Vyšší odborná škola
a Střední průmyslová škola elektrotechnická
Plzeň, Koterovská 85

ROČNÍKOVÁ PRÁCE

Téma:

Model průmyslového šestiosého robota

Autor práce: Tomáš Kubín
Třída: 3.L
Vedoucí práce: Jiří Švihla
Dne: 30. 4. 2025
Hodnocení:

Vyšší odborná škola
a Střední průmyslová škola elektrotechnická
Plzeň, Koterovská 85

ZADÁNÍ ROČNÍKOVÉ PRÁCE	
Školní rok	2024/2025
Studijní obor	78 – 42 – M/01 Technické lyceum
Jméno a příjmení	Tomáš Kubín
Třída	3.L
Předmět	Kybernetika
Hodnoceno v předmětu	Kybernetika
Téma	Model průmyslového šestiosého robota
Obsah práce	Tvorba řízení (softwarová část) pro model průmyslového kolaborativního robota. • Výběr komponent (20. 12. 2024) • Prototyp řízení (24. 1. 2025) • Finalizace zařízení a napojení na síť (28. 2. 2025) • Tvorba dokumentace (28. 3. 2025)
Zadávající učitel Příjmení, jméno	Švihla Jiří
Termín odevzdání	30. dubna 2025

V Plzni dne: 29. 11. 2024

Mgr. Jan Syřínek, v.r.
Zástupce ŘŠ, zástupce statutárního orgánu
Vedoucí organizace VOŠ, SŠ, DM

Anotace a poděkování

Má ročníková práce se zabývá tvorbou řízení pro model kolaborativního průmyslového robota. Cílem bylo vytvořit ne příliš komplexní, avšak efektivní systém jako celek za pomoci běžně dostupných komponent. Na jiných částech robota pracovali Daniel Hornek a Matěj Svoboda.

Chtěl bych poděkovat firmě Würth Elektronik za poskytnutí pasivních a elektromechanických součástek do desky plošných spojů zdarma, vedoucímu práce Jiřímu Švihlovi za podporu při zpracování, krokové motory a vyfrézování.

Prohlašuji, že jsem tuto práci vypracoval samostatně a použil literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací.

Prohlašuji, že jsem nástroje UI využil v souladu s principy akademické integrity a že na využití těchto nástrojů v práci vhodným způsobem odkazuji.

Nesouhlasím s využitím mé práce učiteli VOŠ a SPŠE Plzeň k výuce.

V Plzni dne 30. 4. 2025 Podpis: .....

Obsah

Úvod

V současné době, kdy se rozvíjí koncept průmyslu 4.0, je trendem výrobní procesy co nejvíce automatizovat, a proto se hojně využívá robotů. Kolaborativní roboti jsou schopni spolupracovat s člověkem za absence většiny bezpečnostních prvků nutných u tzv. dospělých robotů. To mě dovedlo k myšlence zkusit si jednoho postavit, protože v běžném provozu si s nimi člověk hrát nemůže a nejedná se ani o běžně přístupné "hračky."

Podílení na stavbě jednoho je přínosnou zkušeností nejen z kreativního a programátorského hlediska, ale i z oblasti spolupráce a organizace projektu více lidí.

1 Cíle

Mým cílem bylo vyvinout řízení pro model průmyslového robota. Věnoval jsem jak softwarové části řízení, tak i té nezbytné hardwarové.

1.1 Softwarové cíle

Software jsem nechtěl od samého začátku dělat zbytečně komplikovaný, funkčnosti a odladěnosti jsem dával přednost. Programoval jsem s myšlenkou variabilnějšího použití robota, mezi které řadím například modularitu, jiné počty a typy motorů. Tedy jsem neopomenul rozšíření projektu v budoucnu do takové míry, kdy bude v podstatě jen stačit změnit konfigurační soubor.

Cíle bych shrnul následovně:

1. Základní ovládání motorů
2. Komunikace
3. Optimalizace a přidání "nice to have"funkcí

1.2 Hardwarové cíle

Již od začátku jsem chtěl všechno umístit do rozvaděče. Tomu jsem uzpůsobil výběr komponent. To bylo primárním důvodem pro návrh vlastní desky plošných spojů (DPS). Minimalizace byla pro mě nesmírně důležitá, chtěl jsem alespoň vnější konstrukci rozvaděče být schopen vytisknout na klasické 3D tiskárně.

Shrnutí cílů:

1. Výběr komponent
2. Návrh DPS
3. Tvorba rozvaděče

2 Komponenty

2.1 Raspberry Pi Pico 2

Aby bylo zajištěno rychlé ovládání motorů, byl zvolen mikrokontrolér Raspberry Pi Pico 2. Ten slouží výhradně k obsluze vstupů a výstupů. Přistoupil jsem k variantě bez bezdrátových možností, všechna komunikace je totiž řešena spolehlivě drátovou cestou.

Raspberry Pico 2 je vybaveno procesorem RP2350A vlastní výroby - dvě jádra M33 Cortex a frekvence 150 MHz. Na desce je 4 MB flash paměť, která postačí i na náročnější projekty. Paměť RAM je v porovnání s předchozí generací téměř dvojnásobná, tzn. 520 KB. (Raspberry Pi Ltd 2025b)

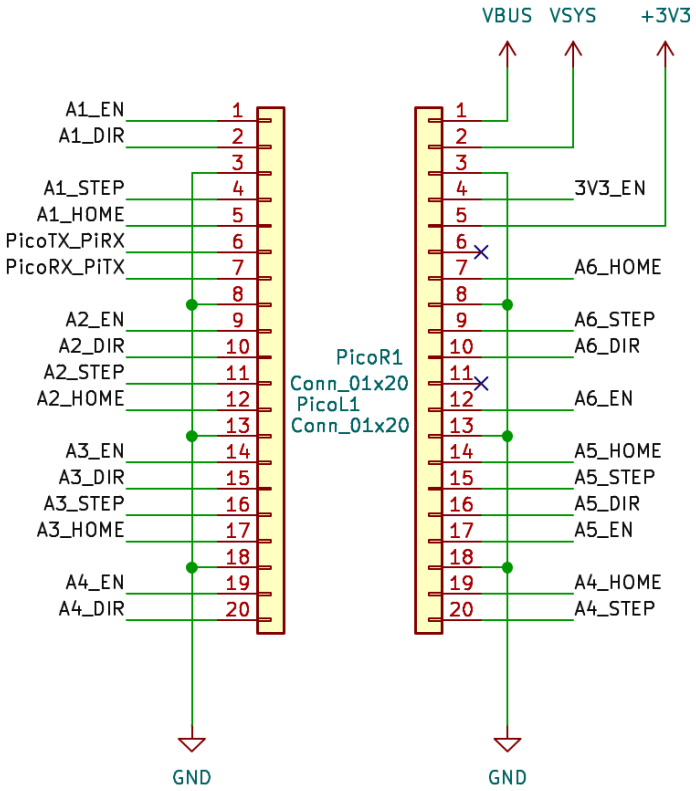
Velkou výhodou této desky je její popularita ve světě. To se odráží na její dostupnosti a v komunitě není obtížné najít pomoc při případných obtížích (hardwarových i softwarových).

Já osobně ji programuji v MicroPythonu, lze však využít i jazyk C a Arduino IDE.

Celkem má 40 pinů, dvacet na každé straně. Rozmístění pinů včetně popisků mnou použitých je na výňatku z přílohy I, na obrázku 2.1. Všech 28 dostupných GPIO pinů je využito.

UART má dva kanály, z nichž jsem použil ten druhý a má pevně dané piny 6 a 7 na levé straně.

Napájení je zajištěno přes VSYS. Mezi zdrojem a VSYS je ještě zapojena schottkyho dioda, aby nedocházelo ke zpětnému napájení.



Obrázek 2.1: Zapojení pinů na Picu [Zdroj: vlastní]

2.2 Raspberry Pi 4

Raspberry Pi 4 není mikrokontrolérem, nýbrž se jedná o relativně výkonný a univerzální jednodeskový počítač. V mém projektu zajišťuje komunikaci s mikrokontrolérem a pendantem.

Mezi jeho hlavní přednosti se v mém případě řadí možnost instalace plnohodnotného operačního systému, což je ideální pro budoucí rozšíření, jako je například hostování vlastní databáze, webového serveru či jiných náročnějších aplikací.

Stejně jako Raspberry Pi Pico je široce rozšířené a dostává se podpory rozsáhlé komunity. Má taktéž 40 pinů, v nichž se od Raspberry Pi Pico velmi liší. Za mě je významnou výhodou více kanálů pro UART. Lze tedy k němu připojit moje Pico, pendant a ještě zůstane velká rezerva. (Raspberry Pi Ltd 2025a)

2.3 Drivery pro krokové motory DRV 8825

Pro řízení krokových motorů je nutné použití driveru. Po konzultaci jsem se rozhodl pro DRV8825 od Texas Instruments. Je určen pro bipolární krokové motory (motor má dvě cívky, tedy z něj vedou 4 dráty, které musíme zapojit) použité v projektu.

Nastavení proudového limitu je možné pomocí malého potenciometru, což chrání motor. Vyšší proudy vyžadují chlazení, alespoň pasivní.

Neméně důležitou vlastností je mikrokrokování (úprava počtu kroků na celou otáčku), to lze nastavit v rozsahu od 1 až po 1/32 (zmenšit jde jen po polovinách).

Rozmístění pinů je na obrázku 2.3 (Bakker 2025).

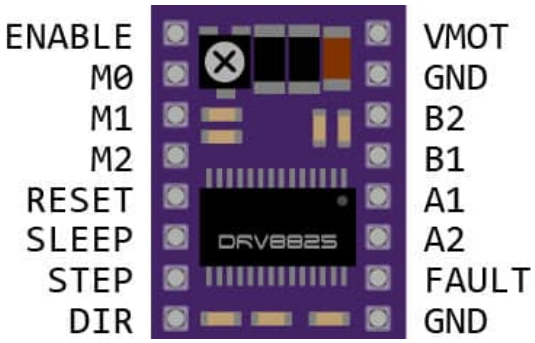
Piny M0, M1, M2 slouží k nastavení mikrokrokování, návod je na DPS.

K pinům A1, A2, B1 a B2 se připojuje vinutí motoru.

Pro funkci je nutné přivést 3 V na piny RESET a SLEEP.

Pin ENABLE je vždy zapojen na výstup, aby se motory daly jednoduše vypnout.

Na VMOT je přivedeno 12 V jako napájení motorů. Driver umí i motory s větším vstupním napětím.

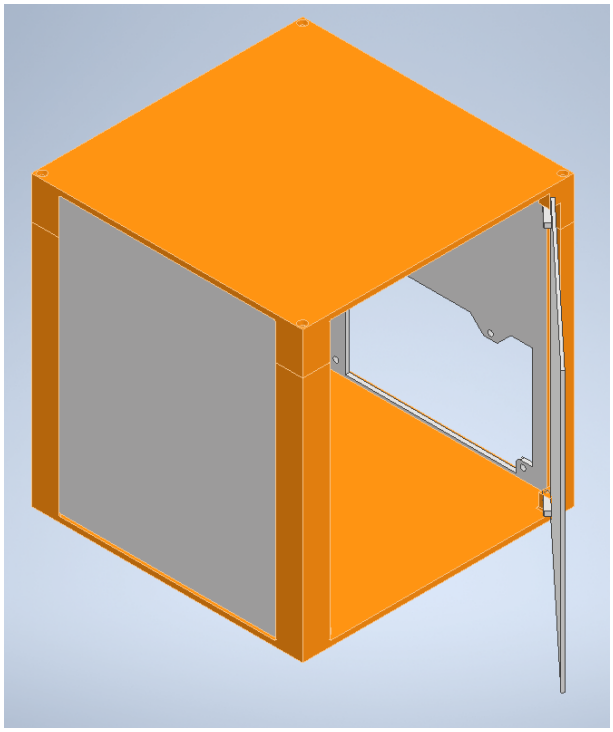


Obrázek 2.2: Rozmísťěný pinů na driveru [Zdroj: makerguides.com]

2.4 Rozvaděč

Rozvaděč jsem vymyslel do jisté míry modulární. Spodní a horní část jsem navrhl tak, aby se dala bez problému vytisknout na 3D tiskárně. Obě části mají výřezy na zasouvací stěny.

Hrany stěn a složité útvary, které by nešly vytvořit jiným způsobem, jsou vyfrézovány do dřevěné desky o tloušťce 3 mm. Díry pro úchyty DPS, Raspberry Pi 4, zdroje jsou dodatečně vyvrtány vrtačkou a tudíž nebudou zahrnuty ve výkresech. Zdrojové soubory pro rozvaděč jsou v GitHub repozitáři (Kubín 2025).



Obrázek 2.3: Rozvaděč ve 3D návrhovém softwaru [Zdroj: vlastní]

3 Návrhový software

Po výběru komponent bylo důležité se rozhodnout, v jakých jazycích a nástrojích budu tvořit.

3.1 Visual Studio Code (VS Code)

Veškerý kód jsem vyvíjel v jazyce Python (resp. MicroPython). Rozhodl jsem se pro populární IDE Visual Studio Code vyvíjené společností Microsoft. Zkoušel jsem předtím často v komunitě doporučovaný Thonny, ale ten mi připadal jako příliš jednoduchý a omezující program.

Mezi nejdůležitější přednosti VS Code řadím hlavně ty zvyšující uživatelský komfort. VS Code má pro mě nesmírně důležité zkratky "Ctrl+Tab"a skvěle fungující "Tab", které bohužel stále v dnešní době moc programů neumí. Nabízí možnost členit kód do regionů, které lze skrýt, což je výrazná výhoda pro lepší orientaci.

Na obrázku 3.1 je prostředí s otevřeným kódem. Na obrázku 3.2 je detail pravého horního rohu, kde se v přehledu ukazují názvy regionů, případně se zvýrazní stejné proměnné, když na nějaké stojíte.

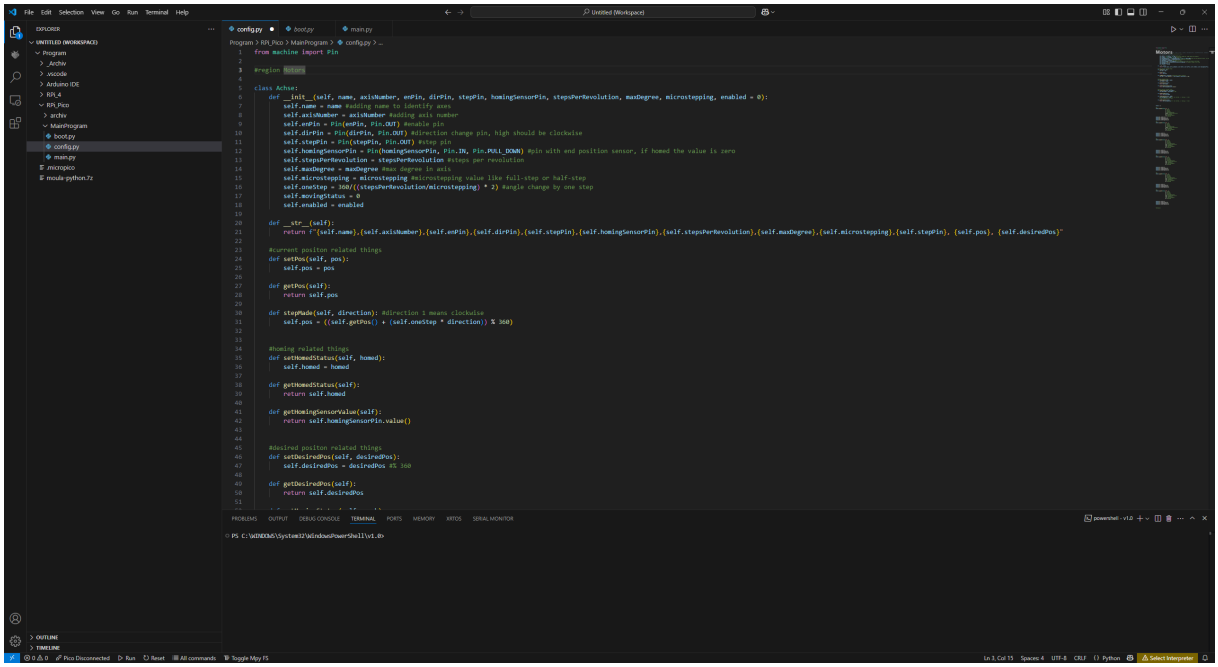
Do VS Code jde doinstalovat spoustu rozšíření, kterých jsem samozřejmě využíval.

Primárně tyto:

1. Micro Pico (Random Nerd Tutorials 2025)

   Tento doplněk umožňuje programovat Raspberry Pi Pico. Umožňuje i ukládání a mazání souborů z Pica. Slouží také ke spouštění kódů na Picu bez nutnosti je mít na něm uložené a čtení dat z konzole.

2. Remote Explorer



Obrázek 3.1: VS Code s otevřeným python skriptem [Zdroj: vlastní]

Obrázek 3.2: Detail pravého horního rohu z VS Code [Zdroj: vlastní]

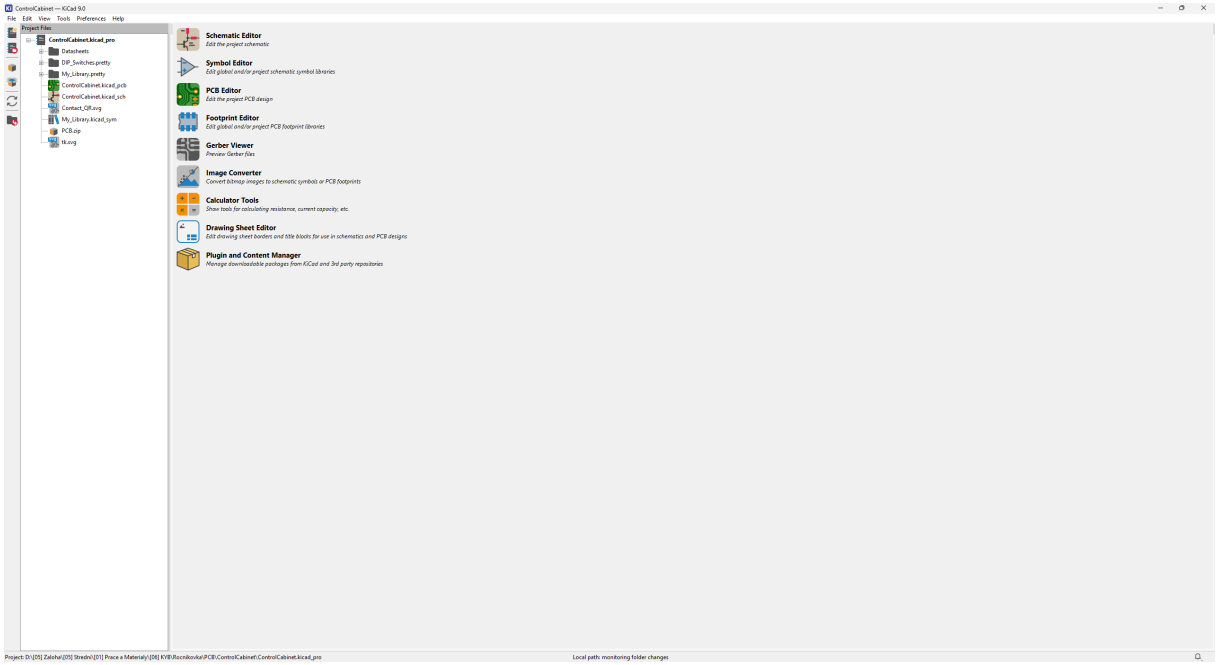
Jedná se o doplněk, který se prostřednictvím SSH připojí na vzdálenou stanici, my mu specifikujeme, kterou složku má vždy otevřít, tou procházíme a otevíráme soubory s kódem, které po každé změně musíme uložit a přes terminál ve spodní části obrazovky spustit.

3.2 KiCad

KiCad je volně dostupným programem pro návrh desek plošných spojů (DPS). Já jakožto začátečník oceňuji jeho jednoduchost a hned jsem se s ním sžil.

Jedná se v podstatě o balík různých nástrojů, do kterých je i členěn viz. obrázek 3.4, a ty pospolu fungují jako celek. Mezi mnou nejvíce využívané patří schematický, PCB a footprint editor.

KiCad umožňuje instalaci doplňků, které mohou zjednodušit export potřebných souborů pro výrobu specializovanou firmou. Můj projekt je na GitHubu (Kubín 2025).



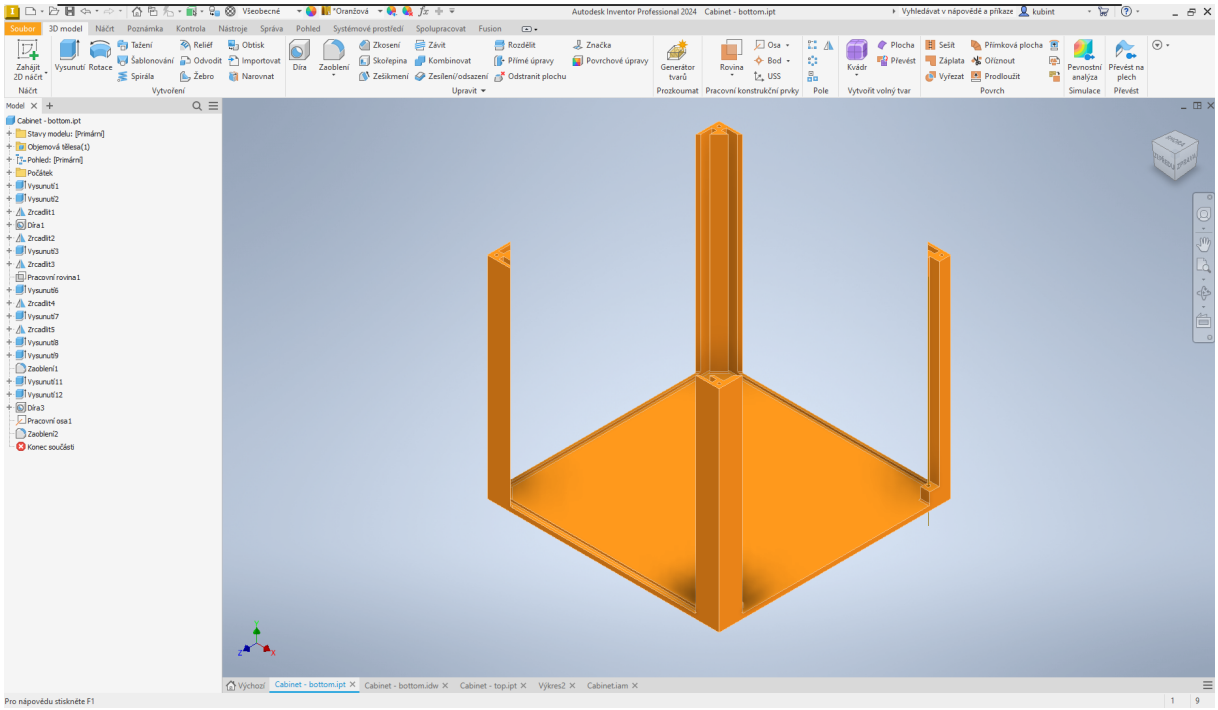
Obrázek 3.3: Pohled na základní prostředí KiCadu s otevřeným projektem [Zdroj: vlastní]

3.3 Autodesk Inventor

3D modely pro rozvaděč jsem vytvářel v profesionálním programu Autodesk Inventor. Jedná se o program, který využívá naše škola ve výuce. Proto mi nepřišel cizí a odpadlo učení se něčeho nového.

Kromě tvorby 3D modelů umožňuje i vytváření sestav a výkresů. Výkresy se nechají všelijak upravovat, vlastní razítko, rámečky atd.

Nativně podporuje export modelů do .stl či .stp souborů pro tisk.



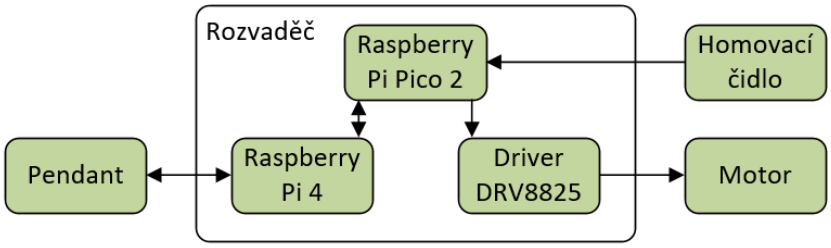
Obrázek 3.4: Pohled na základní prostředí Inventoru s otevřeným projektem [Zdroj: vlastní]

4 Komunikace

Komunikace mezi oběma Raspberry a pendantem je dělána přes UART. Jedná se o jednoduchý asynchronní komunikační protokol, kterému stačí jen dva vodiče. Po jednom jedna strana odesílá data (Tx) a druhá přijímá (Rx), u druhého drátu je směr komunikace opačný.

UART dovoluje nastavit přenosovou rychlost (baudrate), jinými slovy počet bitů za sekundu (bps). Já osobně jsem se rozhodl pro 115 200 bps, jedná se o kompromis mezi rychlostí a spolehlivostí (ani na jedno si nemohu stěžovat)

V pythonu je jeho použití v kódu velmi jednoduché, netřeba instalovat dodatečné knihovny.



Obrázek 4.1: Komunikační diagram [Zdroj: vlastní]

4.1 Struktura dat

V rámci přehlednosti a rozumné kontroly nad posílanými daty jsem se rozhodl posílat si vždy stejně dlouhé pole bajtů. Jeho struktura je popsána v tabulce 4.1. Struktura je všude stejná jak pro Raspberry Pi Pico 2 tak pro pendant, jen některé hodnoty mají jiný význam (jednou aktuální pozice a jindy zas požadovaná).

Rozhodl jsem se pro 24 bajtů, nejedná se sice pěkně o mocninu dvou, ale nechtěl jsem dělat zbytečně velkou rezervu. Rezervu jsem přidal pro jednoduchost přidávání budoucích dat, odpadá potom nějaké to psaní navíc na všech stranách, takhle stačí jen menší úprava.

Pořadí bytu	Jeho obsah
0	counter
1	1. byte požadované pozice první osy
2	2. byte požadované pozice první osy
3	homing k první ose
4	1. byte požadované pozice druhé osy
5	2. byte požadované pozice druhé osy
6	homing k druhé ose
7	1. byte požadované pozice třetí osy
8	2. byte požadované pozice třetí osy
9	homing k třetí ose
10	1. byte požadované pozice čtvrté osy
11	2. byte požadované pozice čtvrté osy
12	homing k čtvrté ose
13	1. byte požadované pozice páté osy
14	2. byte požadované pozice páté osy
15	homing k páté ose
16	1. byte požadované pozice šesté osy osy
17	2. byte požadované pozice šesté osy
18	homing k šesté ose
19	seznam povolených os
20-22	rezerva
23	counter

Tabulka 4.1: Struktura posílaných dat z Raspberry Pi 4

První a poslední bajt obsahují vždy stejný counter (proměnná, která se čítá s každým odesláním), který slouží k detekci přijatých nových dat a zároveň kontrole konzistence (všechna data přišla v pořádku).

Natočení osy může být od 0° do 360°, zatímco maximální hodnota jednoho bajtu může být 255. Proto jsem musel úhel natočení pomocí bitových operací rozdělit do dvou bajtů.

Následuje bajt, ve kterém jsou věci k homingu. Zatím se jedná spíše o neefektivně využitý bajt, protože se v něm posílá jen jestli je osa homovaná respektive jestli má povolení se homovat. V plánu je ale budoucí rozšíření, kde počítám s více parametry, proto tam tedy nechávám jednotlivé bajty.

V 19. bajtu každý bit až na poslední je přiřazen jedné ose. Hodnota bitu TRUE znamená, že osa je povolena, zatímco FALSE znamená, že je zakázána.

V budoucnu je v plánu strukturu upravit na posílání i desetinných čísel v případě natočení motorů.

5 Kód na Raspberry Pi Pico

Veškerý kód tvořený pro Raspberry Pi Pico 2 jsem na něj uložil ve třech souborech. A je dostupný na GitHub repozitáři (Kubín 2025)

5.1 boot.py

Vlastností firmwaru je, že se soubor *boot.py*, je-li k dispozici, spustí automaticky po připojení napájení. Já osobně v něm jenom zablikám s LED, ať vím, že Pico právě naběhlo.

5.2 main.py

Po dokončení kódu z *boot.py* se automaticky spustí *main.py*. V mém případě se jedná tedy o soubor, ve kterém je uložen hlavní kód.

5.2.1 Vlákna

Z důvodu rychlosti (stačil jen jeden motor a byl poznat rozdíl) jsem musel implementovat multithreading. Vytvořil jsem dvě vlákna, to první se stará o komunikaci s Raspberry Pi 4 přes UART a to druhé řeší pohyb motorů. Vlákna spolu sdílejí proměnné.

Napřed je třeba import knihovny `_thread`. V kódu mám vytvořené ony dvě funkce s cyklem while. Vlákna se startují velmi jednoduše ve dvou řádcích:

```
second_thread = _thread.start_new_thread(control_thread, ())

uart_thread()
```

5.3 config.py

Soubor *config.py* je tvořen v podstatě jako knihovna. Je importován na začátku souboru *main.py*.

Je v něm napsána poměrně rozsáhlá třída, která obsahuje veškeré vlastnosti pro jednotlivé motory na každé ose.

Třída se v Pythonu vytváří následovně:

```
class Achse:
```

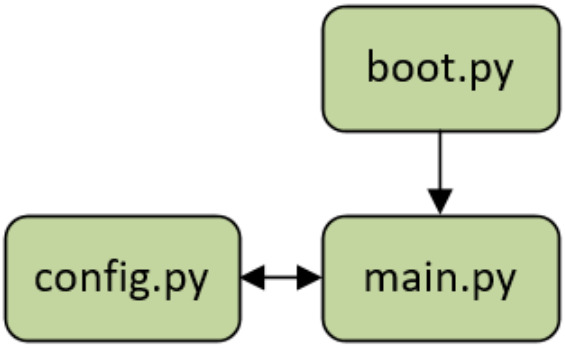
To ale samo o sobě nestačí, dovnitř začneme psát různé funkce. Asi tou nejdůležitější je `__init__`, její vstupními parametry jsou v mém případě:

self, name, axisNumber, enPin, dirPin, stepPin, homingSensorPin, stepsPerRevolution, maxDegree, microstepping, enabled = 0.

Self je povinný argument. Vstupní proměnné obsahující slovo Pin jsou přímo číslo pinu, na který je daná věc zapojena. Ostatní vstupní hodnoty jsou vlastnosti specifické pro každou osu.

Dále obsahuje 11 funkcí, které zahrnují mimo jiné i nastavení požadované pozice, čtení aktuální pozice a veškeré věci týkající se homování.

Po deklaraci třídy je potřeba vytvořit si proměnné (pro každou osu jednu). Pro lepší práci s osami jsem si napřed vytvořil pole, do kterého jsem vložil proměnnou od každé osy. Na obrázku 5.2 je vidět deklarace pole a připojení první osy k němu.



Obrázek 5.1: Diagram spouštění kódu na Pico 2 [Zdroj: vlastní]

```
Achsen = []

#A1
Achsen.append(Achse("A1", 1,
                    0, #enPin
                    1, #dirPin
                    2, #stepPin
                    3, #homingSensorPin
                    12800, #stepsPerRevolution
                    90, #maxDegree
                    1/4, #microstepping
                    1)) #enabled
Achsen[0].setPos(0)
Achsen[0].setDesiredPos(0)
Achsen[0].setHomedStatus(0)
Achsen[0].setMovingStatus(0)
```

Obrázek 5.2: Připojení proměnné pro první osu do pole [Zdroj: vlastní]

6 Kód na Raspberry Pi 4

Situace se soubory je tentokrát mnohem jednodušší. Vše je psáno v souboru *main.py*. Na rozdíl od Raspberry Pi Pico jsem musel ještě v pár jednoduchých krocích nastavit spouštění skriptu po startu. Kód je též na GitHubu (Kubín 2025).

6.1 Autostart skriptu

Postup pro nastavení automatického spouštění skriptu:

1. Spustíme terminál (lze i vzdáleně pomocí ssh, já preferuji program KiTTY)
2. Spustíme program crontab s parametrem e

   `crontab -e`
3. Jsme dotázáni, v jakém editoru chceme editovat, volba je čistě na Vás
4. Pro spuštění kódu hned po startu přidáme do souboru

   `@reboot python3 /home/Code/main.py`
5. Soubor uložíme a můžeme restartovat pro ověření funkčnosti. (Brown 2025)

6.2 Vlákna

Kód je i zde rozdělen do dvou vláken, jen pomocí knihovny `threading`. Implementace je potom též poměrně jednoduchá a ve dvou řádcích:

```
Thread(target = comm_Control).start()

Thread(target = comm_Pendant).start()
```

Vlákna sdílejí některé proměnné. Jedná se jen o proměnné třídy s osami.

Spuštěné funkce si jsou velmi podobné, liší se jen drobnostmi. Já tedy vysvětlím jen princip funkčnosti. Zakládal jsem si na tom, že zařízení jako celek bude ve stylu plug and play, člověk se nebude muset o nic starat a po zapnutí napájení se komunikace sama naváže.

Jsou použity dvě hlavní proměnné pro správu **counter** a **lastTime**, kdy nutnost counteru již byla vysvětlena a proměnná **lastTime**, ukládající čas poslední komunikace, je nezbytně nutná pro detekci výpadku spojení (timeout).

Po naplnění send bufferu se buffer odešle. Raspberry chvíli čeká na odpověď. Když přijdou nějaká data, zkontroluje jejich validitu. Jestli jsou data validní, zpracuje je a inkrememntuje **counter** pro další interakci.

Když data validní nejsou, tak posílá a stále na ně čeká. Pokud již dlouho k žádné komunikaci nedošlo, **counter** se vyresetuje a Raspberry se spojení snaží navázat znovu.

Závěr

Cílem mé práce bylo udělat řízení pro model průmyslového šestiosého kolaborativního robota.

Komponenty byly vybrány vhodně a integrovány. Na mou první takovou zkušenost si myslím, že množství chyb (ne perfektně povedených věcí) je až příliš nízké. Ale i tak jsem určitě pro příště poučen a vím, co dělat jinak nebo lépe.

Vytvořený projekt lze použít jako základ pro budoucí rozšíření jak robota, tak i softwarové stránky.

Práce mě seznámila s novými věcmi, získal jsem zkušenosti z oblasti softwaru a návrhu vlastní desky plošných spojů.

Seznam obrázků

Bibliografie

Bakker, Benne de (2025). "How to control a stepper motor with DRV8825 driver and Arduino". In: *Maker Guides*. DOI: https://www.makerguides.com/drv8825-stepper-motor-driver-arduino-tutorial/.

Brown, Korbin (2025). "Automating Raspberry Pi: How to Autostart Programs". In: *Linux Config*. DOI: https://linuxconfig.org/automating-raspberry-pi-how-to-autostart-programs.

Kubín, Tomáš (2025). $RP_Kubin_Hornek_Svoboda/Code/Rizeni/$. URL: https://github.com/Kybernetika-SPSE/RP_Kubin_Hornek_Svoboda/tree/main (cit. 27. 04. 2025).

Random Nerd Tutorials (2025). "Programming Raspberry Pi Pico with VS Code and MicroPython". In: *Random Nerd Tutorials*. DOI: https://randomnerdtutorials.com/raspberry-pi-pico-vs-code-micropython/.

Raspberry Pi Ltd (2025a). *Raspberry Pi Hardware*. URL: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html (cit. 27. 04. 2025).

— (2025b). *RP2350 Datasheet*. URL: https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf (cit. 27. 04. 2025).

Přílohy

Příloha I: Schéma zapojení

Příloha II: Schéma DPS

Příloha III: Spodek rozvaděče

Příloha IV: Vršek rozvaděče

Příloha V: Levá + zadní stěna rozvaděče

Příloha VI: Pravá stěna rozvaděče

Příloha VII: Rozvaděč