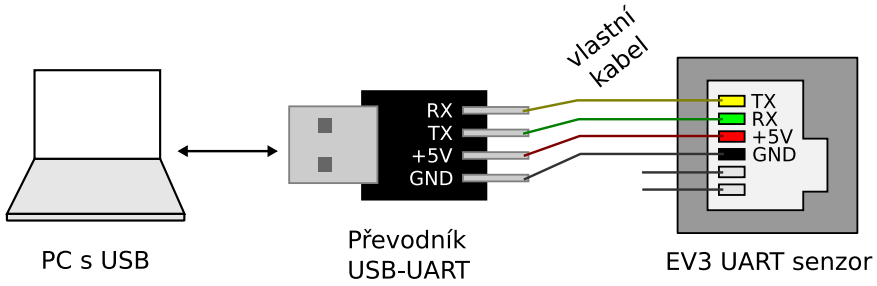
1. Nejprve byla v sekci 3.2 vytvořena aplikace pro čtení digitálních EV3 senzorů na běžném PC. Cílem bylo seznámit se s jejich protokolem a vytvořit základní kostru ovladače. Výsledný kód lze využít pro výukové a testovací účely.
2. Následně bylo v části 3.3 naprogramováno vše potřebné pro podporu senzorů na Open-Cube za použití pouze programovacího jazyka Python. Výkon ovladače ale nebyl ideální.
3. V sekci 3.4 došlo k přepsání jádra ovladače do jazyka C. Tím se podařilo dosáhnout potřebného zrychlení.

3.2 Zprovoznění na PC

Rozhraní UART je poměrně široce používané a existují pro něj převodníky do běžných počítačů. Díky tomu lze připojit digitální EV3 senzory k počítači a komunikovat s nimi přes standardní rozhraní operačního systému.

3.2.1 Propojení PC – senzor

Pro vývoj aplikace bylo použito zapojení na obrázku 3.1. Pro převod mezi UART a USB byl zvolen jeden z komerčně dostupných adaptérů. Propojení převodníku a senzoru zajišťuje upravený LEGO kabel na obrázku 3.2. Kabel vznikl rozpůlením běžného propojovacího kabelu, na volný konec pak byly umístněny dutinky pro zapojení do převodníku.



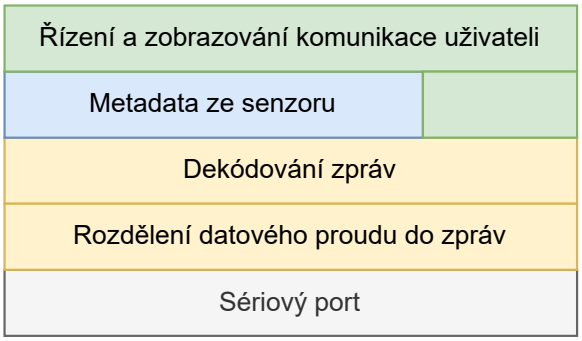
Obrázek 3.1: Schéma propojení EV3 senzoru s počítačem.

Obrázek 3.2: Praktická realizace zapojení 3.1.

3.2.2 Výsledná aplikace

Výstupem této části je konzolová aplikace v Pythonu, která obsahuje základ ovladače pro EV3 UART protokol. Jedinou funkcí aplikace je provést úvodní komunikaci se senzorem a pak začít vypisovat data, která senzor posílá.

Blokové schéma aplikace je naznačené na obrázku 3.3. Další sekce popíší funkce jednotlivých vrstev.



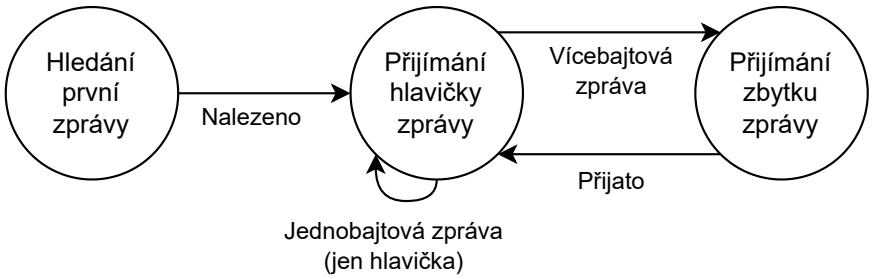
Obrázek 3.3: Blokové schéma aplikace pro komunikaci se senzory.

Sériový port

Na nejnižší vrstvě se řeší ovládání převodníku a tím umožnění komunikace na úrovni protokolu UART. Využita k tomu byla běžně dostupná Python knihovna `pySerial` [23]. Ta poskytuje přívětivější rozhraní nad nízkoúrovňovými voláními operačního systému.

Rozdělení datového proudu do zpráv

Komunikace po sériové lince probíhá na úrovni jednotlivých bajtů, EV3 protokol je ale založený na posílání vícebajtových zpráv. Tato vrstva pomocí jednoduchého stavového automatu zařídí rozdělení proudu bajtů na jednotlivé zprávy. Automat je naznačený na obrázku 3.4.



Obrázek 3.4: Stavový automat zařizující rozdělení příchozího proudu dat na jednotlivé zprávy.

Vrstva musí zajistit také „synchronizaci“ komunikace se senzorem. EV3 protokol neposkytuje žádný jedinečný oddělovací bajt, kterým by začínala každá zpráva. Místo toho má každá zpráva jednobajtovou hlavičku a z ní vyplývá celková délka zprávy. Jakmile kostka úspěšně jednu zprávu dekóduje, dokáže najít začátek následující zprávy.

Postup na dosažení synchronizace na začátku komunikace byl nalezen ve zdrojových kódech EV3 firmwaru [20]. První zpráva vyslaná senzorem má pevný formát a obsahuje číselné ID senzoru. Na začátku komunikace stačí

čekat, dokud poslední tři přijaté bajty neodpovídají této zprávě. Pokud ano, podařilo se navázat synchronizaci a aplikace může dekódovat další zprávy.

Dekódování zpráv

Cílem této vrstvy je překládat mezi binárním formátem zpráv a pomocnými datovými třídami, které napomáhají lepší čitelnosti kódu. Např. zprávu `NACK` přeloží na odpovídající bajt `0x40` a nebo naopak.

Metadata ze senzoru

Tato vrstva zajišťuje zpracování metadat, které senzor posílá po připojení. Informace využívá například pro převod naměřených hodnot do jednotek SI.

Řízení a zobrazování komunikace uživateli

Úkolem této vrstvy je propojit zbylé komponenty aplikace a poskytnout uživateli základní konzolové rozhraní. Přes něj program zobrazuje výpis zpráv, které přijal od senzoru.

Přechod mezi inicializační a datovou fází komunikace (viz sekce 2.3.2) řídí právě tato vrstva. Jakmile od senzoru přijdou veškerá metadata, vydá pokyn k odeslání potvrzovací zprávy zpět.

Dalším zodpovědností vrstvy je posílání pravidelných obnovovacích zpráv do senzoru. Bez nich by se senzor do jedné vteřiny restartoval. Správně by vrstva měla také sledovat, jestli senzor na tyto zprávy odpovídá. Verze pro počítač to neprovádí, pro zjednodušení to bylo implementováno až v pozdější verzi na Open-Cube.

Spouštění a možnosti využití

Program lze na Linuxových operačních systémech spustit příkazem:

```
$ python3 main.py /dev/ttyUSB0 -m <nazev_modu_senzoru>
```

Parametr `/dev/ttyUSB0` je cesta k zařízení převodníku a `nazev_modu_senzoru` je režim, do kterého má aplikace senzor po navázání komunikace přepnout. Seznam režimů a jiná metadata senzoru lze vypsat příkazem:

```
$ python3 main.py /dev/ttyUSB0 -s
```

Kód aplikace lze v budoucnu využít i jiným způsobem. Jednak může sloužit jako pomocný materiál při studiu EV3 protokolu dalšími lidmi. Další možností je přes ní testovat budoucí senzory vlastní výroby.

3.3 Zprovoznění na Open-Cube

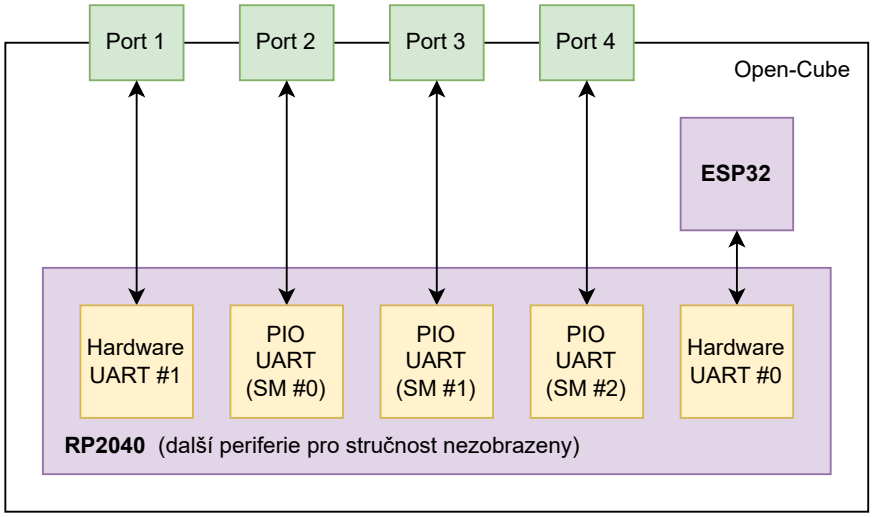
Dalším krokem bylo za pomoci programovacího jazyka Python zprovoznit EV3 senzory na Open-Cube.

3.3.1 Zapojení sériových linek

Nejprve bylo nutné zprovoznit komunikaci na portech pro senzory. Jak bylo naznačeno v sekci 2.1.2, Open-Cube na to nemá dostatečný počet hardwarových UART periferií. Proto bylo nezbytné pro některé porty využít univerzální periferii PIO.

Obrázek 3.5 ukazuje návrh zapojení. Přiřazení HW UART periferií je omezené tím, ke kterým pinům RP2040 je lze připojit. Zapojení PIO periferie lze naopak volně upravovat.

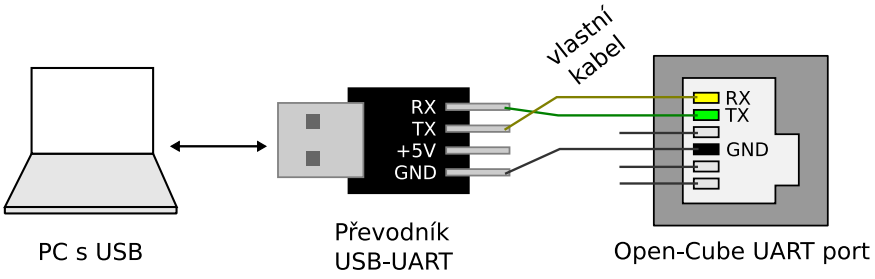
Varianta zvolená níže má za cíl ušetřit instrukční paměť bloků. „SM #0“ v obrázku naznačuje, že příjem běží na stavovém automatu 0 prvního PIO bloku a vysílání běží na témže automatu druhého bloku. Veškerý příjem tak probíhá v PIO 0 a vysílání v PIO 1.



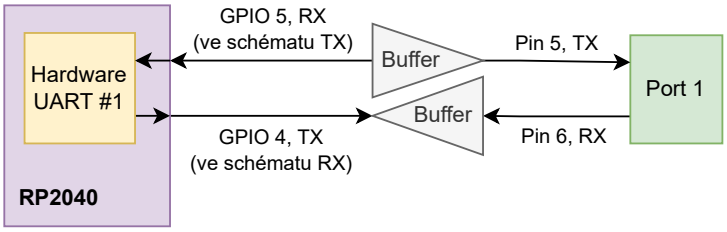
Obrázek 3.5: Zapojení sériových linek Open-Cube.

Při realizaci plánu se ale vyskytl problém. Na prvním portu se nedařilo žádná data přijmout ani odeslat. Pro testování byl opět použit převodník USB-UART, zde v zapojení na obrázku 3.6.

Později bylo zjištěno, že jde o chybu v zapojení Open-Cube. Naznačuje ji obrázek 3.7. RP2040 nedokáže namapovat příjem a vysílání hardwarové periferie tak, jak je Open-Cube zapojená. Přijímat dokáže pouze na svém GPIO pinu 5, vysílat na pinu 4 [6, s. 236]. Schéma ovšem počítalo s obráceným zapojením [4].



Obrázek 3.6: Schéma propojení Open-Cube s počítačem přes senzorový port.



Obrázek 3.7: Znázornění prohozených UART pinů na desce plošných spojů.

Závadu se podařilo vyřešit. Problém byl z mé strany konzultován s autory Open-Cube. Ti pak na použité desce provedli nezbytné prohození přijímacího a vysílacího vodiče. Poté již komunikace fungovala.

Problém bylo možné obejít i jiným způsobem. Na obrázku 3.5 má každý PIO UART dedikovaný přijímač a vysílač. Přijímače nezbytně musí být oddělené – každý senzor vysílá nezávisle na ostatních. Vysílače kostky by ale potenciálně mohly být sdílené. Díky flexibilitě PIO by bylo možné jeden vysílač přepínat mezi několika porty. Na sériové linky by se tak využilo pouze pět stavových automatů z osmi celkových. Toto řešení by ale zesložitilo ovladač.

3.3.2 Ovladač PIO UART

MicroPython neobsahuje zabudovanou podporu pro ovládání sériové linky přes periferii PIO. Musela proto být doprogramována v Pythonu.

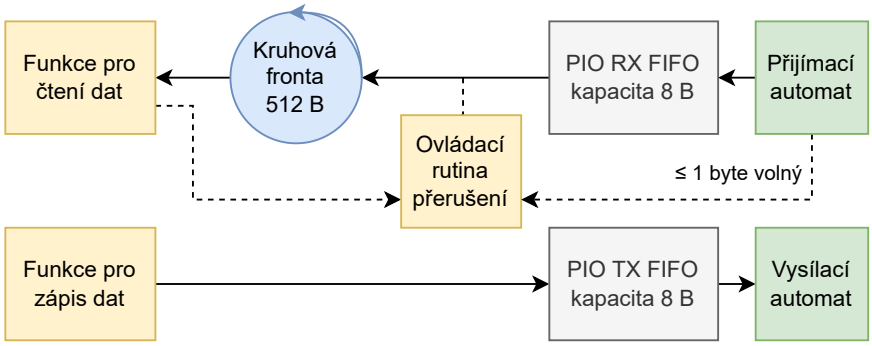
Aby mohlo PIO fungovat jako UART, jsou k tomu nezbytné dvě komponenty:

- Stavový automat, který poběží na primitivním procesoru uvnitř PIO.
- Ovladač, který bude s automatem komunikovat.

Jako základ stavového automatu byla použita ukázka z dokumentace RP2040 [6, s. 348]. Tyto automaty poskytují skoro vše potřebné pro UART – vysílací automat byl použit beze změny. Přijímací automat byl dále upraven tak, aby se snížilo množství generovaných přerušení, což je ještě popsáno níže.

Spolu s automatem bylo potřeba navrhnout způsob, jakým bude probíhat komunikace mezi PIO a ovladačem. Návrh vychází ze struktury ovladače

hardwarové periferie UART v MicroPythonu [24]. Obrázek 3.8 ilustruje zvolené řešení.



Obrázek 3.8: Tok dat frontami sériové linky realizované periferií PIO.

- Přijímací stavový automat bude data průběžně vkládat do hardwarové fronty. Jakmile se fronta naplní, vyvolá se přerušení na ARM jádře.
- Obsluha přerušení přesune data z hardwarové fronty do větší softwarové fronty. Ta slouží pro dočasné uložení dat v případě, že software na vyšších vrstvách nestíhá data zpracovávat. Zvolená velikost fronty odpovídá 88 milisekundám dat na 57600 baudech.
- Funkce pro vyčtení dat z periferie nejprve zavolá sama obsluhu přerušení. Tím dojde k přesunu dat z hardwarové do softwarové fronty. Odtud poté vyčítací funkce odebere potřebný počet bajtů.
- Funkce pro vyslání dat přímo vkládá data do hardwarové fronty, kde si je automat vyzvedne. Automat tak žádná přerušení ARM jader nevytváří. EV3 protokol potřebuje posílat pouze minimum dat z kostky do senzoru, proto lze toto zjednodušení provést.

Úprava přijímacího automatu spočívala v přidání podpory pro opožděné vyvolání přerušení. Původní automat pouze vkládal data do HW FIFO. Nově automat také vyvolá přerušení na ARM jádře v okamžiku, kdy ve frontě zbývá místo na jediný bajt. Myšlenka na tuto optimalizaci pochází z ovladače hardwarové UART periferie RP2040 [24] a v dokumentaci RP2040 [6, s. 423].

Důvodem pro tuto úpravu je omezení režie. V Pythonu musela být napsána i obsluha přerušení. To MicroPython podporuje [25], ale s poměrně velkou latencí [26]. Tato úprava zajišťuje, že se obsluha nevolá pro každý bajt zvlášť, ale naopak vyzvedne blok několika bajtů najednou. To již při rychlosti 57600 baud, kterou senzory používají, může mít podstatný vliv.

Schéma 3.8 bylo následně naprogramováno pomocí knihoven dostupných v MicroPythonu – např. třída `rp2.StateMachine` [27] poskytuje rozhraní k periferii PIO. Některé operace ale nebyly v Pythonu dostupné (např. nastavení úrovně HW FIFO, při které se vyvolá přerušení). Tyto funkce bylo možné

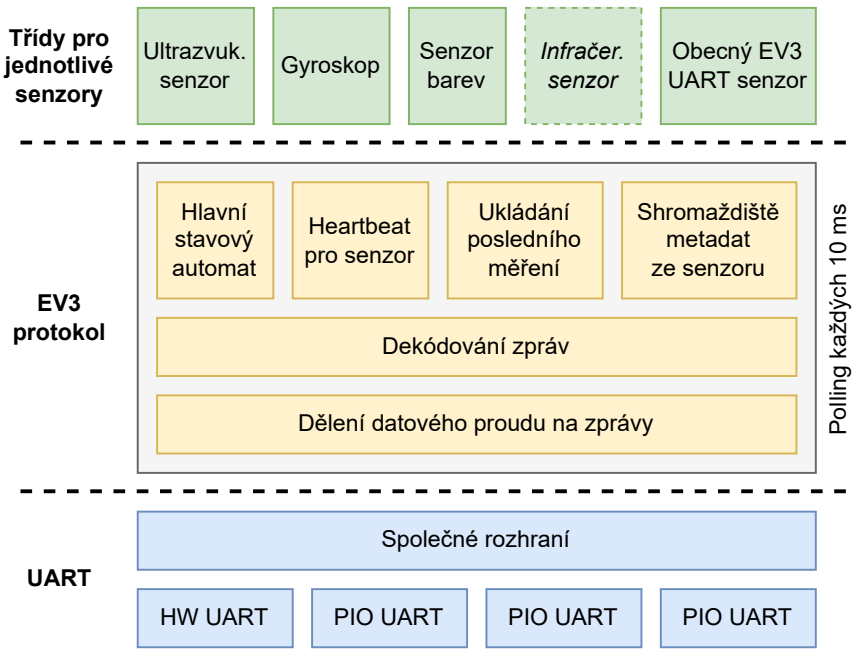
doplnit pomocí přímého přístupu do fyzické paměti, který MicroPython také podporuje [28].

Při programování byl pro další snížení režie použit režim MicroPythonu pojmenovaný jako Viper [29]. V tomto režimu MicroPython kód neinterpretuje, ale rovnou jej překládá do strojového kódu procesoru. Viper navíc s pomocí anotací dokáže některé operace převést na jednoduché instrukce procesoru (např. sečtení dvou čísel nebo přístup ukazatelem do paměti). [29]

Použitím Viperu bylo možné například zrychlit přístup k softwarové kruhové frontě. Před jeho použitím musela její implementace používat datové položky třídy. Hledání položek při každém přístupu ale bylo poměrně pomalé. Pomocí Viperu bylo možné všechna data umístit do jednoho bloku bajtů, ke kterému šlo rychle přistupovat z obsluhy přerušení i z vyčítací funkce.

3.3.3 Přenesení EV3 protokolu

Nad rozhraní pro sériovou linku již bylo možné přesunout implementaci EV3 protokolu, která byla připravena v sekci 3.2. Kód byl dále upraven a rozšířen. Výsledná struktura ovladače je nakreslená na obrázku 3.9.



Obrázek 3.9: Celková architektura Python řešení pro EV3 UART senzory.

Oproti řešení na počítači byly provedeny tyto změny:

- Kód je celkově řešený tak, aby se co nejvíce vyhýbal alokacím nových objektů. Díky tomu by se měla zmenšit frekvence „stop-the-world“ pauz, které MicroPython občas musí provést při hledání volné paměti pro nové alokace [29].
- Rozhraní pro komunikaci po sériové lince používá neblokující přístup. Díky tomu je možné komunikaci na portu pravidelně sledovat na pozadí uživatelského programu.
- Zprávy ze senzoru se zpracovávají každých 10 milisekund v přerušení časovače.
- Algoritmus již kontroluje, jestli senzor odpovídá na pravidelné obnovovací zprávy (v obrázku nazváno jako heartbeat). Pokud ne, ovladač se zresetuje a přejde opět do inicializační fáze komunikace.

Nad obecnou implementací protokolu byly vytvořeny třídy specifické pro konkrétní senzory. Ty poskytují přívětivější rozhraní pro studenty a další uživatele. Otestované jsou třídy pro EV3 ultrazvukový senzor, gyroskop a senzor barev. Třída pro infračervený senzor otestována nebyla, protože tento senzor není dostupný ve výukové sadě EV3.

Mimo tyto třídy lze také využít obecnou třídu `GenericSensor`. Ta na základě přijatých metadat dokáže spolupracovat s libovolným senzorem používajícím stejný protokol.

Pomocí tohoto řešení již bylo možné naprogramovat jednoduchého robota. Video `https://youtu.be/SnLtU2dOrws` (kopii lze nalézt v příloze) zachycuje demonstrační vozítko se sledováním čáry. Ovládání motorů je zde zařízené pomocí dřívějších ovladačů pro NXT motory.

3.3.4 Nedostatečný výkon

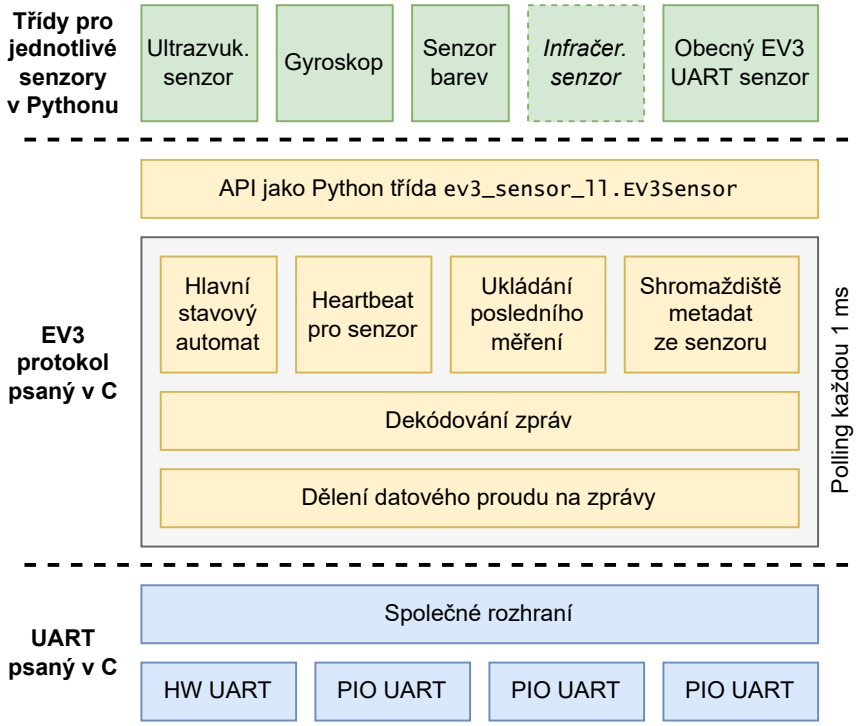
Přestože představené řešení do určité míry fungovalo, nebylo to bez kompromisů. Aby Open-Cube stíhala zpracovávat zprávy z většího počtu senzorů, musela být frekvence komunikační smyčky omezena na 100 Hz. Senzory přitom data mohou posílat až 1000krát za sekundu [17, s. 8]; přebytečné zprávy musí smyčka přeskakovat.

Opatření bylo provedeno kvůli zahlcení kostky zprávami. Když zpracování zpráv probíhalo častěji a bez jejich přeskakování, kostka při připojení více senzorů již hůře stíhala vykonávat uživatelský program.

Omezení na 100 Hz by přitom mohlo působit problémy při vytváření soutěžních robotů. Proto bylo dále jádro ovladače přepsáno do jazyka C. Další pokusy o optimalizaci zpracování zpráv v Pythonu nebyly úspěšné.

3.4 Přepis do C

Podpora sériové linky a implementace EV3 protokolu byly přepsány do podoby nativních uživatelských modulů (viz sekce 2.2.4). Výsledná architektura ovladače je znázorněná na obrázku 3.10.



Obrázek 3.10: Celková architektura nativního řešení pro EV3 UART senzory.

Vrstvy pro UART a pro EV3 protokol byly umístěny do oddělených uživatelských modulů. Dohromady poskytují jeden jmenný prostor `ev3_sensors_ll`, který zpřístupňuje jádro EV3 protokolu do Pythonu.

V této variantě se již pro interakci s hardwarem používá přímo RP2040 SDK [9]. Kód je ale převážně strukturován tak, aby jeho vyšší vrstvy bylo možné využít i na jiných platformách.

Komunikační smyčka se nyní spouští každou milisekundu. Open-Cube tak dokáže od senzorů přebírat data stejně rychle jako kostka EV3 [17, s. 8].

3.4.1 Zhodnocení

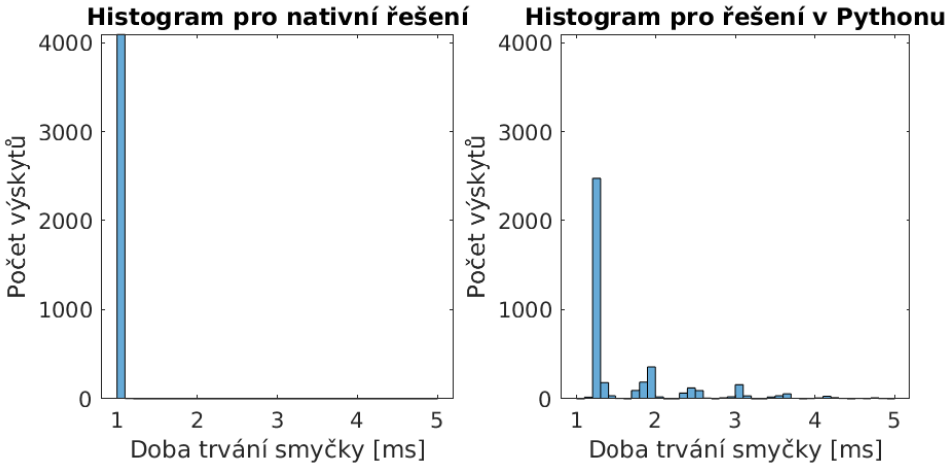
Pro otestování vlivu přepsání ovladače z Pythonu do C byl navržen jednoduchý experiment. V něm se měří doba trvání iterace jednoduché smyčky. K Open-Cube byl současně připojen gyroskop a senzor barev, aby ovladač na pozadí smyčky musel přijímat data.

Měření ideově odpovídá následujícímu kódu. Ten má simulovat běh regulační smyčky programu na frekvenci 1 kHz.

```
last_time = utime.ticks_us()
for index in range(4096):
    utime.sleep_us(1000)
    current_time = utime.ticks_us()
    data[index] = utime.ticks_diff(current_time, last_time)
    last_time = current_time
```

Aby oba jazyky měly srovnatelné podmínky, byla u obou variant nastavena frekvence komunikačních smyček na 1 kHz a bylo deaktivováno přeskakování zpráv.

Z časových intervalů mezi jednotlivými běhy smyčky byl následně vytvořen histogram. Výsledný graf pro obě varianty je na obrázku 3.11.



Obrázek 3.11: Histogram trvání jedné iterace smyčky pro Python i C verzi.

Je patrné, že rozptyl trvání iterací je výrazně větší u řešení v Pythonu. Směrodatná odchylka pro řešení v Pythonu je 1,1 ms, pro nativní řešení jen 6,8 µs. Přepsání jádra ovladače do C tedy skutečně zlepšilo jeho chování – hlavní smyčky uživatelských programů budou s novou verzí mít konzistentnější časování.

3.5 Finální Python API

Třídy určené pro uživatele byly zpřístupněny ve jmenném prostoru `lib.ev3`. Kde to bylo výhodné, tam se rozhraní inspiruje prostředím Pybricks [30] které umožňuje v Pythonu programovat kostky EV3. To by mělo zjednodušit přechod z řídících jednotek EV3 na Open-Cube.

3.5.1 Společné metody

Třídy pro digitální EV3 senzory (ultrazvukový, infračervený, gyroskop a senzor barev) sdílejí část svého rozhraní.

- Konstruktor `__init__(port, wait_ms=None)` vytvoří nový objekt senzoru navázaný k danému portu.

  Povinný argument `port` vyžaduje zadání čísla portu. Přípustné hodnoty jsou Open-Cube konstanty `Port.S1` až `Port.S4`.

  Nepovinný parametr `wait_ms` umožňuje uživateli vyčkat na inicializaci senzoru. Pokud senzor nezačne do dané doby komunikovat, funkce vyvolá výjimku `SensorNotReadyError`.

  V jeden okamžik může být na jednom portu aktivní pouze jeden objekt senzoru. Při pokusu o vytvoření druhého objektu se vyvolá výjimka `PortAlreadyOpenError`.

- Metoda `close()` deaktivuje objekt senzoru a uvolní tak port pro použití jiným objektem.

- Metoda `is_connected()` vrátí pravdivou hodnotu jen tehdy, pokud kostka detekovala libovolný EV3 UART senzor.

- Metoda `is_ready()` zjistí, jestli senzor již posílá data.

- Metoda `wait_for_connection(wait_ms=None)` počká daný čas (ve výchozím nastavení 3 s) na to, aby proběhla inicializace senzoru. Pokud se do té doby spojení nenaváže, vyvolá se výjimka `SensorNotReadyError`.

- Metoda `start_reset()` vyvolá restart senzoru. Ten je u některých revizí gyroskopu nutný k vyvolání rekalibrace, pokud začne měřený úhel driftovat od skutečné hodnoty [31].

3.5.2 Senzor barev

Třída `lib.ev3.ColorSensor` poskytuje tyto metody navíc:

- Metoda `reflection()` změří v % odrazivost povrchu pro červené světlo. Kalibrace hodnot je nastavená v senzoru a nelze ji jednoduše změnit [21].

- Metoda `reflection_raw()` změří také odrazivost, ale využije k tomu nekalibrovaný režim senzoru. Touto funkcí lze dosáhnout lepšího rozlišení za cenu toho, že uživatel musí provést vlastní kalibraci hodnot.

- Metoda `ambient()` určí intenzitu okolního světla v procentech.

- Metoda `rgb_raw()` změří odrazivost povrchu pro červenou, zelenou a modrou barvu. Funkce vrací nekalibrovanou trojici `(r,g,b)`.

- Metoda `rgb()` provede měření podle `rgb_raw()` a výsledek pak na OpenCube přepočítá na procenta pomocí vlastních kalibračních hodnot. Výstupem je trojice `(r,g,b)` obsahující odrazivosti v procentech.

- Atribut `rgb_calibration` obsahuje kalibrační parametry výstupu senzoru v RGB módu. Výchozí kalibrační konstanty byly určeny na základě maximálních naměřených hodnot na jednom senzoru.

  Každá barevná složka má dvě kalibrační konstanty – `x_offset` a `x_scale` (`x` může být `r`, `g` nebo `b`). Přepočet na procenta proběhne podle rovnice

$$x_{\text{pct}} = (x_{\text{raw}} - x_{\text{offset}}) \cdot x_{\text{scale}}. \tag{3.1}$$

- Metoda `hsv()` přepočítá výstup funkce `rgb()` do barevného prostoru HSV. Výstupem je trojice `(h,s,v)`, kde odstín `h` je ve stupních a sytost `s` a jas `v` jsou v procentech. Pro přepočet se využívají vztahy odvozené ve zdroji [32].

- Metoda `color()` přepne senzor do režimu určování barvy LEGO kostičky a vrátí výsledek tohoto měření. Pokud se barvu nepodařilo identifikovat, vrátí funkce `None`. Jinak vrátí jedno z čísel barvy z `lib.ev3.Color`: `BLACK`, `BLUE`, `GREEN`, `YELLOW`, `RED`, `WHITE`, nebo `BROWN`.

3.5.3 Gyroskop

Třída `lib.ev3.GyroSensor` rozšiřuje rozhraní takto:

- Metoda `angle()` vrátí relativní úhlovou polohu senzoru.

- Metoda `speed()` změří úhlovou rychlost senzoru ve stupních za sekundu. Měřitelný rozsah rychlostí je ±440 °/s [33].

- Metoda `angle_and_speed()` vrátí dvojici `(angle,speed)` obsahující současnou úhlovou polohu a rychlost senzoru. Tato metoda je výhodnější než kombinace `angle()`+`speed()`, protože při jejím použití se nemusí pravidelně přepínat režim senzoru.

- Metoda `coarse_speed()` změří úhlovou rychlost senzoru s větším rozsahem. Podle [33] je rozsah hodnot v tomto režimu senzoru přibližně ±1400 °/s.

- Metoda `tilt_speed()` změří rychlost otáčení senzoru podél druhé osy ve stupních za sekundu. Tento režim funguje pouze na některých revizích gyroskopu – ne všechny obsahují dvouosé snímače [33].

- Metoda `tilt_angle()` vrátí relativní úhlovou polohu senzoru podél druhé osy. Tento režim je opět přítomný jen u některých senzorů [33].

- Metoda `reset_angle(angle=0)` umožňuje nastavit úhlový offset pro funkce `angle()`, `angle_and_speed()` a `tilt_angle()`. Metodu je potřeba zavolat až po těchto funkcích. Po jejím provedení začne měření příslušného úhlu vracet hodnotu `angle`. Přepnutí senzoru do jiného režimu offset resetuje.

3.5.4 Infračervený senzor

Třída `lib.ev3.InfraredSensor` zpřístupňuje infračervený senzor.

- Metoda `distance()` změří v procentech relativní vzdálenost senzoru od nejbližšího povrchu [30].
- Metoda `seeker(channel)` vrací dvojici `(heading, distance)` obsahující směr a vzdálenost k dálkovému ovladači na daném kanálu [33].
- Čtení tlačítek ovladače tato třída zatím nepodporuje.

3.5.5 Ultrazvukový senzor

Podporu ultrazvukového senzoru zajišťuje třída `lib.ev3.UltrasonicSensor`.

- Metoda `distance(silent=False)` změří vzdálenost senzoru od nejbližší překážky v milimetrech. Pokud je argument `silent` pravdivý, senzor provede měření pouze jednou a pak se uspí.
- Metoda `presence()` umožňuje zdetekovat, zda-li je v okolí senzoru aktivní ještě jiný ultrazvukový senzor.

3.5.6 Senzor dotyku

Třída `lib.ev3.TouchSensor` zpřístupňuje senzor dotyku. Dostupné jsou pouze tyto metody:

- Konstruktor `__init__(port)` slouží k inicializaci této třídy. Povinný argument `port` akceptuje jednu z konstant `Port.S1` až `Port.S4`.
- Metoda `pressed()` zjistí, zda-li je tlačítko stisklé. Pokud ano, vrátí `True`. Kontrola probíhá prostým přečtením příslušného GPIO pinu.

3.5.7 Nízkoúrovňové rozhraní pro UART senzory

Pro ovládání digitálních senzorů lze využít také obecnou třídu `lib.ev3.EV3UartSensor`. Nad ní jsou postavené i třídy standardních senzorů.

K dispozici jsou kromě metod ze sekce 3.5.1 také tyto metody:

- Konstruktor `__init__(port, wait_ms=None, proper_id=None)` přijímá dodatečný argument `proper_id`. Ten se využívá ke kontrole, že je k řídící kostce připojený správný senzor. Pokud v úvodu komunikace senzor pošle jiné ID, vyvolá se výjimka `SensorMismatchError`.
- Metodou `sensor_id()` lze získat ID senzoru, které kostka přijala při inicializaci komunikace. Pokud není připojený žádný senzor, vrací funkce `None`.
- Metoda `start_set_mode(mode)` vyšle do senzoru žádost o změnu módu senzoru. Takto lze například přepnout barevný senzor z měření odrazivosti na měření intenzity okolního osvětlení.
- Metoda `write_command(buffer)` pošle do senzoru speciální servisní rámec, jehož obsah uživatel specifikuje argumentem `buffer`. Tato metoda není určena pro běžné použití. Lze pomocí ní například vyvolat tovární kalibrační proceduru u senzoru barev [21].
- Metoda `read_raw()` vrací dvojici `(mode, data)` obsahující poslední data, která senzor poslal. Položka `mode` určuje režim senzoru, ze kterého data pocházejí. Proměnná `data` obsahuje pole hodnot, které senzor vrátil.

  Při použití této metody je nutná obezřetnost. Vrácená data jsou platná pouze do dalšího volání funkce `read_raw()` nebo `read_raw_mode()`. Poté se obsah pole změní. Díky této optimalizaci ale není potřeba vytvářet kopii dat pro každé volání.
- Metoda `read_raw_mode(mode)` zajistí, že senzor je v režimu určeném argumentem `mode`. Poté vrátí pole hodnot, které senzor naměřil.

Ve jmenném prostoru jsou dostupné také tři typy výjimek.

- `SensorNotReadyError` odpovídá situaci, že je senzor odpojený (např. při pokusu o čtení dat).
- `PortAlreadyOpenError` signalizuje, že uživatel se pokusil na jednom portu inicializovat dva různé senzory.
- `SensorMismatchError` upozorňuje, že uživatel zapojil do kostky jiný senzor, než který požadoval kód programu.

Kapitola 4

Podpora EV3 motorů

4.1 Cíle

Motory lze podporovat na různé úrovni abstrakce. Na základě srovnání s jinými prostředími pro programování EV3 (leJOS EV3, ev3dev, Pybricks) bylo navrženo toto rozdělení:

- **Abstrakce hardware kostky**. Na této úrovni lze pouze řídit, jaké střední napětí bude Open-Cube do motoru posílat. Lze také zjistit, jaká je aktuální poloha (a příp. rychlost) motoru.

  Příkladem je třída `UnregulatedMotor` z prostředí leJOS [34].

- **Regulace rychlosti nebo polohy**. Zde již software poskytuje rozhraní pro udržování stále úhlové rychlosti nebo polohy motoru. To lze zařídit pomocí zpětnovazebních regulátorů.

  V rámci týmového projektu na Open-Cube vznikly Python knihovny, které poskytují pouze regulátory a nadřazené funkce již nikoliv [8]. Příkladem obdobné funkcionality na EV3 je metoda `Motor.run(speed)` existující v Pybricks [30].

- **Inteligentní řízení rychlosti a zrychlení**. Při pohybu robota po hrací ploše může být výhodné omezit jeho rychlost a zrychlení. Jednou z možností, jak to zařídit, je vygenerovat rovnoměrně zrychlenou trajektorii mezi polohami motoru a tu pak pomocí regulátoru sledovat.

  Tyto funkce jsou v programovacích prostředích obvykle dostupné. Příkladem je funkce `BaseRegulatedMotor.rotateTo(angle)` prostředí leJOS [35] nebo funkce `Motor.run_target(speed, angle)` v Pybricks [30].

- **Řízení celého robota**. Pro dvoukolová vozítka lze rozhraní dále zobecnit na povely pro celého robota. Rozhraní pak poskytuje například funkce pro pohyb po přímce nebo po oblouku.

  Příkladem existující implementace je třída `Chassis`/`WheeledChassis` v leJOS [36] nebo třída `DriveBase` v Pybricks [37].

Open-Cube již základní ovladače pro motory obsahuje [8]. Cílené byly na NXT motory, které jsou s EV3 motory kompatibilní. Při pokusech se ale ukázalo, že jejich kód není plně spolehlivý – software kostky se při jejich použití občas zasekl.

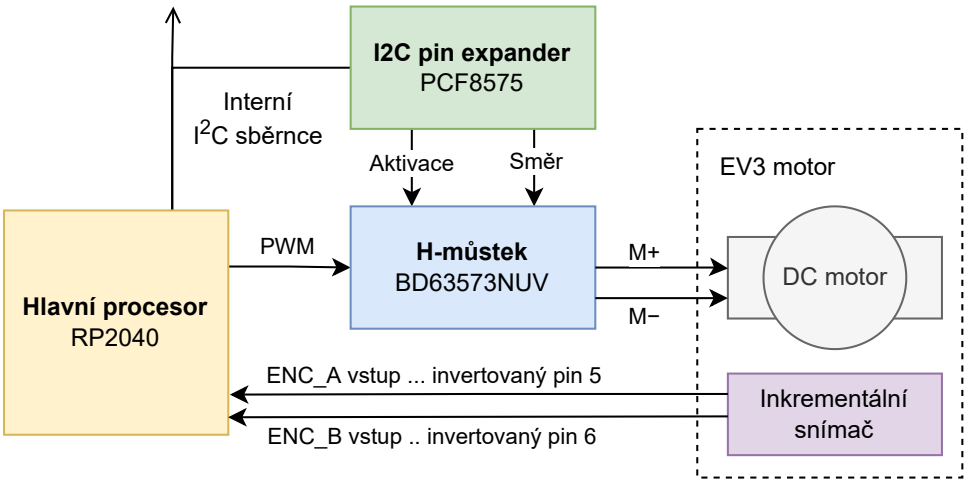
Během vývoje se také ukázalo, že by bylo vhodné aspoň nejnižší vrstvu abstrakce napsat v jazyce C. Paralelně probíhající bakalářská práce totiž cílí mimo jiné na přepis existujících knihoven do nativních modulů. Změna se dotkla i některých nízkoúrovňových komponent, které jsou pro řízení motorů nezbytné.

Z obou důvodů byla podpora pro motory naprogramována odznova.

Na základě konzultace s vedoucím bylo rozhodnuto, že pro naplnění zadání BP je nutné implementovat pouze nejnižší vrstvu abstrakce. Během přípravy testovacích robotů (sekce 6) se ale ukázalo, že program robotů by zjednodušila dostupnost některých pokročilejších funkcí. Původní řešení proto bylo rozšířeno o kód pro řízení rychlosti a zrychlení.

4.2 Abstrakce hardware kostky

Cílem této vrstvy je skrýt detaily vnitřního zapojení Open-Cube, které je naznačené na obrázku 4.1.



Obrázek 4.1: Schéma připojení jednoho motoru k Open-Cube. I²C pin expander je sdílený všemi porty pro motory. H-můstek má každý port vlastní.

Z obrázku 4.1 vyplývají některé kroky nutné k vytvoření vrstvy.

1. Nejprve je nutné zprovoznit I²C komunikaci s použitým pin expanderem.
2. Dále je třeba nakonfigurovat PWM periferii uvnitř RP2040.

3. Ke zjištění polohy motoru je nutné sledovat signály z čidla v motoru.

4. Pro potřeby regulace je vhodné dále určit aktuální rychlost motoru. To nemusí být triviální – enkodér má poměrně nízké rozlišení a tak jednoduché přístupy nemusí dávat dobré výsledky.

4.2.1 I²C pin expander

Pomocí I²C expandéru PCF8575 lze v Open-Cube ovládat dva vstupy použitých H-můstků [4]:

- Deaktivační pin `PS` (power-save). Pomocí něj lze ovládat, zda-li je vinutí připojené k můstku [38]. V odpojeném stavu lze motorem volně otáčet, což může být žádoucí.
- Řídící pin `INB`. Změnou jeho úrovně lze nastavit směr spínání H-můstku a tak směr otáčení motoru [38].

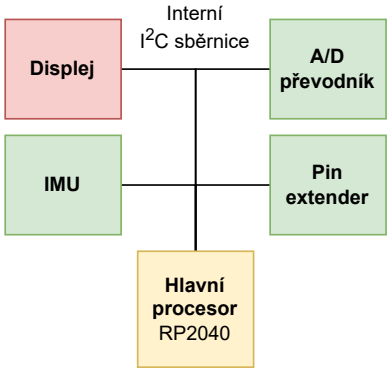
PCF8575 je v Open-Cube připojený i k jiným periferiím. Používá se pro čtení uživatelských tlačítek a na ovládání napájecího zdroje. [4]

Pro tento obvod bylo nutné využít ovladač, který používá zbytek řídícího software. Aktuální verze Open-Cube využívá ovladač od V. Jelínka, který se v paralelně běžící bakalářské práci zaměřuje na jiné části řídícího software.

Nejprve však bylo nutné vyřešit jednu komplikaci.

Konflikty při přístupu ke sběrnici

Sběrnice, ke které je expander připojený, je sdílená několika zařízeními. Zapojení naznačuje obrázek 4.2.



Obrázek 4.2: Zařízení na vnitřní sběrnici Open-Cube. Displej je problematická periferie, protože potřebuje přenášet velké bloky dat.

Sdílení způsobuje problémy se zablokováním sběrnice. Nejvíce situaci komplikuje černobílý 128×64 displej. Při přenosove rychlosti 400 kHz zabere jeho celé překreslení minimálně

$$\frac{(128 \cdot 64)\ \text{px} \cdot 1\ \text{bit/px}}{400000\ \text{bit/s}} \approx 20\ \text{ms}. \tag{4.1}$$

Kvůli tomuto mohou vzniknout při přístupu jiným k zařízení na sběrnici konflikty.

V současné době ovladač displeje Open-Cube provádí přenos dat bez zakázání přerušení [39]. Může se tedy stát, že během přenosu se vyvolá obluha přerušení, například budoucí regulační smyčka motorů. Ta se může pokusit přistoupit k PCF8575 pomocí stejné I^2C periferie procesoru. RP2040 SDK tuto situaci nijak neošetřuje [40] a tak výsledek může být neočekávaný.

V praxi bylo pozorováno, že přístup k pin expanderu během překreslování displeje způsobí vypnutí Open-Cube. Možným vysvětlením je, že po skončení přerušení se data displeje začnou posílat do expanderu a s určitou pravděpodobností tak dojde k vypnutí pinu řídícího napájecí zdroj.

Pozorovaná chyba byla z mé strany opravena pomocí synchronizačních zámků dostupných v RP2040 SDK [41]. Před přístupem ke sběrnici se každý ovladač pokusí získat zámek pro sebe. Je-li úspěšný, může I^2C přenos provést a pak zámek uvolnit. Naopak, je-li zámek obsazený, nesmí ovladač RP2040 I^2C periferii využít a musí přístup odložit na později.

4.2.2 Řízení napětí na motoru

Dále je nutné řídit vstup H-můstku `INA`. Pomocí něj lze v Open-Cube přepínat, zda je vinutí motoru připojené k napájení, nebo jen zkratované [4][38]. Pokud bude procesor tento signál pulzně-šířkově modulovat, může tak řídit střední hodnotu napětí na svorkách motoru.

Pro generování takového signálu lze na RP2040 využít dedikovaný PWM (pulse-width modulation) hardware. K dispozici je 16 výstupů z 8 nezávislých kanálů. Zde jsou potřeba pouze čtyři výstupy, které navíc mohou sdílet kanály díky shodné modulační frekvenci. [6, s. 524]

Příslušná periferie byla nastavena podle dokumentace přes RP2040 SDK.

- Modulační frekvenci byla nastavena na 20 kHz. Tato frekvence je již mimo rozsah lidského sluchu, a tak nebudou motory vydávat pískavý zvuk.
- Pro rozlišení střídy byla vybrána hodnota 1 %. Pak nejkratší logický pulz vytvořený RP2040 odpovídá nejkratšímu povolenému pulzu na vstupu H-můstku – 500 ns [38].

- Periferie se konfiguruje tak, aby vytvářela signál symetrický okolo středu jeho základní periody. Pokud je zdroj [42, s. 10] interpretován správně, mělo by to vést k nepatrnému snížení vysokofrekvenčního rušení při rychlých změnách střídy signálu.

Ovládání PWM periferie a expanderu PCF8575 pak bylo zpřístupněno přes jednotné rozhraní v C. Funkce `motor_pwm_run(port, duty)` aktivuje H-můstek a začne do motoru vysílat napětí o dané střední hodnotě. Napětí je zde relativní vůči napětí baterií Open-Cube. Metoda `motor_pwm_poweroff(port)` naopak způsobení odpojení a tak odbrzdění daného motoru.

4.2.3 Měření polohy motoru

Pro zjištění polohy motoru stačí dekódovat polohové signály popsané v sekci 2.3.3. Počítáním náběžných a sestupných hran jednoho ze signálů lze měřit okamžitou polohu motoru ve stupních.

Pro měření byl použit přístup původně implementovaný v EV3 firmwaru [20]. Při hraně signálu na pinu 5 portu pro motory se vyvolá obsluha přerušení. Funkce pak aktualizuje proměnnou obsahující polohu motoru:

- Při náběhu pinu 5 s aktivním pinem 6 se o 1° sníží.
- Při náběhu pinu 5 s neaktivním pinem 6 se o 1° zvýší.
- Při sestupu pinu 5 s aktivním pinem 6 se o 1° zvýší.
- Při sestupu pinu 5 s neaktivním pinem 6 se o 1° sníží.

Teoreticky je možné dosáhnout rozlišení polohy až půl stupně [43]. Pak procesor musel reagovat na hrany obou signálů. Fázový posun obou signálů z EV3 motoru ale není přesně 90°, viz obrázek 2.11 v sekci 2.3.3. Vzdálenost mezi dvěma kódy polohy by tak nebyla konstantní, ale měnila by se mezi cca. 1/3 a 2/3 stupně [43]. Proto tato varianta nebyla implementována.

4.2.4 Měření rychlosti motoru

Metoda měření

Pro nalezení vhodné metody byla zkonzultována odborná literatura. Přehledový článek [44] popisuje a srovnává několik přístupů:

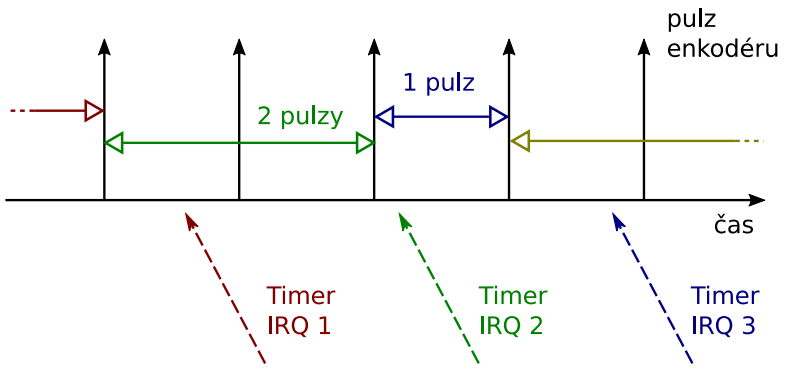
- Měření frekvence: rychlost odpovídá tomu, o kolik stupňů se motor otočil během pevného časového intervalu. Metoda je jednoduchá na implementaci, trpí ale nízkou přesností při malých rychlostech. Hodnoty je možné filtrovat, to ale přidá určité časové zpoždění. [44, s. 3]

- Měření periody je založené na měření intervalu mezi dvěma událostmi čidla. Zde by se jednalo např. o zpoždění mezi náběžnou a sestupnou hranou signálu. Metoda dává dobré výsledky při nízkých rychlostech, je ale poměrně citlivá na neideality snímače. [44, s. 4]
- Kombinovace obou metod. Tu autoři popisují také v samostatném článku [45] a krátce je popsána níže.
- Kalmanův filtr a Luenbergerův pozorovatel stavu: na základě modelu systému a příchozích měření dokáží odhadnout i neměřené stavy systému (zde právě rychlost).

Pro naprogramování do Open-Cube byla zvolena kombinovaná metoda. Ta spojuje výhody měření frekvence a měření periody [44, s. 4].

Myšlenka metody je naznačená na obrázku 4.3. Výpočet rychlosti probíhá v přerušení časovače s pevnou periodou T_s. Při každém přerušení algoritmus změří úhel, který se na snímači načítal od minulého výpočtu. Rozdíl oproti měření frekvence nastává v tom, vůči jakému časovému intervalu se počet pulzů vztáhne. U měření frekvence by šlo o periodu T_s. Kombinovaná metoda místo toho měří skutečný časový interval, za jaký se pulzy načítaly. [45, s. 3]

Metoda obsahuje i řešení případu, že se mezi přerušeními žádné pulzy nenačítaly. Výpočet rychlosti se v tom případě odloží na pozdější přerušení, které by již pulz přijalo. Po příjmu pulzu se rychlost spočítá pomocí celkového času od posledního přijatého pulzu. [45, s. 3]



Obrázek 4.3: Ilustrace zvolené metody měření rychlosti. Inspirováno obrázkem ze zdroje [45, s. 3].

Realizace

K realizaci výpočtu rychlosti pomocí této metody je nutné nejprve znát okamžik, ve kterém dorazila hrana signálu na pin 5. Za tímto účelem byla rozšířena obsluha příslušného přerušení, aby vedle polohy motoru ukládala také aktuální hodnotu systémového časovače.

Výpočet rychlosti probíhá v pravidelném přerušení každých 10 milisekund. Tato frekvence je kompromisem mezi včasnou aktualizací rychlosti a mezi jejím průměrováním při vyšších otáčkách motoru. Frekvence 100 Hz by také neměla procesor znatelně zatížit.

Výsledná rychlost se ukládá do globální proměnné, která je přes pomocnou funkci dostupná dalším knihovnám Open-Cube.

Při výpočtu se uvažují náběžné i sestupné hrany signálu. To může zanést do rychlosti rušení, protože střída signálu není přesně 50 %, viz sekce 2.3.3. Zjednodušuje to ale kód přerušení. Návrh regulátorů v sekci 4.3.4 tuto možnou komplikaci zohlednil.

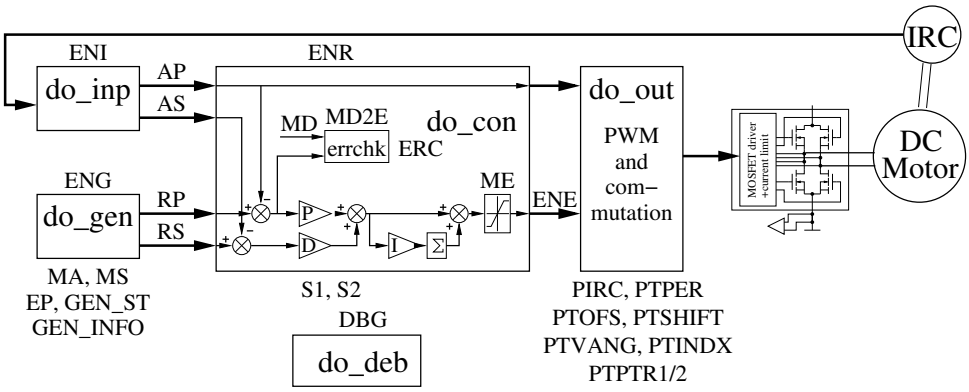
4.3 Řízení pohybu motorů

4.3.1 Celková architektura

PXMC

Návrh regulační smyčky byl inspirován knihovnou PXMC (Portable, highly eXtendable Motion Control library [46]). Ta se mimo jiné používá v robotických řídících jednotkách MARS 8 [46], které jsou nasazené u některých průmyslových robotů na fakultě [47].

Regulační smyčka PXMC při řízení stejnosměrného motoru je znázorněná na obrázku 4.4. PID regulátor v bloku `do_con` zde sleduje trajektorii generovanou blokem `do_gen`. Využívá k tomu zpětnou vazbu z bloku `do_inp`. Motor je pak ovládaný PWM signálem v bloku `do_out`. [48]



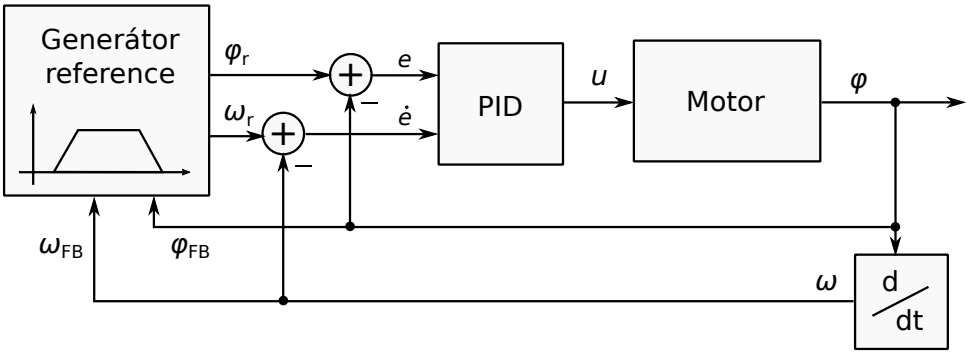
Obrázek 4.4: Blokové schéma regulátoru DC motoru v knihovně PXMC. [48]

Při vývoji bylo zvažováno, jestli pro řízení motorů nepoužít přímo tuto knihovnu. Nakonec bylo rozhodnuto proti této variantě. S knihovnou by bylo nutné se blíže seznámit, přidat do ní podporu pro RP2040 a integrovat ji do systému pro sestavování MicroPythonu. To bylo zhodnoceno z mé strany jako

náročnější, než by bylo vytvoření vlastního regulátoru specializovaného pro Open-Cube.

Návrh pro Open-Cube

Pro Open-Cube byla navržena řídící smyčka na obrázku 4.5. Generátor trajektorie vytváří referenci pro polohu motoru a zároveň analyticky určuje její derivaci. Sledování reference zajišťuje PID regulátor. Ten řídí amplitudu a polaritu napětí na vinutí motoru. Zpětná vazba se uzavírá snímáním aktuální polohy a rychlosti motoru, ze kterých se určí aktuální regulační odchylka. Stav motoru je monitorovaný také generátorem trajektorie.



Obrázek 4.5: Blokové schéma navrženého regulátoru motorů.

Pro jednodušší ladění PID regulátoru byl dále identifikován model použitých motorů. Na jeho základě byly navrženy regulační konstanty. Posledním krokem bylo naprogramovat potřebné pohyby do generátoru reference.

4.3.2 Model motoru

Model motoru vychází z popisu ve zdroji [49]. V něm autoři sestavili a identifikovali stavový model velkého EV3 motoru. Jeho vstupem je napětí na svorkách motoru, výstupem je rychlost otáčení.

Model bylo možné převést do podoby lineárního stavového modelu druhého řádu. Při jeho analýze ale bylo zjištěno, že jeden z jeho pólů je přibližně 60krát rychlejší než zbývající dominantní pól. Velký EV3 motor proto lze bez velké ztráty přesnosti modelovat také jako systém prvního řádu. Jeho přenos pak bude mít tvar

$$H_\omega(s) = \frac{g}{s\tau + 1}, \tag{4.2}$$

kde τ určuje polohu pólu a g mění stejnosměrné zesílení. Pro zjištění polohy motoru je potřeba rychlost zintegrovat, přenos se tak změní na

$$H_\varphi(s) = \frac{1}{s} \cdot \frac{g}{s\tau + 1}. \tag{4.3}$$

4.3.3 Identifikace motoru

Dalším krokem bylo identifikovat konstanty v přenosu 4.3.

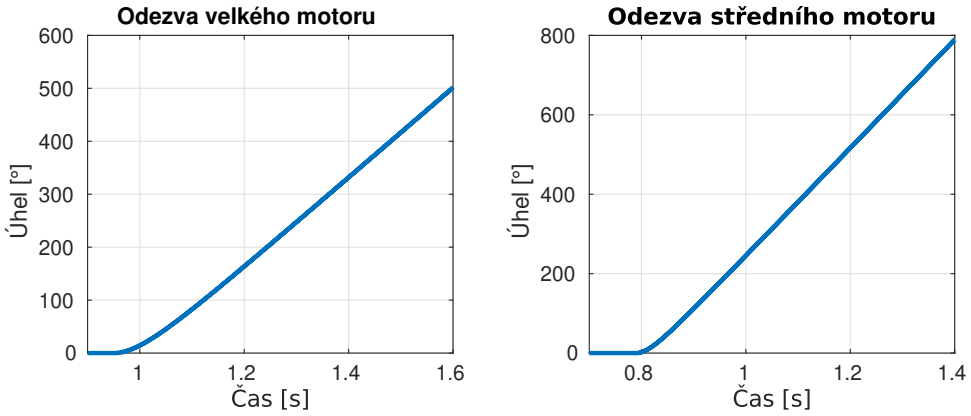
Jejich hodnoty by bylo možné odvodit z již zmíněného modelu velkého EV3 motoru ve zdroji [49]. Autoři zde ale konstanty identifikovali pouze pro velký EV3 motor, pro střední EV3 motor konstanty chybí. Bez dalších informací by tak bylo nutné navrhnout pro oba motory stejný regulátor. Nebylo ale jisté, že by výsledný systém stále dobře fungoval.

Z tohoto důvodu byla provedena nová identifikace. Postup vychází ze zdroje [49]. Na velkém i středním motoru byla naměřena odezva na skok vstupu. Daty byla poté proložena křivka, kterou by sledoval systém s přenosem 4.3.

Pro vytvoření vstupního skoku byl použit výstup Open-Cube. Z pohledu vinutí motoru se jednalo o skok z nulového napětí do napětí baterie (v danou chvíli o něco více než 7 V). Z pohledu střídy PWM se ale jednalo o skok z 0 % (= 0) do 100 % (= 1) a vůči ní byl model dále uvažován. Konkrétní konstanty modelu se tak mohou lišit podle aktuálního napětí baterie.

Měření polohy bylo provedeno pomocí logického analyzátoru. Pomocí něj bylo možné s vysokou přesností navzorkovat signály enkodéru. Převod na polohu pak byl proveden skriptem v prostředí Matlab. Výsledné průběhy jsou znázorněné na obrázku 4.6.

Vstupní napětí motoru při experimentu nebylo snímáno. V další části tak bylo nutné identifikovat také neznámé zpoždění odezvy od počátku měření.



Obrázek 4.6: Odezva velkého a středního motoru na skok vstupního napětí.

Ideální odezva systému 4.3 na skok zpožděný o čas T_D má Laplaceův obraz

$$\Phi(s) = e^{-sT_D}\frac{1}{s^2}\cdot\frac{g}{s\tau+1}. \tag{4.4}$$

Provedením inverzní Laplaceovy transformace lze získat funkci

$$\varphi_{\mathrm{id}}(t) = \mathbb{1}(t-T_D)\cdot\left[g\cdot(t-T_D) - g\tau + g\tau\exp\left(-\frac{t-T_D}{\tau}\right)\right]. \tag{4.5}$$

Pro proložení dat funkcí byla použita metoda Levenberg-Marquardt dostupná v optimalizační toolboxu prostředí Matlab. Pomocí ní byla minimalizována celková chyba proložení E nad všemi N vzorky:

$$E = \sum_{k=1}^{N} e[k]^2, \tag{4.6}$$

$$e[k] = \varphi_{\mathrm{msr}}[k] - \varphi_{\mathrm{id}}(t_{\mathrm{msr}}[k]). \tag{4.7}$$

Za optimalizační proměnné byly zvoleny neznámé g, τ a T_D v rovnici 4.5.

Tímto postupem byly pro velký motor získány konstanty

$$g \approx 845\,^\circ/\mathrm{s}, \tag{4.8}$$

$$\tau \approx 0{,}0592\,\mathrm{s}, \tag{4.9}$$

$$T_D \approx 0{,}948\,\mathrm{s}. \tag{4.10}$$

Pro střední motor bylo obdobně zjištěno

$$g \approx 1360\,^\circ/\mathrm{s}, \tag{4.11}$$

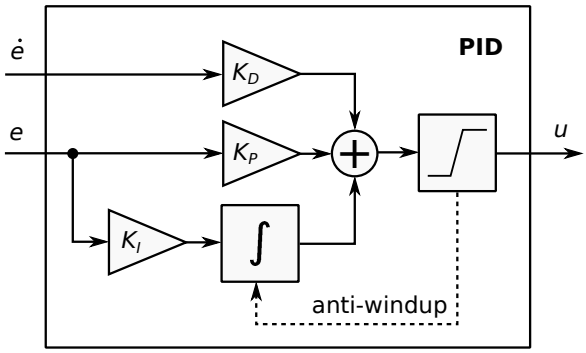
$$\tau \approx 0{,}0337\,\mathrm{s}, \tag{4.12}$$

$$T_D \approx 0{,}786\,\mathrm{s}. \tag{4.13}$$

Dosazením konstant do rovnice 4.3 vznikne matematický model obou motorů v podobě přenosové funkce.

4.3.4 Návrh PID regulátoru

Pro realizaci PID regulátoru bylo zvoleno standardní paralelní zapojení s dvěma úpravami, viz obrázek 4.7.



Obrázek 4.7: Vnitřní zapojení PID regulátoru.

První úprava spočívá v použití vnějšího signálu jako derivace. Jiná zapojení mívají derivační složku počítanou přímo z regulační odchylky e za pomocí filtrované derivace. Zde bylo, obdobně jako PXMC [46], využito rozdílu známé rychlosti motoru a plánované rychlosti na trajektorii.

Druhou úpravu představuje zapojení vnitřní saturace a „anti-windup“ obvodu. Jejich cílem je zmenšit překmit motoru, pokud by regulátor po určitou dobu požadoval příliš vysoký výkon motorů. Po tuto dobu by se na integrátoru akumulovala odchylka, která by se pak musela v překmitu od-integrovat. [50, s. 655]

Anti-windup zde byl realizován pomocí logiky. Pokud výstupní signál překročí nastavenou mez, regulátor jej omezí na přípustnou úroveň. Zároveň ale upraví stav integrátoru tak, aby nový výstupní signál celého PID regulátoru byl přesně na hranici saturace. Tím je zajištěno, že integrátor nevytvoří příliš velký překmit.

Za vzorkovací periodu regulátoru byl zvolen čas $T_\mathrm{s} = 10\,\mathrm{ms}$. S odpovídající frekvencí přichází informace o rychlosti motoru.

Hodnotu frekvence v současnosti nelze přenastavit bez změny zdrojového kódu Open-Cube. Regulátor lze ale obejít a ovládat motory z uživatelské aplikace přímo přes střídu PWM.

Konstanty regulátoru K_P, K_I a K_D byly navrženy opět pomocí prostředí Matlab. Získaný spojitý model motoru byl diskretizován metodou ZOH (zero-order hold) se vzorkovací periodou T_s. Následně byl využit „PID tuner“ [51], který dokáže na základě požadavků navrhnout příslušné konstanty. Jako požadavek byla zvolena šířka pásma otevřené smyčky ω_c.

Postupnou iterací bylo dosaženo hodnoty $\omega_\mathrm{c} = 20\,\mathrm{rad{\cdot}s^{-1}}$. Při té již byla odezva motoru dostatečně rychlá. Regulátor ale stále zvládal potlačovat šum v měřené rychlosti popsaný v sekci 4.2.4.

Po přeškálování konstant pro výstup v % střídy PWM byly pro velký motor získány konstanty

$$K_\mathrm{P} = 3{,}11, \tag{4.14}$$
$$K_\mathrm{I} = 6{,}22, \tag{4.15}$$
$$K_\mathrm{D} = 0{,}135. \tag{4.16}$$

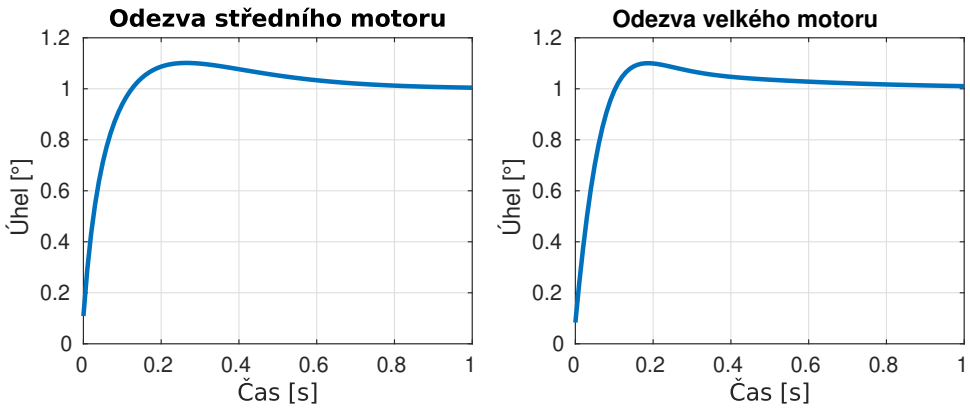
Pro střední motor stejným postupem vyšlo

$$K_\mathrm{P} = 1{,}56, \tag{4.17}$$
$$K_\mathrm{I} = 4{,}75, \tag{4.18}$$
$$K_\mathrm{D} = 0{,}0666. \tag{4.19}$$

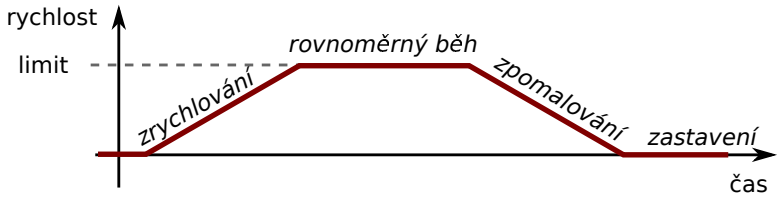
Obrázek 4.8 ukazuje odezvu modelů na jednotkový skok po zapojení regulátoru.

4.3.5 Generování trajektorie

Zbývající částí řídící smyčky je generátor referenční trajektorie. Pro potřeby Open-Cube stačí vytvářet trajektorie s lichoběžníkovým průběhem rychosti, viz obrázek 4.9. Podobné průběhy používá například leJOS EV3 [52].

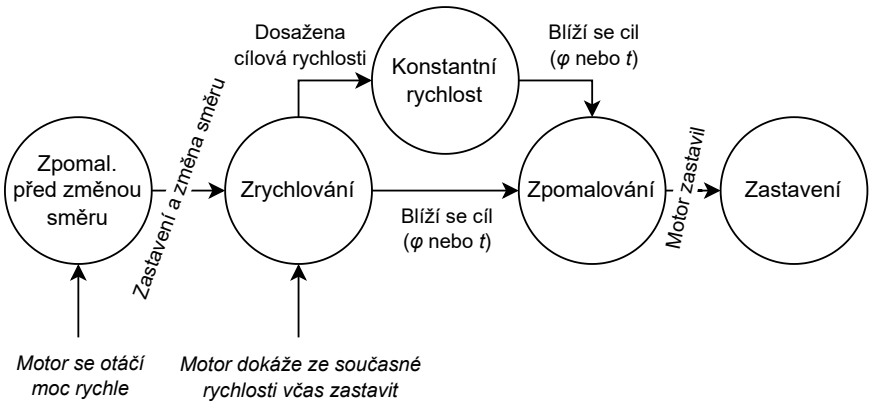


Obrázek 4.8: Odezva regulovaných modelů motorů na skok referenčního úhlu.



Obrázek 4.9: Ukázka možného profilu rychosti.

Generátor dokáže vytvářet trajektorie omezené dobou trvání pohybu nebo cílovým úhlem. Dokáže také vytvářet neomezené trajektorie (motor se bez zásahu uživatele nezastaví). Implementován byl jako stavový automat znázorněný na obrázku 4.10.



Obrázek 4.10: Stavový automat generátoru reference.

Fáze zrychlování

V této fázi motor postupně zrychluje z počáteční (ne nutně nulové) rychlosti ω_0 na cílovou rychlost ω_{target} se zrychlením a. Trajektorie je zde až na znaménka definovaná rovnicemi

$$\varphi_r(t) = \varphi_0 + \omega_0(t - t_0) + \frac{1}{2}a(t - t_0)^2, \tag{4.20}$$

$$\omega_r(t) = \omega_0 + a(t - t_0), \tag{4.21}$$

kde φ_0 a t_0 jsou rychlost motoru a čas na začátku manévru.

Tato fáze trvá pouze dobu nutnou k dosažení cílové rychlosti:

$$T_{\text{rampup}} = \left| \frac{\omega_{\text{target}} - \omega_0}{a} \right|. \tag{4.22}$$

Fáze může skončit dříve, pokud je trajektorie krátká a profil rychlosti neobsahuje fázi konstantní rychlosti. Toto se kontroluje podle úhlu nebo času (záleží na typu pohybu), za který stihne motor ze současné rychlosti zastavit. Kritérium je blíže popsáno u následující fáze.

Fáze konstantní rychlosti

Zde se motor pohybuje rovnoměrně podle rovnic

$$\varphi_r(t) = \varphi_1 + \omega_{\text{target}}(t - t_1), \tag{4.23}$$

$$\omega_r(t) = \omega_{\text{target}}, \tag{4.24}$$

kde φ_1 a t_1 jsou rychlost a čas na počátku této fáze.

Konec této fáze se vyhodnocuje podle typu omezení. Při omezení na celkový čas skončí fáze v okamžiku, kdy zbývá do konce čas

$$T_{\text{rampdown}} = \left| \frac{\omega_{\text{target}}}{a} \right|. \tag{4.25}$$

Za tento čas stihne stihne motor se zrychlením $-a$ zastavit.

Při omezení na celkový úhel se fáze ukončí v momentě, kdy motoru v manévru zbývá úhlová vzdálenost

$$\varphi_{\text{rampdown}} = \left| \frac{\omega_{\text{target}}^2}{2a} \right|. \tag{4.26}$$

Pak zpomalující motor se zrychlením $-a$ zastaví právě na cílovém úhlu.

Pro trajektorie bez omezení je tato fáze konečná.

Fáze zpomalování

Při této fázi motor zpomaluje podle rovnic

$$\varphi_r(t) = \varphi_2 + \omega_2(t - t_2) - \frac{1}{2}\tilde{a}(t - t_2)^2, \tag{4.27}$$

$$\omega_r(t) = \omega_2 - \tilde{a}(t - t_2). \tag{4.28}$$

Proměnné φ_2, ω_2 a t_2 opět odpovídají stavu na počátku fáze. Fáze trvá pouze čas T_{rampdown}.

Tato fáze nepoužívá přímo zadané zrychlení a. Pokud je trajektorie omezená úhlem, použije se jiné zrychlení:

$$\tilde{T}_{\text{rampdown}} = 2\left|\frac{\varphi_{\text{target}} - \varphi_2}{\omega}\right|, \tag{4.29}$$

$$\tilde{a} = \left|\frac{\omega_2}{\tilde{T}_{\text{rampdown}}}\right|. \tag{4.30}$$

Tato změna zajistí, že se motor zastaví v cílové poloze i v případě, že se motor na začátku fáze otáčel mírně pomaleji.

Pro ostatní typy trajektorií platí $\tilde{a} = a$.

Fáze zpomalování pro změnu směru

Tato fáze se normálně neuplatní. Použije se pouze na začátku vybraných trajektorií s cílovým úhlem. Motor musí již být roztočený a v důsledku toho by nestihl se zrychlením $-a$ zastavit na správném místě. Generátor proto nejprve motor rovnoměrně zpomaleně zastaví a až pak z této polohy začne sledovat lichoběžníkovou trajektorii rychlosti.

Rovnice pro tuto fázi odpovídají normální fázi zpomalování s $\tilde{a} = a$.

Fáze zastavení

Tato fáze je konečná pro všechny omezené trajektorie. Generátor vrací hodnoty

$$\varphi_r(t) = \varphi_{\text{target}}, \tag{4.31}$$

$$\omega_r(t) = 0. \tag{4.32}$$

Pro trajektorie omezené dobou trvání pohybu odpovídá φ_{target} poloze motoru na konci zpomalovací fáze. Pro trajektorie omezené úhlem je to právě cílový úhel.

4.4 Aplikační rozhraní

Vytvořený řídící systém je nutné zpřístupnit uživateli. Podobně jako u senzorů bylo jedním z cílů rozhraní co nejvíce přiblížit prostředí Pybricks [30].

Veškerá funkcionalita je obsažená v Python třídě `lib.ev3.Motor`. Její metody a atributy se dají rozdělit do čtyř skupin.

4.4.1 Vytvoření a uvolnění objektu

- Konstruktor `__init__(port, type, positive_direction)` umožňuje uživateli vytvořit novou instanci třídy.
  - Parametr `port` určuje, který port bude tato třída ovládat. Přípustné hodnoty jsou Open-Cube konstanty `Port.M1` až `Port.M4`.
  - Argument `type` říká, jestli je k portu připojený velký nebo střední EV3 motor. Podle toho se zvolí příslušný regulátor. Přípustné jsou hodnoty `Motor.LARGE_MOTOR` a `Motor.MEDIUM_MOTOR`. Argument není povinný, výchozí volbou je velký motor.
  - Parametrem `positive_direction` lze ovlivnit, který směr otáčení motoru bude kladný. Přípustné hodnoty jsou `Motor.CLOCKWISE` (výchozí) a `Motor.COUNTERCLOCKWISE`. Volba ovlivňuje také znaménka funkcí `angle()` a `speed()`.

  Pro daný port může kdykoliv existovat pouze jeden aktivní objekt `Motor`. Při pokusu o vytvoření druhé instance se vyvolá výjimka `RuntimeError`.
- Metoda `close()` uvolní port pro použití jiným objektem.

4.4.2 Nastavení regulátorů

- Atribut `pid` obsahuje trojici `(kp, ki, kd)`. Pomocí něj lze přečíst a případně změnit aktuální regulační konstanty.
- Atribut `max_acceleration` určuje maximální zrychlení motoru, které generátor trajektorie použije. Hodnota je ve stupních za sekundu na druhou. Výchozí hodnota je 6000 °/s^2, ta byla převzata z leJOS EV3 [35].
- Atribut `max_power` určuje maximální dovolenou velikost napětí na svorkách motoru v procentech napětí baterie.

4.4.3 Zjištění stavu motoru

- Metoda `angle()` vrátí současnou polohu motoru ve stupních.

- Metoda `reset_angle(new_angle=0)` umožňuje přenastavit čítač polohy na novou hodnotu `new_angle`.
- Metoda `speed()` vrátí současnou hodnotu rychlosti motoru ve stupních za sekundu.
- Metoda `power()` zpřístupňuje aktuální velikost a polaritu napětí na svorkách motoru. Hodnota je v procentech napětí baterie Open-Cube.
- Metoda `is_complete()` zjistí, jestli motor právě sleduje nějakou omezenou trajektorii (např. pomocí `run_angle`). Pokud nikoliv, vrací funkce `True`. Nekonečné pohyby (`run()`, `dc()`) se považují za skončené již od začátku, aby nedošlo k nečekaným uváznutím uživatelské aplikace.
- Metoda `wait_complete()` počká, dokud `is_complete()` nezačne vracet `True`.

4.4.4 Povely pro řízení motoru

- Metoda `coast()` (též dostupná pod názvem `stop()`) přeruší sledování trajektorie a motor odbrzdí.
- Metoda `brake()` přeruší sledování trajektorie a motor pasivně zabrzdí. V tomto režimu lze s motorem otáčet pouze ztuha.
- Metoda `hold()` zablokuje motor v současné pozici. Motor bude regulátorem aktivně držený na místě a nepůjde s ním otáčet.
- Metoda `dc(power)` přeruší sledování trajektorie a nastaví na vinutí motoru příslušné napětí v procentech napětí baterie Open-Cube.
- Metoda `run(speed)` plynule roztočí motor na rychlost `speed` ve stupních za sekundu.
- Metoda `run_angle(speed, rotation_angle, then=Motor.HOLD, wait=True)` plynule otočí motor o úhel `rotation_angle` ve stupních. Motor se bude otáčet rychlostí nejvýše `speed` stupňů za sekundu.

  Parametr `then` určuje, jakým způsobem motor zabrzdí na konci pohybu. Přípustné možnosti jsou `Motor.COAST`, `Motor.BRAKE` a `Motor.HOLD`. Ty přímo odpovídají voláním `coast()`, `brake()` a `hold()`.

  Argument `wait` určuje, zda metoda počká na konec manévru. Při použití `wait=False` lze využít funkce `is_complete()` a `wait_complete()`.
- Metoda `run_target(speed, target_angle, then=Motor.HOLD, wait=True)` plynule natočí motor do polohy `target_angle`. Zbylé argumenty mají stejný význam jako u funkce `run_angle()`.
- Metoda `run_time(speed, time, then=Motor.HOLD, wait=True)` nechá motor plynule otáčet po celkový čas `time` milisekund. Zbylé argumenty mají stejný význam jako u funkce `run_angle()`.

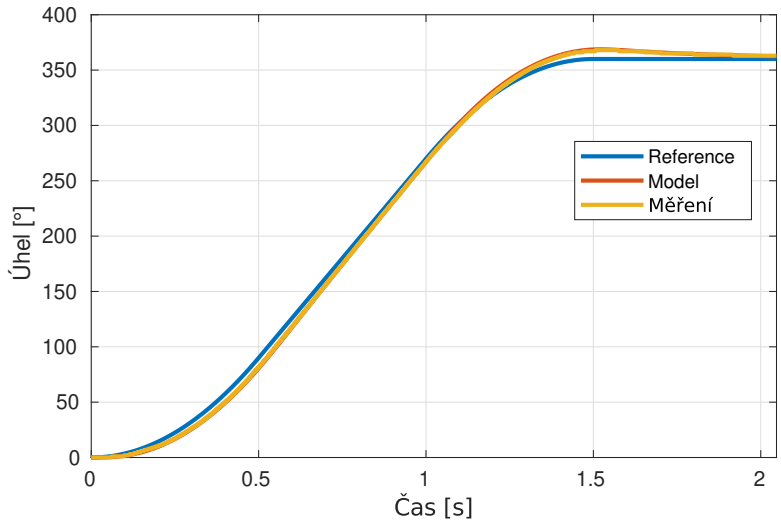
- Metoda `track_target(target_angle, target_speed=0)` umožňuje uživateli sledovat vlastní trajektorii. Zavolání této metody aktivuje jen PID regulátor a nastaví jeho referenci na `target_angle` pro polohu a `target_speed` pro rychlost.

4.5 Výsledek

Chování systému jako celku bylo otestováno změřením odezvy velkého EV3 motoru na povel `run_angle(360, 360)`. Maximální zrychlení bylo předem nastaveno na $720\,°/s^2$.

Výsledkem jsou průběhy na obrázcích 4.11 a 4.12. Ty zaznamenávají tři průběhy: ideální referenci vytvořenou v prostředí Matlab, odezvu matematického modelu a odezvu skutečného motoru.

Odezva z pohledu polohy 4.11 vypadá podle očekávání. Motor referenci sleduje poměrně dobře. Poloha má mírný překmit; to ale odpovídá pozorováním u návrhu PID konstant.



Obrázek 4.11: Časový průběh úhlové polohy při provádění testovacího povelu.

V průbězích rychlosti 4.12 ale lze nalézt několik příležitostí pro zlepšení. Odezva rychlosti při návrhu regulátoru nebyla nesledována, proto se některé vlastnosti projevily až zde.

- **Překmit**. Pro jeho zmenšení by bylo možné zvolit jiné regulační konstanty. Problém řeší také použití jiného nastavení PID tuneru – při zafixování cíle na sledování reference se překmit v simulacích zmenšil.
- **Zpoždění ve sledování reference**. Lepšího výsledku by mohlo jít dosáhout jinou volbou regulačních konstant. Regulátor by také šlo také

rozšířit o přímovazební blok od reference rychlosti, který by zajistil včasnou reakci motoru.

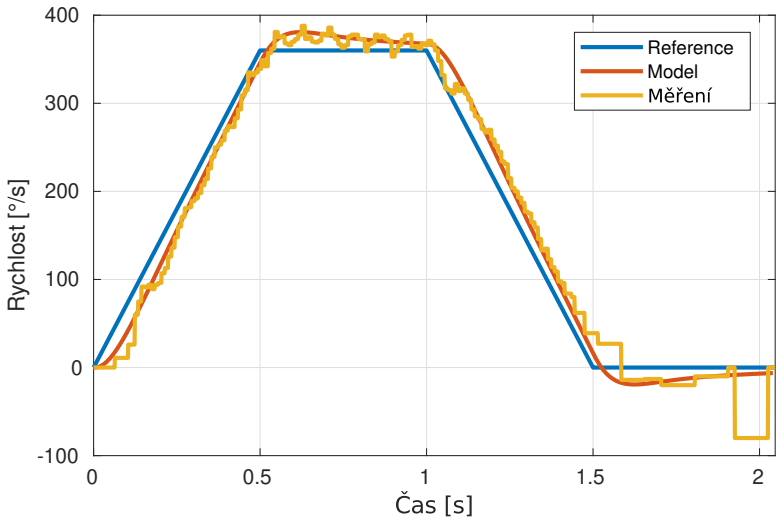
- **Oscilace rychlosti**. Její zdroj se nepodařilo přesně identifikovat. Může se jednat o nepředvídanou interakci zvolené metody měření rychlosti s regulátorem. Může také jít o příliš velké zesílení otevřené smyčky na problematické frekvenci.

- **Proměnlivá frekvence měření rychlosti**. V obrázku 4.12 lze pozorovat nevýhodu metody měření rychlosti 4.2.4. Při nízkých rychlostech přichází nové výsledky měření pomaleji – v nejhorším případě každých 100 ms. Při nízkých rychlostech jsou totiž intervaly mezi pulzy enkodéru dlouhé a zvolená metoda má tomu úměrné zpoždění. Jedním možným řešením by bylo omezit min. měřitelnou rychlost na 50 °/s–100 °/s.

- **Chybná měření při malých rychlostech**. Na konci manévru je patrný chvilkový skok naměřené rychlosti na hodnotu −80 °/s. Motor se ale touto rychlostí reálně nepohyboval. Aby regulátor na tento skok nereagoval, byla do něj přidána podmínka, aby se při požadavku na rychlost nižší než 30 °/s považovala naměřená rychlost za nulovou.

  Později bylo zjištěno, že skok je způsobený implementační chybou při měření malých rychlostí. Finální kód již má tuto chybu opravenou.

- **Vstup do generátoru trajektorie**. Ten aktuálně některá rozhodnutí provádí na základě okamžité polohy a rychlosti motoru. Jak je ale z grafů patrné, hodnoty mohou mít určité zpoždění. Jedním z nepřímých důsledků je náhlý propad rychlosti na začátku zpomalovací fáze. Lepším řešením by bylo generovat trajektorii bez zpětné vazby od motoru.



Obrázek 4.12: Časový průběh úhlové rychlosti při provádění testovacího povelu.

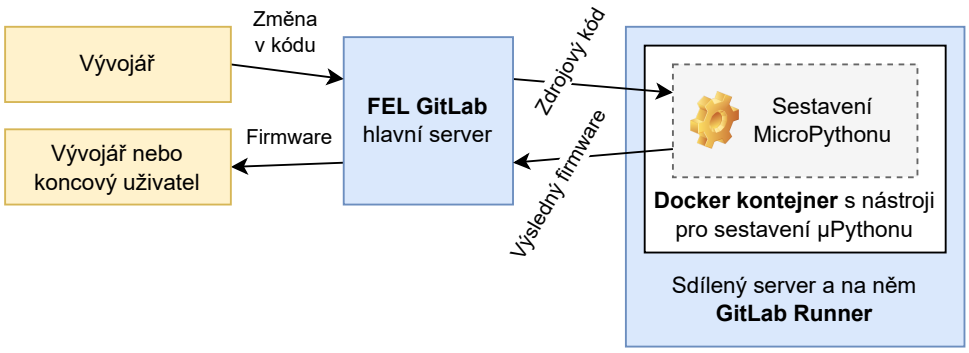
Kapitola 5

Sestavování MicroPythonu na FEL GitLabu

Použití nativních modulů s sebou přineslo určité komplikace. Jak bylo naznačeno v sekci 2.2.1, s nimi již není možné na Open-Cube dále využívat standardní MicroPython firmware pro Raspberry Pi Pico. Bylo tedy nutné vyřešit problém toho, kde sestavovat a jak distribuovat MicroPython firmware obsahující také nové Open-Cube knihovny.

Pro tento účel byl využit FEL GitLab – fakultou provozovaná platforma pro spolupráci a vývoj software. Na té v současné době probíhá vývoj Open-Cube. GitLab mimo správy zdrojových kódů ale poskytuje také mechanismus, jak automaticky sestavovat aplikace po přidání nových změn [53].

Na základě předchozích zkušeností autora a dokumentace GitLabu [53] bylo navrženo a realizováno řešení na obrázku 5.1.



Obrázek 5.1: Sestavení MicroPythonu na FEL GitLabu

5.1 Volba serveru

Nejprve bylo nutné najít server, na kterém by se mohl MicroPython sestavovat. Server by pak šlo ke GitLabu připojit pomocí tzv. runneru. Runner je služba běžící na serveru, která zde umožní GitLabu spouštět různé úlohy. [53]

Původním plánem bylo využít servery fakulty. Ta poskytuje několik GitLab runnerů pro drobné úlohy. Dané servery ale neměly dostatek volného místa na disku pro potřebné nástroje, od jejich využití proto bylo dočasně upuštěno. Místo toho byl využit soukromý server autora postavený na desce Odroid-HC2 [54]. Na něj byl podle dokumentace [53] nainstalován runner a tak byl připojen k FEL GitLabu.

Toto řešení je nešťastné v tom, že prostředí váže na soukromý hardware. Z dlouhodobého hlediska by bylo lepší runner nainstalovat na server vlastněný fakultou nebo katedrou měření. Další kroky proto směřují k tomu, aby server bylo možné později jednoduše vyměnit.

5.2 Prostředí pro sestavení MicroPythonu

Pro sestavení MicroPythonu je potřeba několik nástrojů: přinejmenším CMake a kompilátor pro RP2040 [7, s. 5]. Obojí je nutné na server nainstalovat.

Pro jednodušší přenositelnost bylo využito tzv. kontejnerů a jejich implementaci jménem Docker. Tato technologie umožňuje spouštět programy v definovaném a izolovaném prostředí [55]. GitLab ji dokáže využít pro sestavování projektů [56].

Obsah prostředí lze definovat pomocí textového souboru `Dockerfile` [57]. Společnost vyvíjející Docker ale také poskytuje repozitář s komunitou předpřipravenými obrazy prostředí – Docker Hub [58]. Zde byl nalezen již hotový obraz s nástroji pro sestavení MicroPythonu pro RP2040 [59].

Obraz [59] ale nebylo možné použít přímo, spustit se totiž dokáže pouze na procesorech s architekturou x86-64. Použitý server ale využívá procesor architektury ARM [54]. Byl proto vytvořen s ní kompatibilní klon obrazu [59]. Jeho textová definice byla vložena do repozitáře Open-Cube.

Díky využití kontejnerů by mělo jít jednoduše změnit server, kde sestavení probíhá. Nezbytné je pouze přenastavit GitLab na používání obrazu [59] (viz dále) a pak nastavit, aby se úlohy odesílaly na runner na novém serveru. Stažení obrazu kontejneru se provede automaticky.

5.3 Nastavení FEL GitLabu

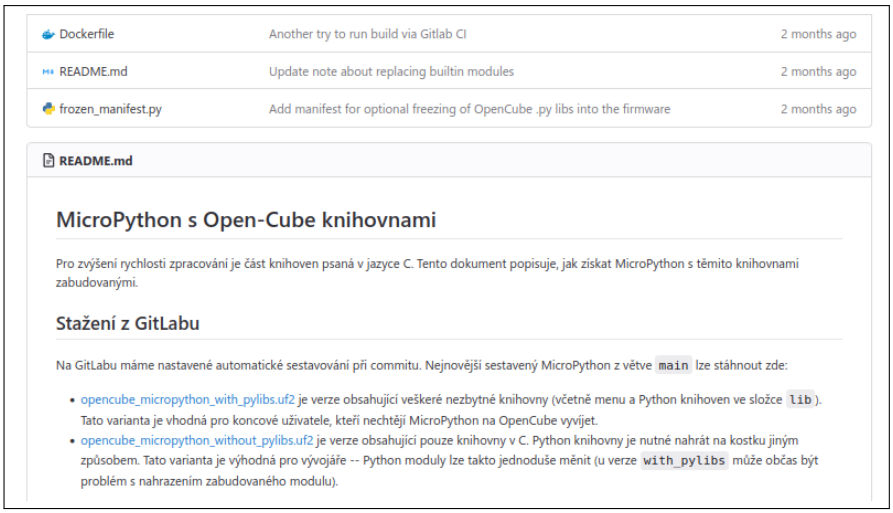
Dalším krokem bylo nastavit FEL GitLab tak, aby v prostředí 5.2 vyvolal sestavení MicroPython firmware po každé změně v repozitáři.

Nastavení bylo provedeno pomocí souboru `.gitlab-ci.yml` v repozitáři projektu. Soubor definuje, co a jak se má sestavit a jaké jsou výstupní soubory. [60] Definice kroků sestavení vychází dokumentace k Pythonu na RP2040 [7, s. 5].

Výsledkem sestavení jsou dva UF2 soubory:

- `opencube_micropython_without_pylibs.uf2` obsahuje pouze MicroPython a nativní Open-Cube knihovny.
- `opencube_micropython_with_pylibs.uf2` obsahuje navíc Open-Cube knihovny v jazyce Python. Tento soubor je dále popsán v sekci 5.4.

Soubory bylo zprvu poměrně obtížné v rozhraní GitLabu najít. Řešením bylo na úvodní stránku dokumentace vložit dynamický odkaz na nejnovější dostupný firmware, viz obrázek 5.2.



Obrázek 5.2: Odkazy na sestavený firmware v README souboru Open-Cube.

Firmware si z této stránky může kdokoliv stáhnout a na Open-Cube nainstalovat.

5.4 Vytvoření jednotného MicroPython balíčku

Jak bylo uvedeno v sekci 2.2.1, dříve nebyly Open-Cube Python knihovny součástí firmwaru. Naopak je bylo nutné do kostky nahrát jako soubory. Nativní knihovny se v tomto liší, jsou již do firmware zabudované. To přináší určité výhody, mimo jiné jednodušší instalaci a aktualizaci knihoven.

MicroPython nicméně poskytuje podporu i pro vestavění Python souborů. Jejich data mohou být ve firmware uložená v předzpracované podobě, která šetří místo v paměti a rychleji se načítá. [61]

Funkci lze aktivovat pomocí tzv. manifest souboru. Přes něj je možné specifikovat, které Python soubory se do firmware mají vložit a v jaké podobě. [61] Pro Open-Cube byl příslušný soubor vytvořen tak, aby se do MicroPythonu

zabudovaly Open-Cube knihovny, menu a zároveň pomocné soubory interní pro MicroPython.

Vestavění Python knihoven bylo následně zprovozněno i na GitLabu. Stažením a instalací souboru `opencube_micropython_with_pylibs.uf2` lze na Open-Cube nahrát nový firmware se všemi potřebnými knihovnami. Tím se zjednodušil postup aktualizace firmware.

Kapitola 6

Praktické ověření

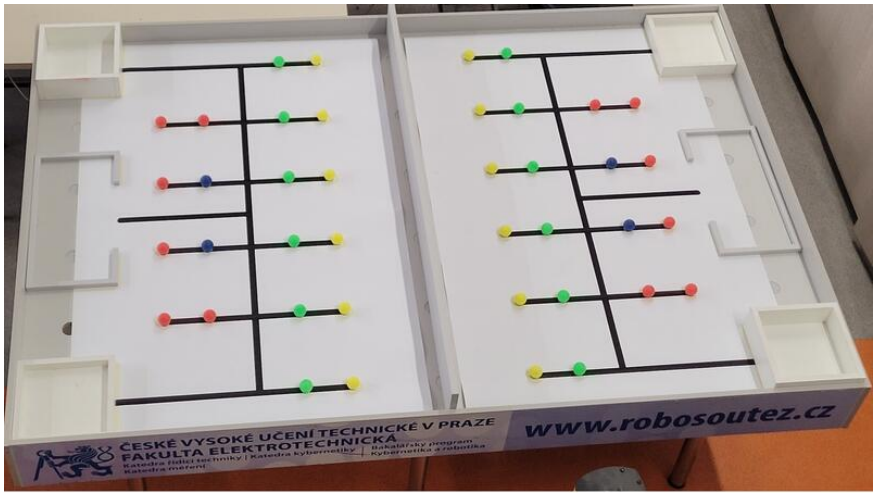
Posledním nutným bodem zadání je otestovat knihovnu na dvou úlohách na Robosoutěž. Vybrány byly úlohy Ping-pong a Pac-man.

Cílem této části není nutně sestavit nejlepšího robota, důraz je kladen více na prověření naprogramovaných funkcí. Proto byla místy volena řešení, která by v soutěži nevyhrála, ale jsou jednodušší na sestavení.

6.1 Úloha „Ping-pong“

6.1.1 Podstata zadání

Hrací plocha pro tuto úlohu je ukázaná na obrázku 6.1. Na ploše je rozložených 20 míčků různých barev. Toto rozložení je pro každou soutěžní jízdu stejné. Mimo to jsou zde tři zásobníky – vysoký, střední a pak nízký s vjezdem.



Obrázek 6.1: Hřiště na úlohu Ping-pong. [62]

Cílem soutěžících je pomocí robota přemístit míčky do těchto zásobníků. Za každý přesunutý míček získá tým určitý počet bodů. Ten závisí na výšce zásobníku: ve vysokém lze získat 3 body, ve středním 2 body a v nízkém jen 1 bod. Navíc, pokud v nějakém zásobníku jsou míčky pouze jedné barvy, zisk bodů za míčky sem umístěné se zdvojnásobí. [62]

Výjimkou z tohoto bodování jsou dva „otrávené“ modré míčky. Pokud robot modrý míček umístí do zásobníku nebo jej nechá na hrací ploše, několik bodů za to tým ztratí. Aby se tak nestalo, robot musí buď míček z plochy zvednout, nebo jej poslat na plochu protihráče. [62]

6.1.2 Konstrukce

Pro ověření funkcí Open-Cube byla zvolena strategie se sbíráním míčků po jednom. Její výhodou je, že pro její realizaci stačí jednodušší konstrukce. Pro sbírání vícero míčků najednou by bylo nutné navrhnout zásobník a další konstrukční prvky.

Robot byl sestaven jako dvoukolové vozítko s třetím opěrným bodem, viz obrázek 6.2b.

Pro sbírání míčků byl navržen systém podobný vysokozdvižnému vozíku. Na předku robota jsou připevněné dvě ližiny, které se mohou posouvat nahoru a dolů, viz obr. 6.2a. Pokud jsou snížené dolů, robot s nimi může najet pod míček a zvednout jej. U zásobníku pak může míček mírným snížením ližin upustit. Zvedání řídí střední motor skrytý uvnitř konstrukce.

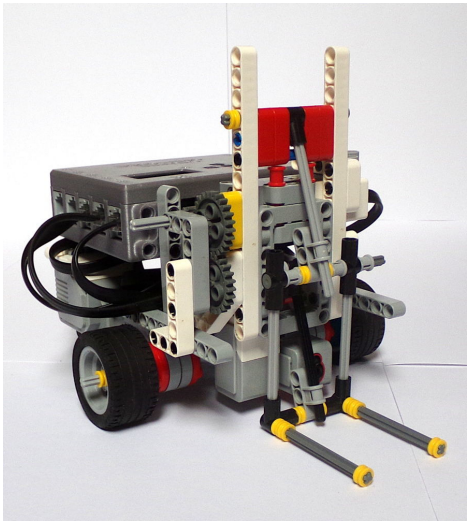
Přítomnost míčku na ližinách může robot detekovat pomocí prvního senzoru barev, viz obr. 6.2a. Původně bylo plánováno zde využít ultrazvukový senzor, ten ale nedokázal míčky spolehlivě detekovat.

Navigaci v bludišti je řešena pomocí druhého senzoru barev. Pomocí něj může robot sledovat jednu stranu čáry. Senzor je posunutý tak, aby při sledování byl robot zarovnaný na střed čáry a mohl tak bez kolize najet pod míček. Druhý senzor by běžně na Robosoutěži nebylo možné použít; po konzultaci s vedoucím mi ale bylo uznáno i toto řešení.

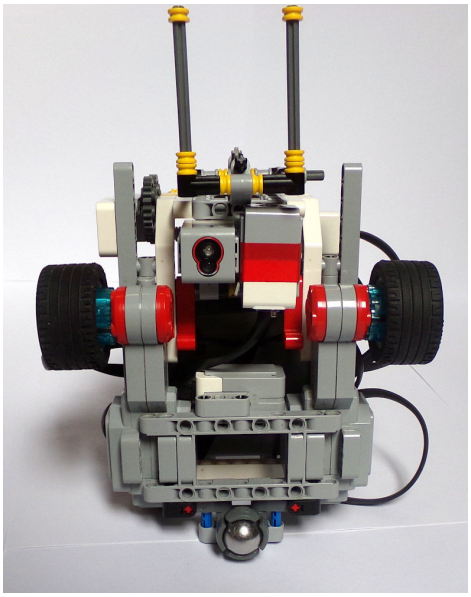
Konstrukce robota obsahuje také gyroskop (spodní senzor na obr. 6.2b). Jeho cílovým využitím byla navigace v bludišti, tu se ale podařilo vyřešit jiným způsobem. Senzor proto nakonec zůstal nevyužitý.

6.1.3 Software

Robota na hrací ploše řídí program běžící uvnitř Open-Cube. Lze jej rozložit na tři části: regulátor motorů, algoritmus sledování čáry a logika sbírání míčků.



(a) : Pohled zepředu.



(b) : Pohled zespoda.

Obrázek 6.2: Konstrukce testovacího robota pro úlohu Ping-pong.

Regulátor motorů popsaný v sekci 4.3 původně vznikl jako součást řešení této úlohy. Po prvním otestování úlohy byl regulátor přepsán a začleněn do knihovny pro motory. Následně bylo ověřeno, že úloha funguje i s novou verzí.

Pomocí regulátoru bylo možné s robotem jednodušeji pohybovat – například s ním na místě otočit o 180°. Problém by bylo možné řešit i jinak (gyroskop, senzor čáry), takové řešení by ale šlo hůře zobecnit.

Další částí je sledování čáry. To je na nižší úrovni založené na PID regulátoru. Robot se pomocí něj při jízdě snaží udržet v takové poloze, aby senzor barev vracel hodnotu přibližně v polovině kalibrovaného rozsahu.

Na vyšší úrovni program obsahuje detekci křižovatek na čáře (viz obr. 6.1). Na křižovatce tak robot dokáže zatočit nebo pokračovat v jízdě. Tímto způsobem bylo možné plně vyřešit navigaci v bludišti.

Poslední částí programu je logika sbírání míčků a celkového řešení úlohy. Protože rozložení míčků na herní ploše je neměnné, do programu stačilo napevno vložit postup operací, které má robot provést. Mezi operace patří například sebrání a upuštění míčku a navigace po křižovatkách.

6.1.4 Výsledek

Úlohu se mi podařilo vyřešit. Videozáznam z ukázkové jízdy je dostupný na adrese `https://youtu.be/yZZEdAmElWA` (kopii lze nalézt v příloze). Robot sbírá pouze zelené a žluté míčky, řešení by ale bylo možné rozšířit i na zbylé barvy. Vozítko je také poměrně pomalé, v bludišti se ale pohybuje spolehlivě.

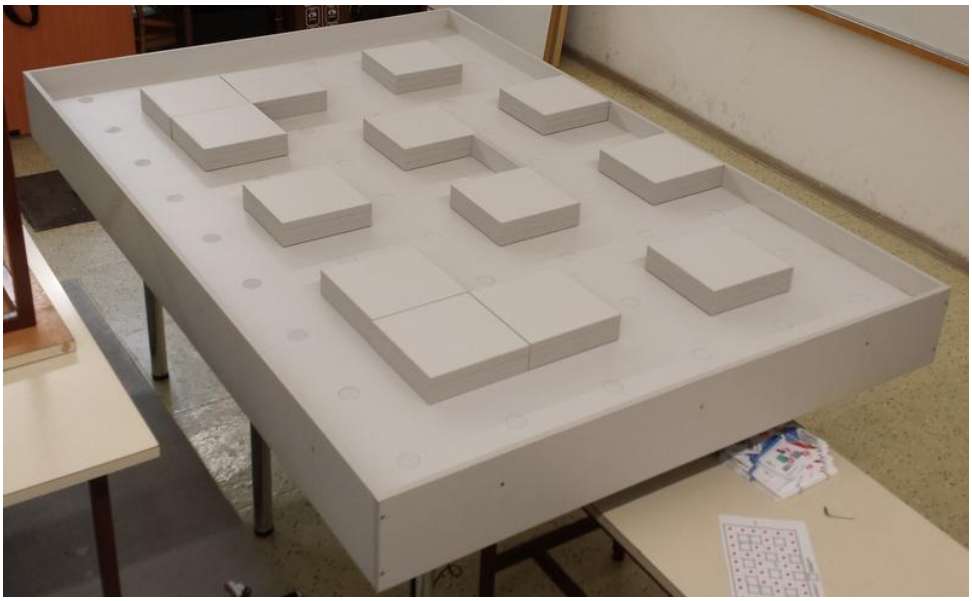
Touto úlohou bylo ověřeno několik funkcí:

- Snímání čáry senzorem barev funguje. Rozhraní motorů zároveň dokáže dostatečně rychle reagovat na požadavky tohoto regulátoru. Pro sledování byl použit neregulovaný režim motorů.
- Snímání RGB hodnot pomocí senzoru barev také funguje. Detekce míčku je založená na transformaci do barevného prostoru HSV, kde se následně provede prahování jasové složky.
- Metody pro řízení motoru lze použít pro jednoduché pohyby robota (otáčení na místě, jízda vpřed). Omezení zrychlení bylo také využito při zvedání míčku.

6.2 Úloha „Pac-man“

6.2.1 Podstata zadání

Cílem soutěžících v této úloze je sestavit robota, který do 90 sekund projede co největší část bludiště na obrázku 6.3. Roboti startují vždy ze startovacího boxu uprostřed bludiště. Počet bodů, které tým za soutěžní jízdu získá, přímo odpovídá počtu různých políček navštívených robotem. [63]



Obrázek 6.3: Hřiště na úlohu Pac-man. [63]

Úloha existuje ve dvou mutacích. Ve variantě pro střední školy [63] soutěžící předem neznají rozložení hrací plochy. Zároveň se v bludišti může pohybovat

„duch“ – nepřátelský robot, kterého se robot soutěžících nesmí dotknout. Zadání pro základní školy [64] ducha nepřipouští a soutěžící zde mají k dispozici plány čtyř přípustných rozložení hrací plochy.

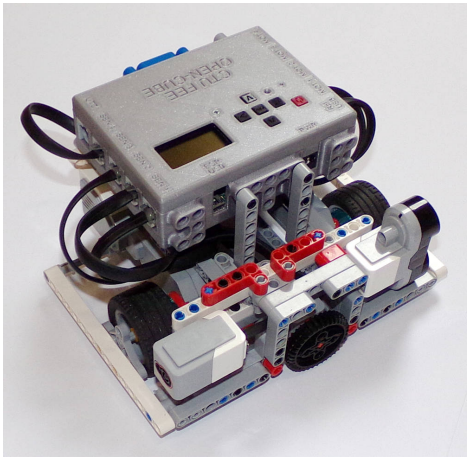
Pro otestování Open-Cube byla zvolena varianta pro střední školy s tím, že nebude použit duch. Obdobné podmínky měli soutěžící ve finále Robosoutěže 2014 [63].

6.2.2 Konstrukce

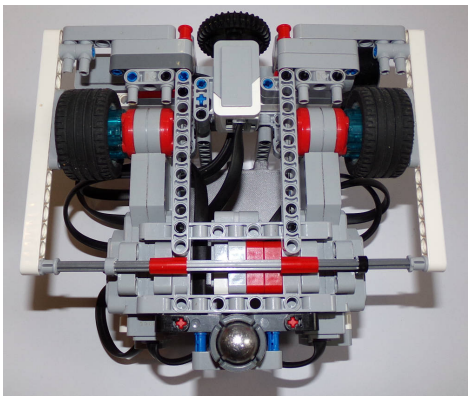
Pro otestování Open-Cube bylo opět použito dvoukolové vozítko. Díky tomu stačilo pouze upravit konstrukci ze sekce 6.1 odstraněním ližin a přidáním vhodných senzorů. Vozítko také dovolilo lépe otestovat gyroskop, ultrazvukový senzor a senzor dotyku, které se v sekci 6.1 nepodařilo využít.

Výsledná konstrukce je znázorněná na obrázku 6.4. Využity byly všechny senzory dostupné ve výukové edici stavebnice EV3:

- Ultrazvukový senzor míří nalevo od robota a slouží k poznávání bludiště.
- Senzor barev sleduje druhou stranu robota a plní stejnou funkci.
- Senzor dotyku umožňuje robotovi poznat, že narazil do překážky.
- Gyroskop uvnitř konstrukce zajišťuje držení správného směru v bludišti a detekci nárazu do rohu překážky.



(a) : Pohled zepředu.



(b) : Pohled zespoda.

Obrázek 6.4: Konstrukce testovacího robota pro úlohu Pac-man.

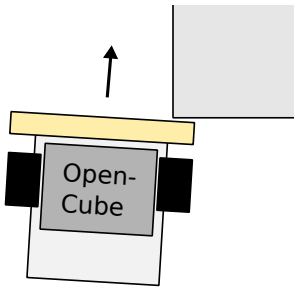
6.2.3 Software

Řídící program se opět skládá z několika částí. První z nich je mapa hrací plochy, kterou si robot průběžně sestavuje. Reprezentovaná je 9×6 polem

typů čtverců: zatím neznámý, překážka, nenavštívené volné pole, navštívené volné pole a startovací box. V této mapě dokáže robot pomocí Dijkstrova algoritmu vyhledat nejbližší nenavštívené nebo neznámé políčko. Doplňování mapy probíhá průběžně pomocí měření ze senzorů.

Navazující částí programu je řízení pohybů mezi poli na hrací ploše. Při většině pohybů robot sleduje úhel a rychlost z gyroskopu. Pomocí 100 Hz PD regulátoru pak upravuje svoje natočení tak, aby stále mířil správným směrem.

Zároveň lze gyroskopem detekovat náraz do rohu překážky, viz obrázek 6.5. Pokud regulátor vyvolá takový zásah, že se jedno z kol přestane otáčet, aktivuje se opravná procedura. Pomocí ní se robot dokáže opět zarovnat se stěnou bludiště a tak neztratit orientaci.



Obrázek 6.5: Znázornění kolize robota s rohem překážky.

Hlavní smyčka programu spojuje zbylé části dohromady. Řešení úlohy probírá po „tazích“. Jeden tah odpovídá přesunu na další hrací pole, případně s otočením robota. V každém z nich robot vždy jede k nejbližšímu neznámému nebo nenavštívenému bloku.

6.2.4 Výsledek

Robot úlohu úspěšně vyřešil. Záznam z jízdy je možné zhlédnout na adrese `https://youtu.be/TUHyd7wESdU` (kopii lze nalézt v příloze). V čase 1:23 lze také pozorovat průběh opravné procedury, díky které robot může pokračovat i po kolizi s překážkou.

Pomocí úlohy byla oveřena funkčnost všech senzorů kromě infračerveného, ten ale ve výukových sadách EV3 není dostupný. Pro ovládání motorů byl použit převážně neregulovaný režim, který již byl otestovaný úlohou v sekci 6.1. Řízené pohyby byly použity pouze pro posun zpět po nárazu do překážky.

Úloha byla otestována také na alternativním programu, který nesledoval průběžně gyroskop. Řízení pohybů robota zde zajišťovaly regulátory navržené v sekci 4.3. Výsledné řešení bylo funkční, robot ale častěji narážel do rohů překážek. Problém je nejspíše způsobený nepřesným otáčením robota při změně směru. Stejný problém byl ale pozorován i u dřívějších řešení pomocí sad NXT a EV3.

Zpoždění měření rychlosti při malých otáčkách motoru

Během programování úlohy byla nalezena problematická vlastnost měření rychlosti ze sekce 4.2.4. Pokud motor rychle zastaví, trvá 100 ms–200 ms, než hodnota měřené rychlosti přejde do nuly. Toto bylo diskutováno v sekci 4.5.

Nenulová měřená rychlost ale může zmást generátor trajektorií, který se snaží plynule navázat na předchozí rychlost motoru. Kvůli tomu může robot na začátku pohybu po přímce mírně zatočit.

Problém je možné řešit několika způsoby:

- Pokud by uživatel přes rozhraní dokázal deaktivovat plynulé navázání rychlosti, problém by bylo možné obejít. K tomuto účelu by mohl sloužit nový argument příslušných funkcí, například `smooth_ramp_up`.
- Pro plynulé navázání rychlosti v nepoužívat rychlost měřenou na motoru, ale pouze poslední výstup generátoru. Tím by generátor přestal záviset na zpožděné hodnotě rychlosti.
- Použít alternativní metodu měření rychlosti, která by měla menší zpoždění při nízkých rychlostech.
- Vylepšit regulátor motorů tak, aby se minimalizovala chyba sledování rychlosti během zrychlování a zpomalování.

Kapitola 7

Závěr

Hlavním výstupem této práce je sada knihoven pro novou kostku Open-Cube. Knihovny umožňují pomocí programovacího jazyka Python ovládat motory a senzory ze stavebnice LEGO Mindstorms EV3. Nová kostka tak může postupně nahradit zastarávající řídící jednotky EV3.

Prvním mým krokem bylo seznámit se s Open-Cube a projektem MicroPython, který tvoří jádro řídícího software kostky. Dále bylo nutné zjistit, jak fungují periferie stavebnice EV3.

Na základě těchto informací byla nejprve vytvořena knihova pro spolupráci se senzory EV3. To obnášelo tři hlavní úlohy. První z nich bylo zprovoznit sériovou komunikaci na portech kostky. Nad ní pak bylo možné naprogramovat podporu pro protokol, který senzory používají. Tu pak bylo nutné zpřístupnit budoucím uživatelům Open-Cube v podobě několika Python tříd.

Dalším úkolem bylo přidání podpory ovládání motorů. Nejdříve bylo vytvořeno základní rozhraní, které umožňuje motory řídit bez regulace. Nad ním byl navržen a naprogramován polohový regulátor a generátor rovnoměrně zrychlených trajektorií. Vše následně bylo zpřístupněno uživatelům Open-Cube pomocí jednotné Python třídy `Motor`.

Dále bylo potřeba zajistit, že si uživatelé budou moci firmware kostky i s novými knihovnami stáhnout ze stránek Open-Cube. Pro tento účel byla využita fakultní instance GitLabu, kde bylo následně vše připraveno.

Knihovna byla na závěr otestována na dvou úlohách pro Robosoutěž. Zvoleny byly úlohy Ping-pong a Pac-man. Obě se podařilo vyřešit a ověřit na nich, že podpora pro senzory a motory pracuje správně. Zároveň ale byly nalezeny určité možnosti pro zlepšení knihovny, na kterých autor plánuje dále pracovat.

Seznam literatury

1. *Robosoutěž* [online]. Praha: ČVUT v Praze, Fakulta elektrotechnická, 2023 [cit. 2023-03-17]. Dostupné z: `https://robosoutez.fel.cvut.cz/`.
2. DIMINO, Marissa. *MINDSTORMS Education EV3 Retirement FAQs* [online]. The LEGO Group [cit. 2023-03-17]. Dostupné z: `https://community.legoeducation.com/blogs/36/95`.
3. *Open-Cube* [online]. Praha: MAGLAB, ČVUT v Praze, Fakulta elektrotechnická, 2023 [cit. 2023-03-17]. Dostupné z: `https://maglab.fel.cvut.cz/open-cube/`.
4. NOVOTNÝ, David. *Schémata kostky Open-Cube* [online]. Praha: MAGLAB, ČVUT v Praze, Fakulta Elektrotechnická, 2023 [cit. 2023-03-17]. Dostupné z: `https://gitlab.fel.cvut.cz/open-cube/hardware/-/blob/main/devkit/brick/LEGOkostka.pdf`.
5. *RP2040 Product brief* [online]. Cambridge: Raspberry Pi Ltd., 2023 [cit. 2023-03-17]. Dostupné z: `https://datasheets.raspberrypi.com/rp2040/rp2040-product-brief.pdf`.
6. *RP2040 Datasheet: A microcontroller by Raspberry Pi* [online]. Cambridge: Raspberry Pi Ltd., 2023 [cit. 2023-03-17]. Dostupné z: `https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf`.
7. *Raspberry Pi Pico Python SDK: A MicroPython environment for RP2040 microcontrollers* [online]. Cambridge: Raspberry Pi Ltd., 2023 [cit. 2023-03-18]. Dostupné z: `https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-python-sdk.pdf`.
8. ŠTYCH, David; INDRYCH, Martin; ŠKVRNA, Jan; KOLDINSKÝ, Michal; FUČELA, Peter. *LEGO Mindstorm Open-Cube: Výstup z týmového projektu B3MPVT1*. Praha, 2022. Školní práce. ČVUT v Praze, Fakulta elektrotechnická.
9. *RP2040: The C/C++ SDK* [online]. Cambridge: Raspberry Pi Ltd., 2023 [cit. 2023-03-20]. Dostupné z: `https://github.com/raspberrypi/pico-sdk`.

10. *MicroPython documentation: Class UART* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-05-18]. Dostupné z: https://docs.micropython.org/en/latest/library/machine.UART.html.

11. *MicroPython Internals: MicroPython external C modules* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-04-09]. Dostupné z: https://docs.micropython.org/en/latest/develop/cmodules.html.

12. *MicroPython Internals: Native machine code in .mpy files* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-04-09]. Dostupné z: https://docs.micropython.org/en/latest/develop/natmod.html.

13. *MicroPython Internals: Folder structure* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-04-09]. Dostupné z: https://docs.micropython.org/en/latest/develop/gettingstarted.html#folder-structure.

14. *Ukázkový modul rozšiřující MicroPython* [online]. Autoři MicroPythonu, 2023 [cit. 2023-04-10]. Dostupné z: https://github.com/micropython/micropython/tree/master/examples/usercmodule/cexample.

15. *Zdrojové kódy MicroPythonu* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-04-10]. Dostupné z: https://github.com/micropython/micropython.

16. VÖRÖS, Zoltán. *User-defined micropython modules* [online]. Německo, 2022 [cit. 2023-04-10]. Dostupné z: https://micropython-usermod.readthedocs.io/en/latest/.

17. *LEGO MINDSTORMS EV3 Hardware Developer Kit* [online]. Billund: The LEGO Group, 2013 [cit. 2023-03-31]. Dostupné z: https://education.lego.com/v3/assets/blt293eea581807678a/blt471320cb14ed0291/5f880386631d5a2165df4144/lego-mindstorms-ev3-hardware-developer-kit.zip.

18. VALK, Laurens. *EV3 and NXT: Difference and Compatibility* [online]. Netherlands [cit. 2023-03-17]. Dostupné z: http://robotsquare.com/2013/07/16/ev3-nxt-compatibility/.

19. KÖHLER, Sven. *UART Sensor Protocol* [online]. 2015 [cit. 2023-03-31]. Dostupné z: https://sourceforge.net/p/lejos/wiki/UART%20Sensor%20Protocol/.

20. *Zdrojový kód EV3 firmware 1.09E* [online]. Billund: The LEGO Group, 2020 [cit. 2023-03-31]. Dostupné z: https://education.lego.com/v3/assets/blt293eea581807678a/blt130d48cde1031bd7/5f8807e41c5db60f7d0ae47c/lego-mindstorms-ev3-firmware-source-code.bz2.

21. *EV3 Color Sensor #exposed* [online]. Kladno: Jakub Vaněk, 2020 [cit. 2023-03-31]. Dostupné z: https://github.com/JakubVanek/lms2012-stuff/tree/master/ev3color.

22. LECHNER, David. *How to Repair the Rotation Sensor on a LEGO MINDSTORMS EV3 Motor* [online]. 2018 [cit. 2023-05-18]. Dostupné z: https://lechnology.com/2018/07/how-to-repair-the-rotation-sensor-on-a-lego-mindstorms-ev3-motor/.

23. *pySerial: Python serial port access library* [online]. Chris Liechti, 2020 [cit. 2023-04-08]. Dostupné z: https://github.com/pyserial/pyserial.

24. *Zdrojové kódy MicroPythonu: Ovladač UART periferie RP2040* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-05-21]. Dostupné z: https://github.com/micropython/micropython/blob/05e143dbddbde07e4f31a0376b8869bcf9fb1939/ports/rp2/machine_uart.c.

25. *MicroPython documentation: Writing interrupt handlers* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-05-21]. Dostupné z: https://docs.micropython.org/en/latest/reference/isr_rules.html.

26. NOVOTNÝ, David. *Testování latence přerušení v MicroPythonu*. Praha, 2022.

27. *MicroPython documentation: Class StateMachine* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-05-21]. Dostupné z: https://docs.micropython.org/en/latest/library/rp2.StateMachine.html.

28. *MicroPython documentation: Functions related to the hardware* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-05-21]. Dostupné z: https://docs.micropython.org/en/latest/library/machine.html.

29. *MicroPython documentation: Maximising MicroPython speed* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-04-09]. Dostupné z: https://docs.micropython.org/en/latest/reference/speed_python.html.

30. *Ev3-micropython 2.0.0 documentation: LEGO® MINDSTORMS® EV3 motors and sensors* [online]. Billund: The LEGO Group, 2020 [cit. 2023-04-13]. Dostupné z: https://pybricks.com/ev3-micropython/ev3devices.html.

31. *Gyro Sensor Revisited* [online]. Sanjay a Arvind Seshan, 2018 [cit. 2023-04-13]. Dostupné z: https://ev3lessons.com/en/ProgrammingLessons/advanced/GyroRevisited.pdf.

32. SMITH, Alvy Ray. Color Gamut Transform Pairs. In: *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1978, s. 12–19. SIGGRAPH '78. ISBN 9781450379083. Dostupné z DOI: 10.1145/800248.807361.

33. *Linux Kernel Drivers for ev3dev-buster: Sensor Data* [online]. Edmond, Oklahoma: Ev3dev.org, 2022 [cit. 2023-04-08]. Dostupné z: http://docs.ev3dev.org/projects/lego-linux-drivers/en/ev3dev-buster/sensor_data.html.

34. *Dokumentace leJOS EV3: Třída UnregulatedMotor* [online]. 2015 [cit. 2023-05-08]. Dostupné z: https://lejos.sourceforge.io/ev3/docs/lejos/hardware/motor/UnregulatedMotor.html.

35. GLASSEY, Roger; SHAW, Andy. *Dokumentace leJOS EV3: Třída BaseRegulatedMotor* [online]. 2015 [cit. 2023-05-08]. Dostupné z: https://lejos.sourceforge.io/ev3/docs/lejos/hardware/motor/BaseRegulatedMotor.html.

36. BOUWMEESTER, Aswin. *Dokumentace leJOS EV3: Rozhraní Chassis* [online]. 2015 [cit. 2023-05-08]. Dostupné z: https://lejos.sourceforge.io/ev3/docs/lejos/robotics/chassis/Chassis.html.

37. *Ev3-micropython 2.0.0 documentation: Třída DriveBase* [online]. Billund, 2020 [cit. 2023-05-08]. Dostupné z: https://pybricks.com/ev3-micropython/robotics.html%5C#pybricks.robotics.DriveBase.

38. *BD63573NUV: Single H-Bridge Driver High-Speed Switching Type* [online]. Kyoto: ROHM Co., Ltd., 2015 [cit. 2023-03-17]. Dostupné z: https://fscdn.rohm.com/en/products/databook/datasheet/ic/motor/dc/bd63573nuv-e.pdf.

39. *Zdrojové kódy Open-Cube: Ovladač displeje SH1106* [online]. 2022 [cit. 2023-05-21]. Dostupné z: https://gitlab.fel.cvut.cz/open-cube/firmware/-/blob/main/lib/cube/sh1106.py.

40. *Zdrojové kódy RP2040 SDK: Ovladač hardwarové I^2C periferie* [online]. Cambridge: Raspberry Pi Ltd., 2021 [cit. 2023-05-21]. Dostupné z: https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_i2c/i2c.c#L135.

41. *Dokumentace RP2040 SDK: Mutex* [online]. Cambridge: Raspberry Pi Ltd., 2021 [cit. 2023-05-21]. Dostupné z: https://www.raspberrypi.com/documentation/pico-sdk/high_level.html#mutex.

42. YU, Zhenyu. *Symmetric PWM Outputs Generation with the TMS320C14 DSP* [online]. Texas Instruments, 1997 [cit. 2023-05-10]. Dostupné z: https://www.ti.com.cn/cn/lit/an/spra278/spra278.pdf.

43. LECHNER, David. *Comment regarding the doubling of motor encoder precision* [online]. Ev3dev GitHub issues, 2018 [cit. 2023-05-21]. Dostupné z: https://github.com/ev3dev/ev3dev/issues/148%5C#issuecomment-403342827.

44. PETRELLA, Roberto; TURSINI, Marco; PERETTI, Luca; ZIGLIOTTO, Mauro. Speed measurement algorithms for low-resolution incremental encoder equipped drives: a comparative analysis. In: *2007 International Aegean Conference on Electrical Machines and Power Electronics*. 2007, s. 780–787. Dostupné z DOI: 10.1109/ACEMP.2007.4510607.

45. FACCIO, M.; GRANDE, P.; PARASILITI, F.; PETRELLA, R.; TURSINI, M. An embedded system for position and speed measurement adopting incremental encoders. In: *Conference Record of the 2004 IEEE Industry Applications Conference, 2004. 39th IAS Annual Meeting.* 2004, sv. 2, 1192–1199 vol.2. Dostupné z DOI: 10.1109/IAS.2004.1348564.

46. *PXMC - Portable, highly eXtendable Motion Control library* [online]. Praha: PiKRON, 2023 [cit. 2023-05-13]. Dostupné z: http://www.pxmc.org/.

47. *Robot CRS* [online]. Praha: ČVUT v Praze, Fakulta Elektrotechnická, 2023 [cit. 2023-05-13]. Dostupné z: https://cw.fel.cvut.cz/b221/help/common/robot_crs.

48. PÍŠA, Pavel. *Dokumentace PXMC* [online]. Praha: PiKRON, 2023 [cit. 2023-05-13]. Dostupné z: https://gitlab.com/pikron/sw-base/pxmc/-/tree/master/doc.

49. MOROZOV, Maxim. *Mathematical Model of Lego EV3 Motor* [online]. 2015 [cit. 2023-05-13]. Dostupné z: http://nxt-unroller.blogspot.com/2015/03/mathematical-model-of-lego-ev3-motor.html.

50. FRANKLIN, Gene; POWELL, David; EMAMI-NAEINI, Abbas. *Feedback Control of Dynamic Systems*. Seventh Edition. Upper Saddle River, New Jersey: Pearson Higher Education, Inc., 2015. ISBN 978-0-13-349659-8.

51. *Pidtune: PID tuning algorithm for linear plant model* [online]. Massachusetts: The MathWorks, Inc., 2023 [cit. 2023-05-24]. Dostupné z: https://www.mathworks.com/help/control/ref/dynamicsystem.pidtune.html.

52. SHAW, Andy. *Zdrojové kódy leJOS EV3: Generátor trajektorií* [online]. 2015 [cit. 2023-05-13]. Dostupné z: https://sourceforge.net/p/lejos/ev3/code/ci/master/tree/ev3classes/src/lejos/internal/ev3/EV3MotorPort.java.

53. *Dokumentace GitLabu: GitLab CI/CD* [online]. Nizozemsko: GitLab B.V., 2023 [cit. 2023-05-01]. Dostupné z: https://docs.gitlab.com/ee/ci/.

54. *ODROID-HC2: Home Cloud Two* [online]. GyeongGi, South Korea: Hardkernel [cit. 2023-05-21]. Dostupné z: https://www.hardkernel.com/shop/odroid-hc2-home-cloud-two/.

55. *What is a Container?* [Online]. Palo Alto: Docker, Inc., 2023 [cit. 2023-05-02]. Dostupné z: https://www.docker.com/resources/what-container/.

56. *GitLab Documentation: Run your CI/CD jobs in Docker containers* [online]. San Francisco: GitLab Inc, 2023 [cit. 2023-05-03]. Dostupné z: https://docs.gitlab.com/ee/ci/docker/using_docker_images.html.

57. *Dockerfile reference* [online]. Palo Alto: Docker, Inc., 2023 [cit. 2023-05-03]. Dostupné z: `https://docs.docker.com/engine/reference/builder/`.

58. *What is Docker Hub?* [Online]. Palo Alto: Docker, Inc., 2023 [cit. 2023-05-03]. Dostupné z: `https://www.docker.com/products/docker-hub/`.

59. TRENTINI, Matt. *Build environment for the MicroPython ports using Arm-based microcontrollers* [online] [cit. 2023-05-03]. Dostupné z: `https://hub.docker.com/r/micropython/build-micropython-arm`.

60. *GitLab Documentation: The .gitlab-ci.yml file* [online]. San Francisco: GitLab Inc, 2023 [cit. 2023-05-03]. Dostupné z: `https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yaml.html`.

61. *MicroPython documentation: MicroPython manifest files* [online]. Cambridge: Autoři MicroPythonu, 2023 [cit. 2023-05-03]. Dostupné z: `https://docs.micropython.org/en/latest/reference/manifest.html`.

62. *Zadání soutěžní úlohy "Ping-pong"* [online]. Praha: ČVUT v Praze, Fakulta Elektrotechnická, 2023 [cit. 2023-05-14]. Dostupné z: `https://robosoutez.fel.cvut.cz/zadani-soutezni-ulohy-ping-pong`.

63. *Zadání soutěžní úlohy "Pac-man" (SŠ)* [online]. Praha: ČVUT v Praze, Fakulta Elektrotechnická, 2014 [cit. 2023-05-14]. Dostupné z: `https://robosoutez.fel.cvut.cz/zadani-soutezni-ulohy-pac-man-0`.

64. *Zadání soutěžní úlohy "Pac-man" (ZŠ)* [online]. Praha: ČVUT v Praze, Fakulta Elektrotechnická, 2014 [cit. 2023-05-14]. Dostupné z: `https://robosoutez.fel.cvut.cz/zadani-soutezni-ulohy-pac-man`.

Příloha A

Obsah přiloženého CD

Na přiloženém CD se nacházejí tyto soubory:

- `vanekj19_opencube.pdf` obsahuje tuto bakalářskou práci v elektronické podobě.
- `pc-ev3-sensors.zip` obsahuje konzolovou aplikaci vyvinutou v sekci 3.2.
- `opencube_firmware_repo.zip` obsahuje kopii repozitáře s Open-Cube firmware z doby odevzdání bakalářské práce. Obsah archivu dále popíši níže. Kód je dostupný také ve veřejném repozitáři `https://gitlab.fel.cvut.cz/open-cube/firmware` jako commit `b05f91909f0259569aa220258079265828ee4951`.
- `opencube_micropython_with_pylibs.uf2` je nejnovější sestavenou verzí firmware pro Open-Cube.
- Soubory `linefollower.mp4`, `pingpong.mp4` a `pacman.mp4` zachycují jízdy robotů ze sekcí 3.3.3, 6.1 a 6.2.

Archiv `opencube_firmware_repo.zip` obsahuje celkový zdrojový kód Open-Cube včetně práce jiných studentů. Výstup této bakalářské práce se nachází v těchto cestách:

- Adresář `lib/hw_defs` obsahuje základní definice zapojení Open-Cube pro moduly psané v Pythonu.
- Adresář `lib/ev3` obsahuje Python třídy pro jednotlivé EV3 senzory. Jedná se o nejvyšší vrstvu ovladače 3.4.
- Adresář `micropython/modules/opencube_pindefs/` obsahuje základní definice zapojení Open-Cube pro moduly psané v C. V této BP vznikly pouze definice pro EV3 senzory a motory, pro NXT senzory přidal definice V. Jelínek.

- Adresář `micropython/modules/opencube_uart/` obsahuje abstrakční vrstvu pro sériové linky senzorových portů, tj. nejnižší vrstvu ovladače 3.4. Lze zde také nalézt ovladač PIO UART a příslušný stavový automat.
- Adresář `micropython/modules/ev3_sensors/` obsahuje jádro EV3 protokolu, tj. střední vrstvu ovladače 3.4.
- Soubor `micropython/modules/lib/i2c_locking.c` a příslušný hlavičkový soubor obsahují řešení kolizí na vnitřní I^2C sběrnici popsané v sekci 4.2.1.
- Soubor `micropython/modules/opencube_motors/encoder.c` a příslušný hlavičkový soubor obsahují implementaci měření polohy a rychlosti ze sekce 4.2.
- Adresář `micropython/modules/jv_motors/` obsahuje zbytek podpory motorů (PWM, regulátor, generátor reference, Python API). K oddělení od modulu `opencube_motors` (jehož autorem převážně nejsem) došlo kvůli kolizi s paralelní bakalářskou prací.
- Soubory `micropython/typing/jv_motors.pyi` a `ev3_sensors.pyi` tamtéž obsahují dokumentaci rozhraní, které poskytují příslušné nativní moduly do Pythonu.
- Soubor `micropython/Dockerfile` definuje obraz Docker kontejneru ze sekce 5.2 pro sestavení MicroPythonu.
- Soubor `.gitlab-ci.yml` definuje kroky podniknuté GitLab serverem k sestavení MicroPythonu, viz 5.3.
- Soubor `micropython/frozen_manifest.py` je manifest soubor ze sekce 5.4 určující, které Python soubory se zabudují do firmware.
- V adresářích `programs/pingpong` a `programs/pacman` lze nalézt programy pro testovací roboty 6.1 a 6.2.
- Adresář `programs/examples/ev3` obsahuje soubory s ukázkami použití API pro senzory.