ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická
Katedra elektroniky a informačních technologií

BAKALÁŘSKÁ PRÁCE

Ovládací systém pro víceosý pozicionér
Control system for multi-axis positioner

Autor práce: **Martin Mixán**
Vedoucí práce: **Ing. Pavel Broulím, Ph.D.**

2022

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin MIXÁN**
Osobní číslo:	**E18B0218P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Téma práce:	**Ovládací systém pro víceosý pozicionér**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

1. Prostudujte možnosti sestavení a ovládání tříosého pozicionéru.
2. Navrhněte modul pro ovládání posunů jednotlivých os s jednotným připojením k počítači přes USB rozhraní.
3. Pro navržený modul zpracujte ovládací software.
4. Výsledné řešení otestujte a diskutujte.
5. Navrhněte případná vylepšení, např. změna komunikačního rozhraní s ovládacím počítačem.

Rozsah bakalářské práce: **30 – 40**
Rozsah grafických prací: **dle doporučení vedoucího**
Forma zpracování bakalářské práce: **elektronická**

Seznam doporučené literatury:

1. PINKER, Jiří. *Mikroprocesory a mikropočítače*. Praha: BEN – technická literatura, 2004. ISBN 80-7300-110-1.
2. PINKER, Jiří, POUPA, Martin. *Číslicové systémy a jazyk VHDL*. Praha: BEN – technická literatura, 2006. ISBN 80-7300-198-5.
3. Online dokumentace k produktům dle konkrétní realizace.

Vedoucí bakalářské práce: **Ing. Pavel Broulím, Ph.D.**
Research and Innovation Centre for Electrical Engineering

Datum zadání bakalářské práce: **8. října 2021**
Termín odevzdání bakalářské práce: **26. května 2022**

Západočeská univerzita v Plzni
Fakulta elektrotechnická
L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 8. října 2021

Abstrakt

Cílem práce je sestavit víceosý pozicionér pro vzdálené ovládání polohy detektoru ionizujícího záření, který je uzavřen ve stíněné komoře. Zařízení je navrženo užitím IoT prvků s využitím jazyka Python a automatizačního systému OpenHAB. K řešení je využit mikropočítač Raspberry Pi 4 B a komerčně dostupné drivery pro ovládání motorů (Pololu Tic T500). Celý pozicionér se sestává ze 3 os, kdy jedna osa je z důvodu tuhosti ukázkově řešena ze 2 posuvů řízených synchronně. Celkově k systému lze připojit až 6 motorů při využití 2 motorů na osu. Závěrem jsou shrnuty vlastnosti navrženého řešení a možné kroky pro budoucí vylepšení systému.

Klíčová slova

Víceosý pozicionér, CNC, IoT, elektrický manipulátor, MQTT, Pololu Tic T500, Raspberry Pi, Python, OpenHAB, Openhabian, HANSPOSE 17HS3401S, HANSPOSE 23HS4128, krokový motor, I^2C

Abstract

The aim of this work is to assemble a multi-axis positioner for remote control of the position of the ionizing radiation detector, which is enclosed in a shielded chamber. The device is designed using IoT elements using Python language and OpenHAB automation system. The solution uses a Raspberry Pi 4 B microcomputer and commercially available drivers for motor control (Pololu Tic T500). The whole positioner consists of 3 axes, with demonstration of using 2 step motors per one axis driving synchronously due to toughness. In total, up to 6 motors can be connected to the system using 2 motors per axis. Finally, the features of the proposed solution and possible steps for future system improvements are summarized.

Key Words

Multi-axis positioner, CNC, IoT, electric manipulator, MQTT, Pololu Tic T500, Raspberry Pi, Python, OpenHAB, Openhabian, HANSPOSE 17HS3401S, HANSPOSE 23HS4128, stepper motor, I^2C

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce. Dále prohlašuji, že použitý software je legální.

V Plzni dne 21.5.2022 .....

Vlastnoruční podpis: Martin Mixán

Obsah

Seznam symbolů a zkratek

Značka	Popis
CNC	Počítačem řízený obráběcí stroj (Computer Numerical Control)
SSH	Zabezpečený protokol, pomocí SSH lze ovládat a spravovat vzdálené počítače (Secure Shell)
OOP	Objektově orientované programování (Object-Oriented Programming)
MQTT	Protokol pro výměnu zpráv mezi zařízeními, často využívaný v IoT (Message Queuing Telemetry Transport)
IoT	Internet věcí (Internet of Things)
TCP/IP	Hlavní protokol celosvětové internetové sítě (Transmission Control Protocol/Internet Protocol)
QoS	Kvalita služeb v protokolu MQTT (Quality of Service)
I^2C	Počítačová sériová sběrnice vyvinutá firmou Philips (Inter-Integrated Circuit)
localhost	V počítačové terminologii znamená localhost odkaz na právě používaný počítač
SDA	Datový linka v I^2C sběrnici (Synchronous Data)
SCL	Hodinový signál v I^2C sběrnici (Synchronous Clock)
TX	Vysílání (Transmission)
RX	Přijímání (Receive)
RC	Rádiové ovládání (Radio Controlled)
PC	Osobní počítač (Personal Computer)
RAM	Paměť s náhodným přístupem (Random Access Memory)
GPIO	Univerzální vstupní/výstupní pin (General-Purpose Input/Output)
TTL	Tranzistorově-tranzistorová logika (Tranzistor-Transistor-Logic)
DIR	Směr (Direct)
RPI	Raspberry Pi

1 Úvod

Tato práce se zabývá hardwarovým a softwarovým řešením víceosého pozicionéru. Ten je zamýšlen na vzdálenou změnu polohy detektoru ionizujícího záření, z důvodu uzavření ve stíněné komoře, vzdáleného ovládacího pracoviště, nutnosti změny polohy detektoru při aktivním svazku ionizujícho záření atd.

Současná situace je taková, že dostupné komerční řešení se pohybují ve vysoké cenové hladině vzhledem k atypické úloze. Z důvodu vhodného možného rozšíření a úspory finančních prostředků, bylo přistoupeno k vlastní variantě řešení. Podobné technologie a principy se však používají například u CNC stojů a 3D tiskáren, které, co se týče požadavků na libovolné rozšiřování, otevřenost systému pro budoucí inovace a ovládací softwary, neodpovídají povaze úlohy, kterou má tato práce řešit.

Cílem práce je vylepšit stávající konstrukci (prototyp), bez přímé návaznosti na současné řešení. Stávající řešení, které využívá pouze jeden pojezd na každou osu, má určité problémy, které je žádoucí eliminovat v rámci této bakalářské práce. Cílem práce je tedy vytvořit hardwarový prototyp na menších pojezdech s možností jednoduchého rozšíření o větší konstrukce. K tomuto řešení bude vytvořen i příslušný software, který bude jednotlivé pojezdy s motory ovládat.

Vytvořený prototyp víceosého pozicionéru bude mít 3 hlavní osy (X, Y, Z), z čehož minimálně jedna osa (v tomto případě osa X), má být řešena dvěma synchronními pojezdy, pro větší tuhost a zatížitelnost. Software pro ovládání bude navrhnut pro tři synchronní osy, tedy pro 6 motorů, a to pro budoucí rozšiřitelnost bez softwarových úprav. Bude schopný pracovat v režimu různého počtu os, kdy každá může mít jeden nebo dva motory s pojezdy. Celkem bude k dispozici 26 kombinací zapojení (každá osa samostatně nebo různá kombinace os, v režimu jednoho nebo dvou motorů). Ovládací systém využije připojení přes síťové prvky (wifi, ethernet) a prvky z internetu věcí (Internet of Things - IoT), to povede k lepší škálovatelnosti a i budoucí rozšiřitelnosti celého systému. Celý koncept je navrhován tak, aby v budoucnu šlo naprogramovat nové uživatelské rozhraní dle budoucích potřeb s využitím API, které autor navrhl, s možností využívat cloudové IoT služby pro vzdálené ovládání.

2 Druhy pohonů a využití pozicionérů

2.1 CNC stroje

CNC stroj je zkratka vycházející z anglického názvu (Computer Numerical Control), to česky znamená „počítačem řízený stroj“. Obecně se zkratka CNC používá u počítačově řízených obráběcích strojů, které jsou automaticky řízené. CNC stroj zpracovává materiál podle programovatelných pokynů tak, aby splňoval dané specifikace a nebylo nutné přímé manuální řízení operátorem. Tyto stroje mají nejčastěji 1 až 5 os řízených současně. Mají dva základní druhy příkazů, těmi jsou G kódy a M kódy, které jsou posílány do CNC stroje sekvenčně. Samotný obráběcí program (kód, sekvence) lze psát manuálně nebo jej nechat vygenerovat příslušným softwarem. Automatizace pomocí CNC strojů přinesla efektivnější, produktivnější, a především také přesnější obrábění. S tímto souvisí také snížení výrobních nákladů a ceny výroby. [1][2]

2.2 3D tiskárny

3D tiskárny určitým způsobem navazují na samotné CNC stroje. Na rozdíl od nich, výrobek neobrábí, ale tisknou. 3D tiskárny jsou speciální tiskárny uzpůsobené pro tisk 3D objektů.

Zajímavé možnosti 3D tiskáren:

- tisk modelů, figurek, různých nástrojů;
- tisk jídla;
- tisk nemovitostí;
- sebereplikace (vytištění dílu na sestavení další 3D tiskárny, dnes již cca 70 % dílů pro sestavení tiskárny lze vytisknout);
- tisk lidských orgánů nebo implantátů;
- tisknutelná elektronika atd. [3][4]

2.3 Manipulátory

Manipulátory obecně mohou sloužit pro jakoukoli manipulaci věcí, nástrojů, osob atd. Pomocí manipulátorů lze například automatizovat sklady v logistice, vytvořit automatické bary, nebo je používat jako zařízení pro polohování měřících zařízení v nebezpečných oblastech (detektor ionizujícího záření, deaktivace výbušnin, práce na vysokém napětí atd.).

3 Použité technologie

V následující části jsou popsány softwarové (SW) a hardwarové (HW) prostředky použité pro návrh a sestavení celého zařízení.

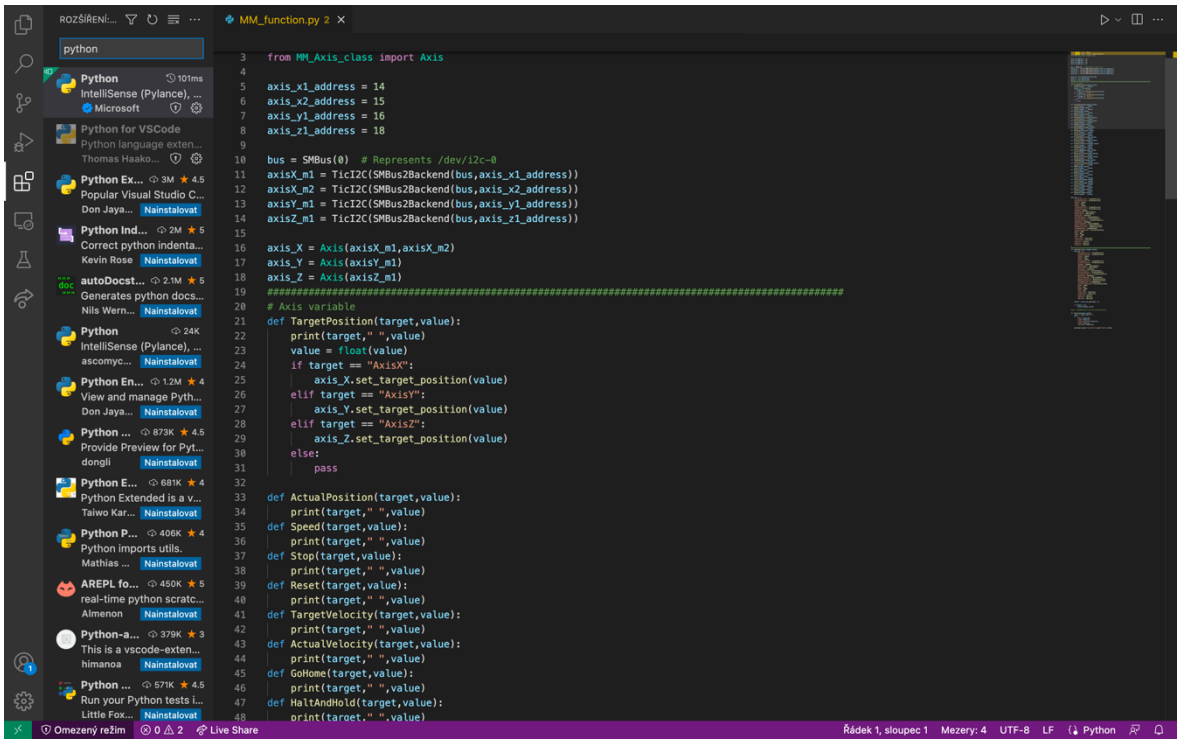
3.1 Softwarové prostředky

3.1.1 Vývojové prostředí Visual Studio Code

Visual Studio Code je bezplatný nástroj pro vytváření, úpravu a ladění zdrojového kódu. Nejpoužívanější programovací jazyky dostupné v tomto nástroji jsou například Python, Java, C/C++, JavaScript, .NET, C#, PHP, YAML, HTML/CSS.

Visual Studio Code se umí připojovat přes SSH (Secure Shell) protokol, který je potřeba při vzdáleném přístupu k mikropočítači Raspbery Pi, použitého na řízení celého systému. Pomocí SSH protokolu je možné se vzdáleně připojit k jinému počítači nebo serveru a programovat, nastavovat, upgradovat nebo kompletně ovládat celý počítač.

Visual Studio Code zvýrazňuje klíčová slova v kódu různými barvami, což pomáhá snadno identifikovat různé části programu, jako jsou třídy, metody, funkce a různé typy proměnných (číslo, řetězec, bool atd.). Nástroj také poskytuje návrhy na dokončení řádků kódu a rychlé opravy běžných chyb syntaxe. Lze využít vestavěného debuggeru (ladícího nástroje), pro ladění kódu. Visual Studio Code lze nainstalovat na Windows, Linux i Mac. Visual Studio Code je vyvíjen společností Microsoft a má širokou škálu rozšíření, které lze snadno doinstalovat přímo v prostředí. Vznik tohoto nástroje je datován v roce 2015. [5]



Obr. 1 Vývojové prostředí Visual Studio Code

3.1.2 SSH

Secure Shell (SSH) je zabezpečený protokol, který vznikl, jako reakce na špatně zabezpečené protokoly a příslušné služby, jako například Telnet (teletype network). Pomocí SSH lze ovládat a spravovat vzdálené počítače, a dokonce přenášet a kopírovat soubory. Využívá se zejména pro úkony, které nevyžadují práci s grafickým rozhraním. Secure Shell protokol byl vyvinut v roce 1995. [6][7]

3.1.3 Python

Python je hybridní jazyk (nebo také multiparadigmatický), to znamená, že umožňuje při psaní programů používat:

- objektově orientované paradigma (OOP) – snaha o přiblížení se k reálnému světu při programování (komunikace a interakce mezi jednotlivými objekty);
- procedurální – přesná posloupnost příkazů (procedura, algoritmus);
- funkcionální – výpočet jako vyhodnocení matematických funkcí, jedna funkce obsahující vstupní parametry mající jediný výstup. [8]

Někdy bývá zařazován mezi takzvané skriptovací jazyky. Jeho možnosti jsou ale větší. Python byl navržen tak, aby umožňoval tvorbu rozsáhlých, plnohodnotných aplikací včetně grafického uživatelského rozhraní. Kód programu v Pythonu je ve srovnání s jinými jazyky krátký a dobře čitelný. Python je jako programovací jazyk poměrně jednoduchý na učení,

k tomu například přispívá i dělení programových bloků pouze pomocí odsazování, nikoli za použití středníků, jak je zvykem u jiných programovacích jazyků. To má za následek, že kód už z principu musí být dobře formátovaný a nejde zapomenout na středník na konci řádku, který se často špatně odhaluje při nefunkčnosti kódu. Python vyniká zejména v efektivitě psaní kódu, která je znát na malých programech, ale i na velkých týmových projektech. S vysokou produktivností a efektivitou souvisí i velká dostupnost a snadná použitelnost široké škály knihoven, umožňujících snadné řešení úloh z řady oblastí, které již někdo vytvořil a optimalizoval. Python se snadno vkládá do jiných aplikací (embedding). Tím lze doplňovat pružnost aplikacím psaných v jiných jazycích. Naopak jiné aplikace nebo knihovny mohou implementovat rozhraní, které umožní jejich použití v Pythonovském modulu, Pythonovský program je tedy může využít jako dostupný modul (extending). Nebezpečnou vlastností Pythonu je, že obsahuje i nedokumentované funkce a lokální proměnné, které mohou být zneužity pro spuštění příkazu v operačním systému. Pro Python je dostupný vlastní repositář balíčků s knihovnami, PyPI, který podporuje snadnou instalaci balíčků programem pip. Další nevýhodou může být samotná rychlost Pythonu v porovnání s ostatními programovacími jazyky jako například C, může být i několikanásobně pomalejší dle typu úlohy. [9]

Python lze použít například pro tyto typy úloh:

- vytváření webových aplikací;
- k databázovým systémům;
- čtení a úprava souborů;
- zpracování velkých dat a provádění složité matematiky;
- rychlé prototypování nebo vývoj softwaru připraveného k produkci. [10]

3.1.4 Internet věcí (Internet of Things – IoT)

Internet věcí (IoT) je systém vzájemně propojených počítačových zařízení, mechanických a digitálních strojů, předmětů, zvířat nebo lidí, s jedinečnými identifikátory (UID), se schopností přenášet data po síti, aniž by bylo nutné zásahu člověka. Jsou to objekty a zařízení, které jsou vybaveny elektronikou, softwarem, senzory, pohyblivými částmi a síťovou konektivitou, která umožňuje těmto zařízením se propojit a vyměňovat si data. V síti IoT může být naprosto cokoliv, ať už člověk se srdečním implantátem, automobil, hospodářské zvíře, domácí mazlíček, domácí spotřebiče atd. Nutná podmínka je, aby to byl

objekt, který je schopen přenášet data po síti a lze mu přiřadit adresu internetového protokolu (IP).

Ekosystém IoT slouží ke shromažďování, odesílání a k práci s daty. Lze sdílet data z různých senzorů a posílat je na cloud k analýze, nebo je analyzovat lokálně. Zařízení mohou komunikovat i mezi sebou pro vzájemnou kooperaci. [11][12]

3.1.5 Message Queuing Telemetry Transport (MQTT)

MQTT je technologie, na které zásadním způsobem tato práce stojí, a je to významný prvek, díky kterému bude zařízení velmi snadno škálovatelné a modifikovatelné. Je to univerzální protokol pro cloudové a IoT aplikace. MQTT je otevřený, jednoduchý a nenáročný protokol pro předávání zpráv mezi klienty prostřednictvím centrálního bodu, MQTT brokeru. [13]

Základní myšlenka

MQTT protokol vznikl na základní myšlence, že (obecně) snímače, senzory, spínače atd., nemají určovat, co se má stát na základě změřené hodnoty, ale mají pouze odeslat hodnoty do centrálního místa. Zařízení nebo systémy, které na základě této hodnoty mají vykonat nějakou činnost, se tak musejí přihlásit k odběru těchto hodnot a na jejich základě pak příslušně reagovat. Komunikace může být obousměrná a zařízení mohou posílat i zpětnou vazbu, například relé na příkaz sepnutí, může poslat zprávu o tom, že je skutečně sepnuté, pomocí pomocného kontaktu. Touto mezivrstvou, pomocí MQTT brokera, lze zajistit oddělení části zpracování (backendu) a uživatelského rozhraní (frontendu) a vyvíjet je nezávisle na sobě bez přímého napojení. [14][15]

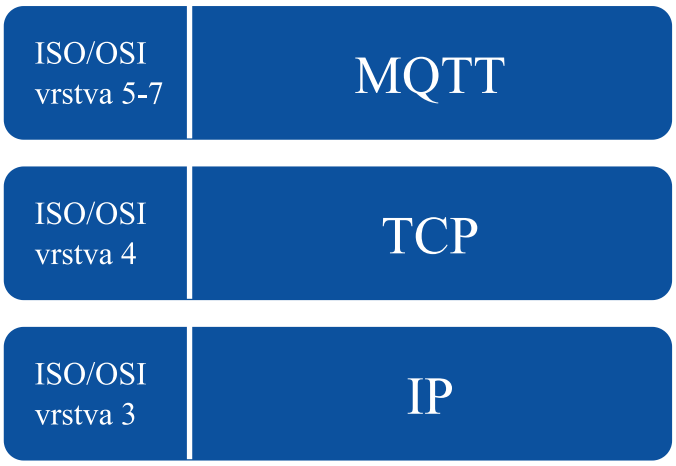
Vlastnosti protokolu MQTT

- Asynchronní protokol;
- kompaktní zprávy;
- provoz v podmínkách nestabilního připojení;
- podpora několika úrovní kvality služeb (QoS);
- snadná integrace nových zařízení;
- protokol MQTT je postavený nad TCP/IP a jako výchozí port používá 1883 (alternativně 8883, při připojení přes SSL). [14][15]

Přenosový model

Protokol MQTT definuje pouze strukturu zpráv, které se přenášejí. Nedefinuje, ale způsob přenosu. Pro přenos využívá protokol TCP/IP, který funguje na LAN nebo WAN

internetové síti. MQTT tvoří pouze aplikační vrstvu ISO/OSI modelu, má tak jednoduchou strukturu. Vzhledem k tomu, že využívá běžné internetové rozhraní, lze jej snadno implementovat i do počítačů s malými procesory. Výhodu to má zejména při implementacích do zařízení, které již mají implementován TCP/IP protokol, protože integrace je poměrně jednoduchá a nevyžaduje modifikaci hardwaru. [14][15]



Obr. 2 Hierarchie vrstev protokolu MQTT [15]

Obsah zpráv

Každá zpráva se skládá z tématu (topic) a obsahu (payload). Obsah zprávy, která se má přenést jsou pouze binární data, která nejsou nijak definovaná. Nejčastěji používanými formáty dat jsou JSON (JavaScript Object Notation), BSON (Binary JSON) a řetězce (text). Vzhledem k tomu, že MQTT broker data nezpracovává, ale jen je příslušně přesměruje, mohou být data libovolného formátu. Komu naopak záleží na formátu dat, je příjemce/odběratel (SUBSCRIBER), který musí vědět, jak data chápat, aby je mohl zpracovat. MQTT přidává jen minimum servisních dat, aby byla velikost zprávy co nejmenší. Tímto se velmi hodí na přenášení občasných a na rychlost přenosu méně náročných informací, výhodné to je právě při využití v IoT. [14][15]

Výměna zpráv

Na začátku navazuje klient spojení s brokerem a po navázání spojení pošle zařízení zprávu „CONNECT“. Přihlášení může probíhat i včetně ověření pomocí jména a hesla. Po dokončení připojování broker odešle zprávu „CONNACK“ a tím potvrdí připojení. Po tomto procesu následuje nastavení odebíraných témat, které si volí odběratel (SUBSCRIBER). Lze odebírat jedno téma s jednou proměnnou, několik proměnných zvlášť, nebo celý blok témat (proměnných). Tento odběr broker potvrzuje zprávou „SUBACK“. V průběhu fungování se zařízení mohou přihlašovat a odhlašovat z vybraných témat kdykoli dle potřeby. Po navázání

spojení je možné ihned zprávy posílat i jako takzvaný odesílatel (PUBLISHER), na libovolné již aktivní téma nebo pomocí publikování vytvořit téma nové. MQTT dokáže detekovat, že zařízení je stále aktivní, protože pokud zařízení v určitém čase nereaguje, považuje ho za odpojené a pošle takzvanou závět. Aby se zařízení nedostalo do tohoto stavu, v případě, že nemá žádnou zprávu, posílá „PINGREQ“, na kterou broker odpoví „PINGACK“. V případě ztráty spojení může broker poslat výše uvedenou „závěť“, tato zpráva není povinná. [14][15]

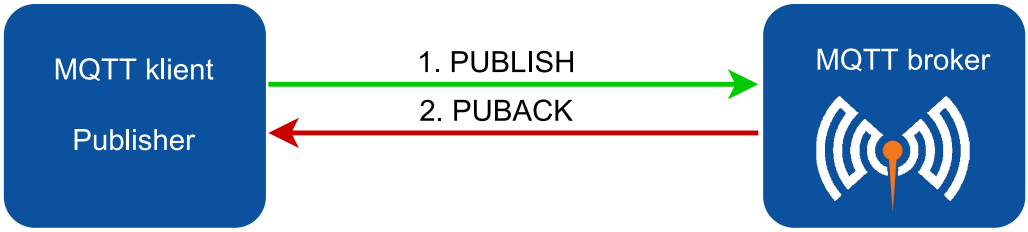
Úroveň kvality služeb (Quality of Services – QoS)

V souvislosti se samotným přenosem zpráv protokol MQTT definuje tři úrovně kvality služeb QoS. Úrovně se liší podle toho, jak je zajištěno dodání zprávy, označení je číselné 0, 1, 2.

Nejnižší úroveň kvality služeb je **„QoS 0“ -** zpráva je odeslána a není potvrzeno její přijetí. Výstižný popis je anglicky „at most once“ (maximálně jednou) nebo „fire and forget“ (vystřel a zapomeň).

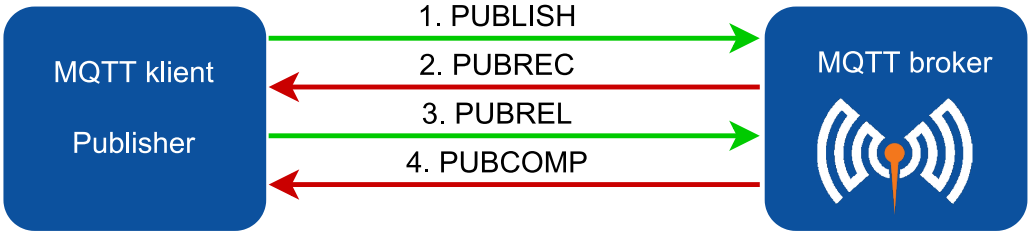
Obr. 3 MQTT úroveň služeb „QoS 0“ [15]

Střední úroveň kvality služeb **„QoS 1“ -** nazývá se „at least once“ (alespoň jednou). Broker zprávu přijme a pošle ji odběratelům. Odeslání proběhne s „QoS 1“ nebo s „QoS 0“, pokud zařízení „QoS 1“ neumí. Obecně platí, že se zpráva odešle s takovým QoS, s jakým je přijata a čeká na potvrzení. Jakmile odběratel potvrdí přijetí zprávy zprávou „PUBACK“, broker zprávu odstraní a pošle „PUBACK“ odesílateli (PUBLISHER) zpět. Odesílatel se tím dozví, že zpráva prošla brokerem, a může ji zahodit. Broker může poslat „PUBACK“, aniž by měl potvrzení od všech odběratelů. Přesné chování závisí na konkrétní implementaci, MQTT umožňuje oba scénáře, tj. čekat i nečekat na všechny odběratele. [14][15]



Obr. 4 MQTT úroveň služeb „QoS 1“ [15]

Nejvyšší úroveň **„QoS 2“ -** zpráva je doručena „exactly once“ (právě jednou). Zprávu posílá odesílatel (PUBLISHER) brokerovi příkazem „PUBLISH“. Ten ji, stejně jako v předchozím případě přijme a posílá ji odběratelům. Odběratel vrátí přes brokera zprávu „PUBREC“ (potvrzení přijetí). Odesílatel (PUBLISHER) odpoví zprávou „PUBREL“, broker zprávu smaže a potvrdí zprávou „PUBCOMP“. Tím je výměna zprávy hotova. [14][15]



Obr. 5 MQTT úroveň služeb „QoS 2“ [15]

Platí, že broker odesílá zprávu na té úrovni QoS, na které přišla, s možností snížit úroveň, pokud klient umí pouze nižší. Klient nemusí podporovat všechny tři uvedené úrovně QoS, protože samotný MQTT protokol to nepožaduje.

Kromě QoS se u zprávy nastavuje příznak (retain flag), který říká, že broker nemá zprávu zahazovat po rozeslání, ale uložit a poslat novým odběratelům daného tématu. Posílá se vždy poslední uložená zpráva s příznakem „retain“.

Důležité je, že odesílatel (PUBLISHER) může zprávu poslat s jakýmkoli QoS. Odběratel (SUBSCRIBER) si v rámci odběru tématu může říct, jaký maximální QoS chce dostávat. Broker pak došlé zprávy posílá tak, že nemají vyšší QoS než žádaný. Pokud tedy přijde zpráva s „QoS 2“, ale odběratel chce maximálně „QoS 1“, broker pošle tutéž zprávu s „QoS 1“. To samozřejmě znamená, že tento odběratel může tuto zprávu dostat vícekrát, protože „QoS 1“ nezaručuje doručení „právě jednou“. [14][15]

Struktura zpráv

Zprávy jsou rozděleny na 3 části, které jsou ve zprávě obsaženy vždy nebo jen v určitých zprávách:

- fixní hlavička (ve všech zprávách);
- variabilní hlavička (pouze v určitých zprávách);
- data (pouze v určitých zprávách). [14][15]

Fixní hlavička

Tabulka 1 Fixní hlavička zprávy MQTT [15]

Bit	7	6	5	4	3	2	1	0
Bajt 1	Typ zprávy (message type)				Příznaky specifické pro každý paket			
Bajt 2	Zbývající délka							

- **Typ zprávy –** například CONNECT, SUBSCRIBE, PUBLISH atd.
- **Příznaky specifické pro každý paket –** tyto 4 bity se používají pro pomocné příznaky, jejich přítomnost a stav je závislý na typu zprávy.
- **Zbývající délka –** představuje délku zprávy (variabilní hlavička + data), velikost je 1-4 bajty.

Tabulka 2 Specifické příznaky paketů MQTT [15]

Bit	7	6	5	4	3	2	1	0
Bajt 1					DUP	QoS	QoS	Retain
Bajt 2								

- **DUP** – duplikát, je nastaven, když klient nebo MQTT broker provede opětovné odeslání paketu (používá se v PUBLISH, SUBSCRIBE, UNSUBSCRIBE, PUBREL).
- **QoS** – kvalita služeb (0, 1, 2).
- **Retain** – při publikování dat s nastaveným příznakem „Retain“, broker data uloží. Při dalším odběru tohoto tématu, broker okamžitě odešle zprávu s tímto příznakem. Používá se pouze ve zprávách typu PUBLISH.

Variabilní hlavička

Variabilní hlavička je obsažena jen v některých záhlavích a obsahuje následující údaje:

- identifikátor paketu, který je přítomný ve všech typech zpráv, kromě CONNECT, CONNACK, PUBLISH (QoS < 1), PINGREQ, PINGRESP, DISCONNECT;
- název protokolu, pouze ve zprávě typu CONNECT;
- verze protokolu, pouze ve zprávě typu CONNECT;
- příznaky připojení, určují chování klienta během připojení.

Tabulka 3 Část variabilní hlavičky MQTT [15]

Bit	7	6	5	4	3	2	1	0
Bajt 8	Jméno	Heslo	„Will Retain“	„Will QoS“		„Will Flag“	Čistý start	-

- **Jméno –** pokud je tento příznak obsažen v datech, musí být uvedeno uživatelské jméno (používá se pro autentifikaci klienta).
- **Heslo –** pokud je tento příznak obsažen v datech, musí být zadáno heslo (používá se pro autentifikaci klienta).
- **Will Retain –** pokud je příznak nastaven na 1, broker udržuje Will Message.
- **Will QoS –** kvalita služeb pro Will Message, pokud je nastaven příznak Will Flag, jsou Will QoS a Will Retain povinné.
- **Will Flag –** pokud je nastaven příznak poté, co se klient odpojí od brokera bez odeslání příkazu DISCONNECT (například při výpadku komunikace atd.), broker to oznámí všem klientům prostřednictvím Will Message.
- **Čistý start (clean session) –** pokud je příznak nastaven na „0“, broker uloží všechny odběry klienta, a při příštím připojení mu předá všechny zprávy s „QoS 1“ a „QoS 2“, které byly přijaty brokerem během odpojení klienta. Pokud je příznak nastaven na „1“, při opětovném připojení se klient bude muset znovu přihlásit k odběru témat.

Nastavení relace (Session Present – SP),

Nastavení relace se používá ve zprávě typu „CONNACK“.

- Pokud broker přijímá spojení s Clean Session = 1, musí SP nastavit na „0“.
- Pokud broker přijímá spojení s Clean Session = 0, pak hodnota bitu SP závisí na tom, zda broker dříve uložil relaci s tímto klientem (pokud ano, pak je SP nastaven na „1“ a naopak). To znamená, že tento parametr umožňuje klientovi určit, zda byla předchozí relace uložena brokerem. [14][15]

Návratový připojovací kód (Connect Return code),

Tento kód je použit, pokud broker z nějakého důvodu od klienta nemůže přijmout správně vytvořený „CONNECT“ paket. Pak ve druhém bajtu „CONNACK“ paketu musí nastavit odpovídající hodnotu z tabulky č. 4.

Tabulka 4 Návratový připojovací kód MQTT [15]

Hodnota	Návratový kód
0	0x00 – připojení přijato
1	0x01 – připojení odmítnuto, nepřijatelná verze protokolu
2	0x02 – připojení odmítnuto, klient ID není v seznamu povolených
3	0x03 – připojení odmítnuto, připojení navázáno, ale MQTT není k dispozici
4	0x04 – připojení odmítnuto, nesprávné heslo nebo jméno
5	0x05 – připojení odmítnuto, přístup k připojení je zakázán
6–255	rezervováno

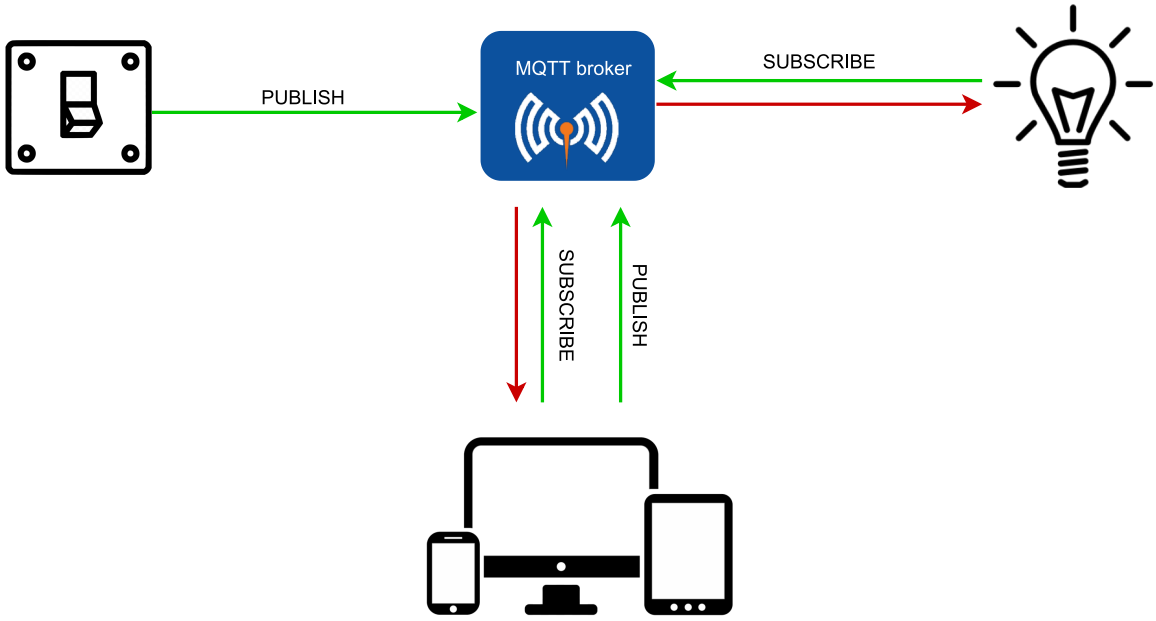
Data

Obsah a formát dat přenášených ve zprávách MQTT je definován v každé aplikaci individuálně. Velikost dat lze vypočítat odečtením od zbývající délky z hlavní hlavičky. Celkem má protokol MQTT 15 typů zpráv, které jsou popsané v tabulce č. 5.

Tabulka 5 Typy zpráv protokolu MQTT (C – klient, S – server) [15]

Typ zprávy	Hodnota	Směr přenosu	Popis
Rezervováno	0000 (0)	-	Rezervováno
CONNECT	0001 (1)	C -> S	Žádost klienta o připojení k serveru
CONNACK	0010 (2)	C	Potvrzení připojení
PUBLISH	0011 (3)	CS	Publikace zprávy
PUBACK	0100 (4)	CS	Potvrzení publikace
PUBREC	0101 (5)	CS	Publikace přijata
PUBREL	0110 (6)	CS	Povolení odstranění zprávy
PUBCOMP	0111 (7)	CS	Dokončení publikace
SUBSCRIBE	1000 (8)	C -> S	Žádost o odběr
SUBACK	1001 (9)	C	Potvrzení k odběru
UNSUBSCRIBE	1010 (10)	C -> S	Žádost o odhlášení
UNSUBACK	1011 (11)	C	Potvrzení odhlášení
PINGREQ	1100 (12)	C -> S	Žádost o PING
PINGRESP	1101 (13)	C	Odezva PING
DISCONNECT	1110 (14)	C -> S	Odpojení od serveru
Rezervováno	1111 (15)	-	Rezervováno

Příklad využití MQTT v praxi



Obr. 6 Příklad využití MQTT v praxi [14]

MQTT je velmi snadno použitelná technologie u které existuje spoustu implementací brokerů. Jedna z nejznámějších je MQTT broker Mosquitto, který je open source. S MQTT se lze setkat i u cloudových služeb. Hodně využívaný je také v systémech domácí automatizace. Lze konstatovat, že MQTT by šlo považovat za jeden ze standardů používaných v rámci IoT. [14][15]

MQTT broker

MQTT broker je komunikační komponenta (server), která dovoluje snadné a efektivní propojení různých senzorů s dalšími prvky komunikační infrastruktury pomocí protokolu MQTT, k brokerovi se mohou připojovat klienti.

MQTT klient

MQTT klient je jakékoli zařízení připojující se k MQTT brokeru. Může být jak odesílatel (PUBLISHER), tak odběratel (SUBSCRIBER) nebo obojí.

Sémantika témat

Témata (topic) jsou znaky s kódováním UTF-8 a hierarchie témat má stromovou strukturu, to usnadňuje jejich organizaci a přístup k datům. Témata se skládají z jedné nebo více úrovní, které jsou oddělena lomítkem (/). Příklad takového tématu, „Control/Primar/AxisX/ActualPosition“.

Jak již bylo zmíněno výše, odběratel (SUBSCRIBER) může přijímat data z několika témat současně. Pro tyto účely existují zástupné znaky. Existují dva typy zástupných znaků.

- **Jednoúrovňový zástupný znak –** k označení se používá znak (+), pokud například potřebujeme získat data aktuální pozice všech os, můžeme odebírat toto téma neboli „topic“: „Control/Primar/+/ActualPosition“.
- **Víceúrovňový zástupný znak –** k označení se používá znak (#), pokud například potřebujeme získat všechny proměnné osy X, můžeme odebírat toto téma: „Control/Primar/AxisX/#“. [15]

3.1.6 OpenHAB (Open Home Automation Bus)

OpenHAB je open-source software napsaný v programovacím jazyce Java, který slouží především k řízení inteligentních domů. Sám o sobě neposkytuje přímé služby pro ovládání jednotlivých prvků (IoT), ale poskytuje jednotné prostředí pro sjednocení jejich ovládání pomocí webového rozhraní. Díky této modularitě lze software použít i na řízení víceosého pozicionéru. OpenHAB funguje na libovolném zařízení, které je schopné spustit Javu (Windows, Linux, Mac). Ke klasickému webovému rozhraní umožňuje navíc zobrazení na mobilních platformách iOS a Android. Lze jej škálovat s příchodem nových systémů a služeb.

Na Raspberry Pi lze využít systému OpenHABian, který je samočinně se konfigurující nastavení systému Linux, které splňuje potřeby pro použití systému openHAB. [17][18][19][20]

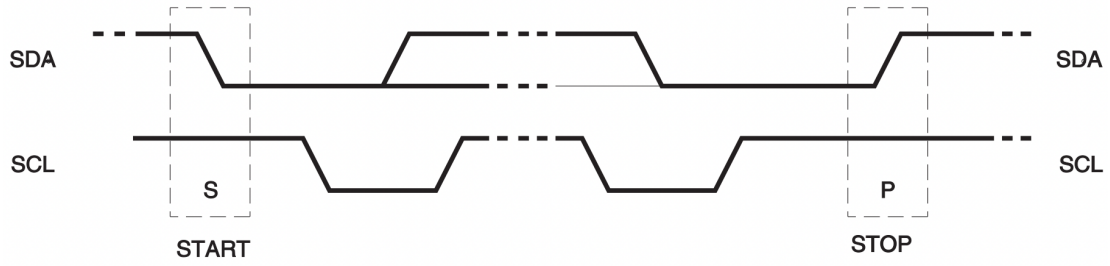
3.1.7 Sběrnice I^2C

Sběrnice I^2C je takzvaně multi-masterová sběrnice vyvinutá firmou Philips, která slouží k propojování nízkorychlostních periferií. Anglický název je (Inter-Integrated Circuit), správně se tento název čte „I-squared-C“ a často se nesprávně používá „I-two-C“. Podobné rozhraní existuje s anglickým názvem „Two Wire Interface“ (TWI), které je prakticky stejné jako I^2C, ale protože je název I^2C chráněn, byla vyvinuta tato alternativa, kterou využívá například firma Atmel, ale i další výrobci. I^2C se stal celosvětovým standardem, který implementuje tisíce různých integrovaných obvodů a je využíván v dalších systémech, jako je například sběrnice pro správu systému (SMBus), sběrnice pro správu napájení (PMBus) nebo kanál pro display (Display Data Chanel – DDC). Pro úspěšné fungování sběrnice I^2C nebo TWI je třeba dvou vodičů. Jedním vodičem jsou přenášena data (SDA) a jedním hodinový signál (SCL). Sběrnice je sériová a nepodporuje duplexní přenos. Na sběrnici jsou

dva typy zařízení, jedno z nich je „master“ (šéf), který řídí přenos, zahajuje a ukončuje komunikaci a také generuje hodinový signál na lince SCL. Druhý typ zařízení je „slave“, v překladu něco jako podřízený. Aby vše správně fungovalo, připojená zařízení na sběrnici musí mít unikátní adresu. Tato adresa je pevně nastavená od výrobce, softwarově nastavitelná, případně nastavitelná pomocí jumperu nebo nějakého přepínače (v celém nebo omezeném rozsahu). Protože je I^2C plnohodnotná sběrnice, kam lze připojit více řadičů, obsahuje také detekci kolizí a proces arbitráže, aby se zabránilo poškození dat, když začne více zařízení vysílat současně. [21][22][23]

Fyzická vrstva

Při sedmibitovém adresování sběrnice umožňuje propojení až 128 různých zařízení, propojených pouze dvěma vodiči. Z elektrického hlediska jsou oba kanály zapojeny jako otevřený kolektor. Nejvyšší přípustná kapacita vedení je 400 pF a tím je omezena maximální délka vodičů. Každý vodič musí být připojen jedním pull-up (typicky cca 10k) rezistorem ke kladnému napětí (nejčastěji 3 až 5 V), což zajistí vysokou úroveň v klidovém stavu. Při probíhajícím přenosu jsou na SDA vysílány jednotlivé datové bity, kdy platí pravidlo, že logická úroveň na SDA se smí měnit pouze je-li SCL v nízké úrovni (hranice mezi nízkou a vysokou úrovní může být například 2,1 V, jako je to u použitých driverů Pololu Tic T500). Toto pravidlo je porušeno ve dvou speciálních stavech. Tyto stavy jsou při vyslání bitu START a bitu STOP a používají se pro zahájení a ukončení přenosu. [21][22][23]

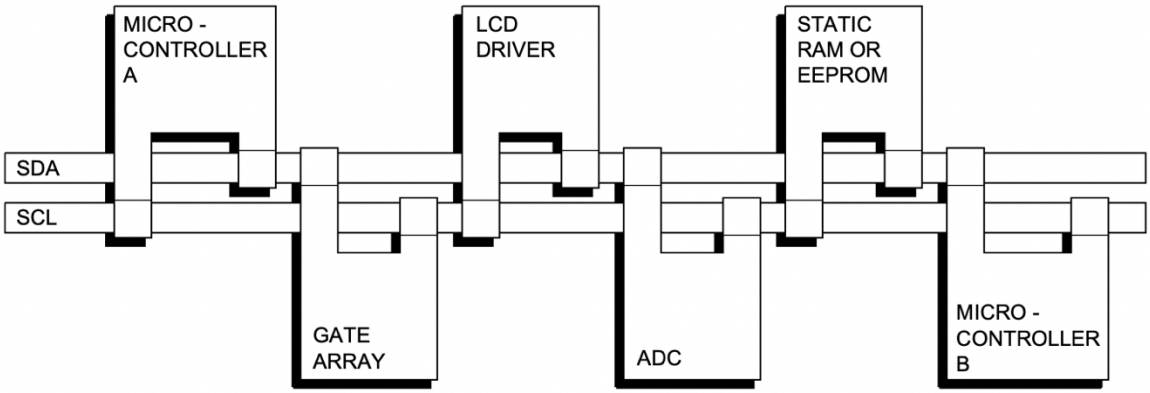


Obr. 7 Podmínky START a STOP na sběrnici I^2C [21]

Maximální frekvence hodinového signálu SCL je podle verze I^2C 100 kHz nebo 400 kHz a je dána minimální povolená doba setrvání v nízké a vysoké úrovni na lince SCL. Při komunikaci i při přenosu dat si jednotlivá zařízení synchronizují generátory hodin tak, že trvání vysoké úrovně na SCL je odměřováno vnitřním časovačem každého zařízení až od okamžiku, kdy SCL skutečně vysoké úrovně dosáhne (vzhledem k tomu, že SCL je typu otevřený kolektor, může být v nízké úrovni držen i v situaci, kdy se nějaké zařízení pokouší nastavit vysokou úroveň). Podobně je doba trvání nízké úrovně na SCL odměřována od sestupné hrany. Tento mechanismus umožňuje některému zařízení zpomalit přenos dle

potřeby (pomalé zařízení může držet určitou dobu signál SCL v nízké úrovni a tím zabránit vysílajícímu zařízení ve vyslání dalšího bitu). Sběrnice I^2C neumožňuje duplexní přenos, v jednom okamžiku vysílá jen jedno zařízení. Všechna zařízení připojená na sběrnici musí mít individuální adresu o délce 7 nebo 10 bitů a implementovaný mechanismus komunikace pomocí I²C sběrnice.

Ke sběrnici I^2C lze připojit více než jedno zařízení schopné řídit sběrnici. Protože zařízení jsou obvykle mikrokontrolery, lze přenášet data i mezi dvěma mikrokontrolery připojenými ke sběrnici. [21][22][23]



Obr. 8 Příklad připojení více zařízení typu "master" na sběrnici I^2C [21]

Popis situace dvou mikrokontrolerů z obrázku č. 8. V případě situace, že mikrokontroler **A** chce odeslat informace do mikrokontroleru **B**, tak **A** bude jako vysílač a odesílá data do **B** jako do přijímače. **A** následně ukončuje přenos. V tomto případě **A**, jakožto „master“, vygeneruje hodinový signál. Možnost připojení více než jednoho mikrokontroleru ke sběrnici I^2C znamená, že se více než jedno zařízení může pokusit zahájit přenos dat současně. Pro předcházení chaosu, který by z takové události mohl vyplynout, je vyvinut arbitrážní postup. [21][22][23]

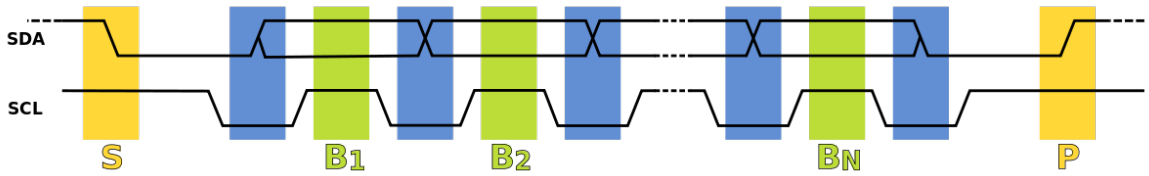
Arbitráž

Arbitráž je část protokolu, která je vyžadována pouze v případě použití více řadičů na jedné sběrnici. Cílové zařízení nejsou zapojeny do arbitrážního řízení. Řadič může zahájit přenos pouze v případě volné sběrnice. Dva řadiče mohou generovat bit START s minimální dobou prodlevy. K určení, který řadič „master“ dokončí přenos, je vyžadována arbitráž. Během každého bitu, když je SCL ve vysoké úrovni, všechny řadiče kontrolují, zda úroveň SDA odpovídá tomu, co odeslaly. Tento proces může trvat i několik bitů. Může se stát, že dva řadiče dokončí celý přenos bez chyby v případě, že je přenos identický. Když se zařízení poprvé pokusí odeslat vysokou úroveň, ale zjistí, že úroveň SDA je v nízké úrovni, zařízení

ví, že ztratilo arbitráž a vypne svůj výstupní ovladač SDA. Druhé zařízení pokračuje v dokončení přenosu.

Během arbitrážního řízení se neztratí žádné informace. Zařízení, které prohraje arbitráž, může generovat hodinové impulsy až do konce bajtu, ve kterém arbitráž ztratí. Následně musí restartovat svůj přenos, když je sběrnice volná. [21][22]

Časový diagram

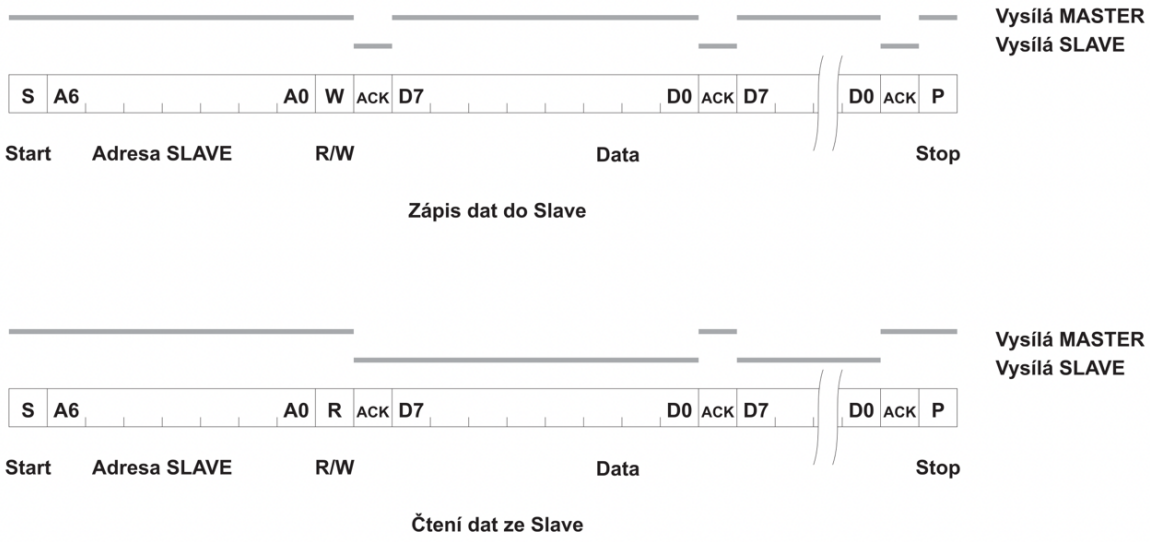


Obr. 9 Časový diagram I^2C přenosu [23]

Jak je vidět na obrázku č. 9, přenos dat se zahajuje bitem START (S), zatímco je SDA v nízké úrovni a SCL zůstává v úrovni vysoké. Po přepnutí SCL na nízkou úroveň, se na SDA nastaví přenášený bit a SCL je stále v nízké úrovni (modrá). Poté při vzestupné hraně na SCL (zelená) jsou odebrány přijatá data. Když je přenos dokončen, je poslaný bit STOP (P) pro uvolnění datové linky. To se provede změnou SDA a SCL na vysokou úroveň. Při vysoké úrovni SDA a SCL se zabraňuje falešně detekci a zároveň je to klidový stav sběrnice. [21][22][23]

Linková vrstva

Jak již bylo výše popsáno, každému přenosu dat předchází vyslání bitu START. Potom se vyšle 7bitová adresa příjemce a jeden bit pro čtení READ (R) nebo zápis WRITE (W), který indikuje požadovanou operaci. Další bit ACK je vyslán s vysokou úrovní a slouží k potvrzení od příjemce, že signál byl přijat. Následně mohou být přenášená data ve směru určeném předchozím bitem R/W. Bit ACK následuje po každém bajtu. Po ukončení přenosu je vyslán bit STOP. [21][22][23]



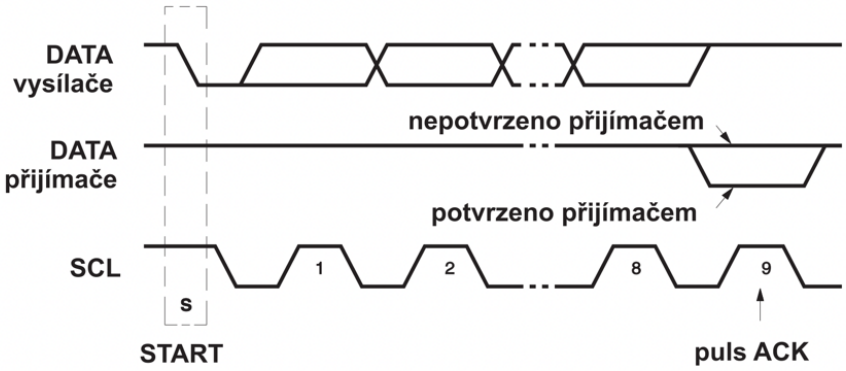
Obr. 10 Formát rámců na sběrnici I^2C [22]

Adresování

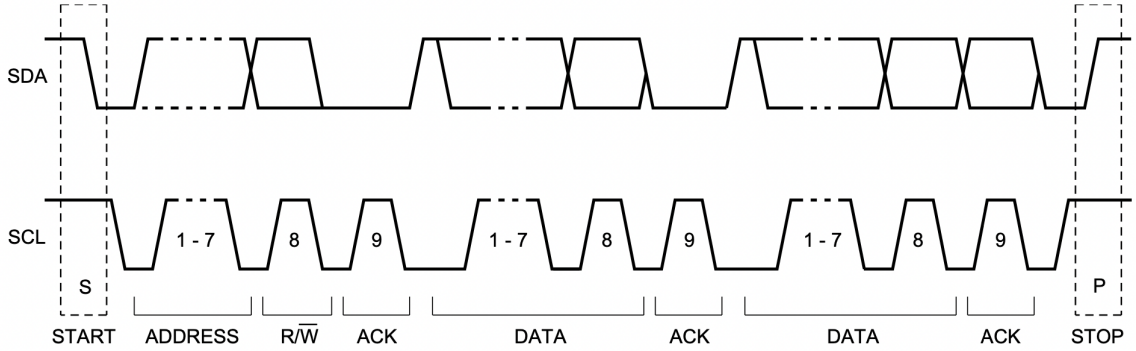
Každé zařízení připojené na I^2C má přidělenou 7bitovou adresu. Po zachycení bitu START porovnávají všechny obvody svou adresu s přijatou adresou, která je vysílána na sběrnici. Pokud zařízení vyhodnotí shodu, je vysílání určeno jemu a musí přijetí adresy potvrdit bitem ACK. Potom přijímá nebo vysílá další data. Několik adres je na I^2C vyhrazeno pro speciální účely. Například adresa 0000000 je určena pro vysílání broadcast atd. V praxi je časté, že několik adresních bitů je určeno už při výrobě. Zbývající bity se volí pomocí vývodů k tomu určených, které lze dle potřeby připojit na vysokou nebo nízkou úroveň a tím nastavit příslušné adresní bity. Adresa 11110aa indikuje 10bitové adresování, kde aa označuje dva nejvyšší bity adresy, zbývajících 8 bitů je vysíláno v následujícím bajtu. [21][22][23]

Potvrzování

Každý vyslaný bajt je potvrzen jedním bitem ACK. Vysílač jej vyšle vysokou úrovní. Přijímač potvrzuje přijetí tím, že v době vysílání ACK připojí na SDA nízkou úroveň. Když vysílač nedostane potvrzení, ukončí vysílání bitem STOP. [21][22][23]



Obr. 11 Potvrzování zpráv na sběrnici I²C bitem ACK [21]



Obr. 12 Kompletní přenos na sběrnici I²C [21]

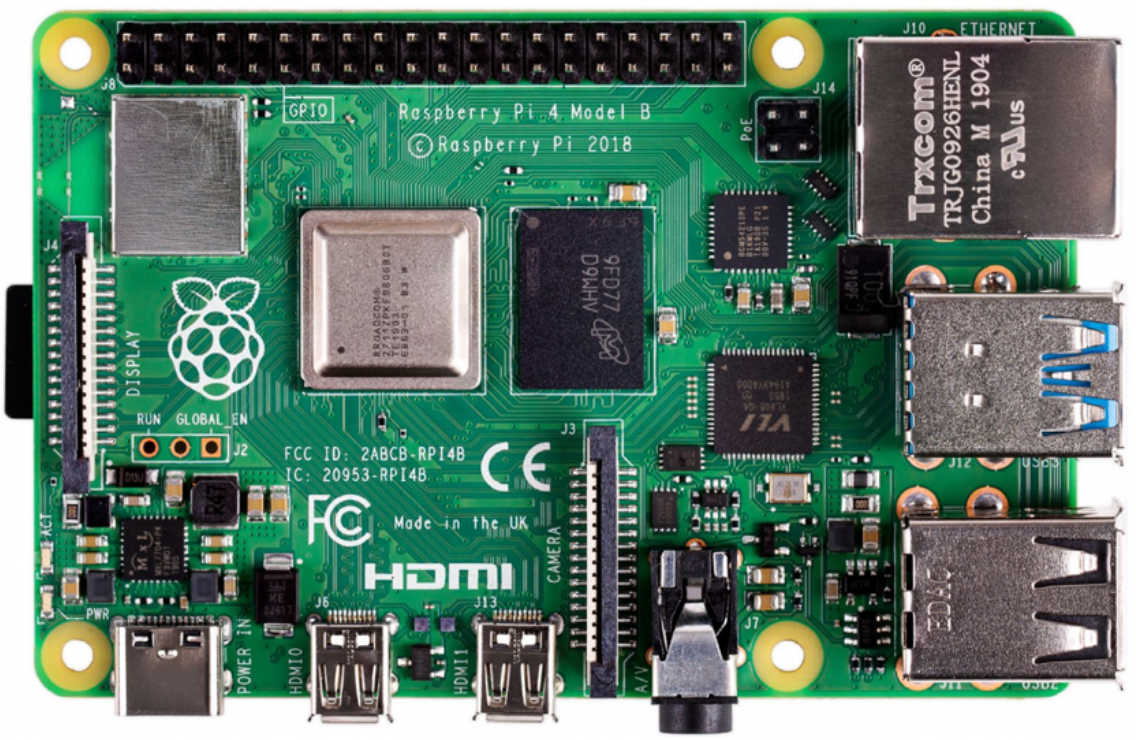
Výhody sběrnice I²C

- Extrémně nízká spotřeba;
- vysoká odolnost proti rušení;
- široký rozsah napájecího napětí;
- široký rozsah provozních teplot. [21][22][23]

3.2 Hardwarové prostředky

3.2.1 Raspberry Pi 4 model B

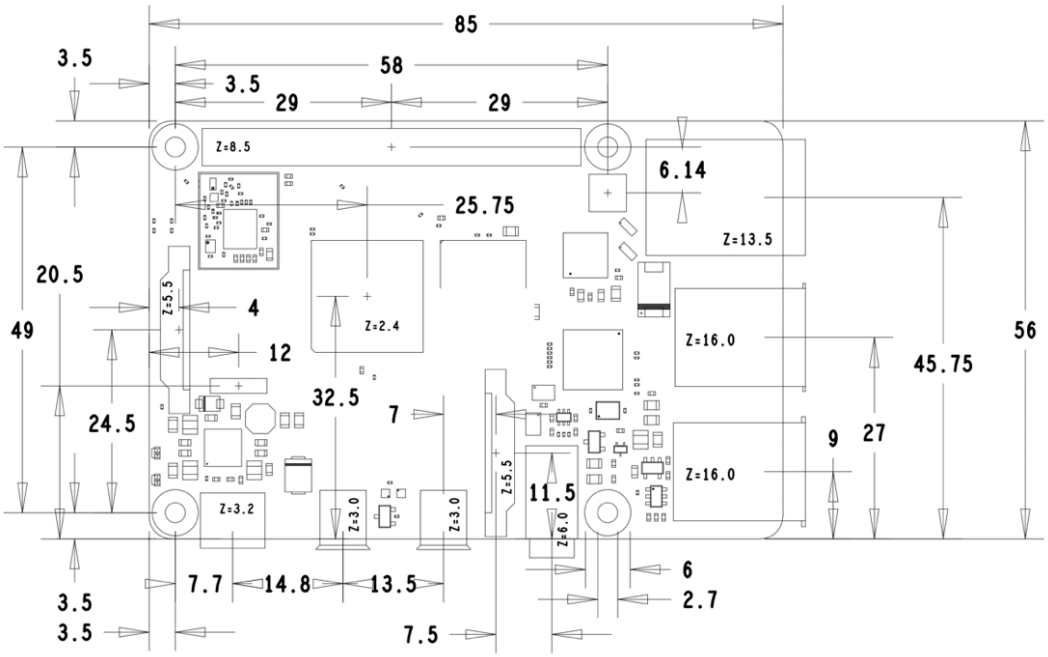
Raspberry Pi je malý jednodeskový počítač velký přibližně jako platební karta. V roce 2012 byl vyvinut britskou nadací Raspberry Pi Foundation s cílem podpořit výuku informatiky ve školách. Primárním operačním systémem je Raspbian postavený na systému Linux (Debian). Označení „Raspberry Pi“ má ochrannou známku. [24]



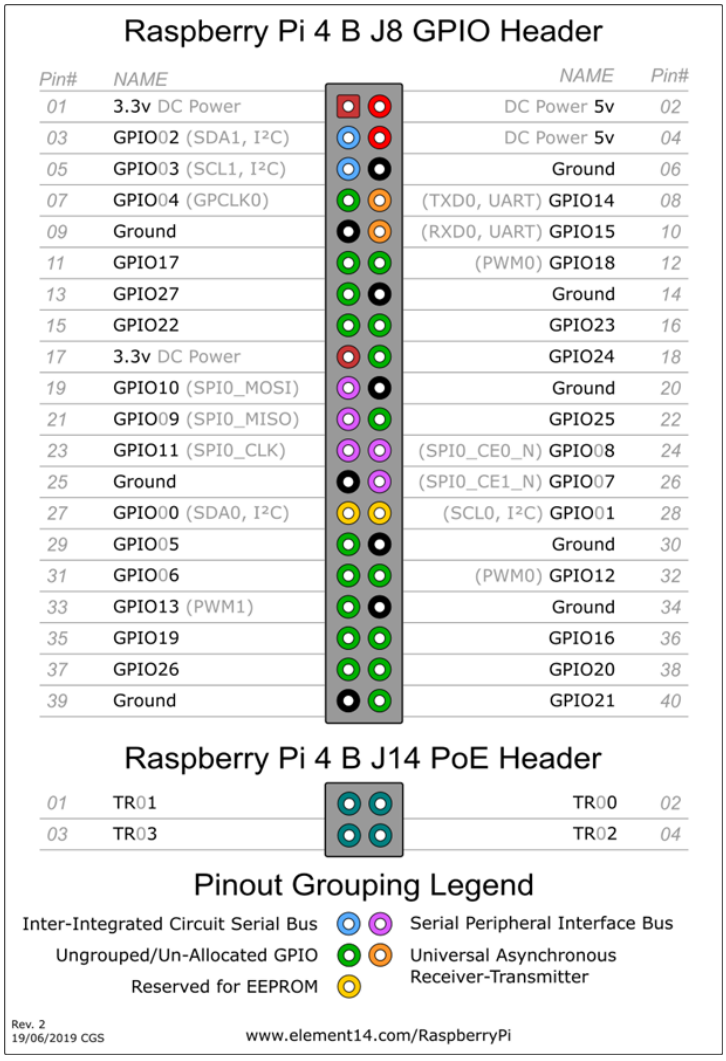
Obr. 13 Použité Raspberry Pi 4 model B [24]

Tabulka 6 Některé specifikace použitého Raspberry Pi [24]

Procesor:	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Operační paměť RAM:	8 GB
Rozhraní:	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE, Gigabit Ethernet, 2 × USB 3.0 ports, 2 × USB 2.0 ports
GPIO:	standardní 40ti pinový GPIO konektor
Video a zvuk:	2 × micro HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port, 2-lane MIPI CSI camera port, 4-pole stereo audio and composite video port

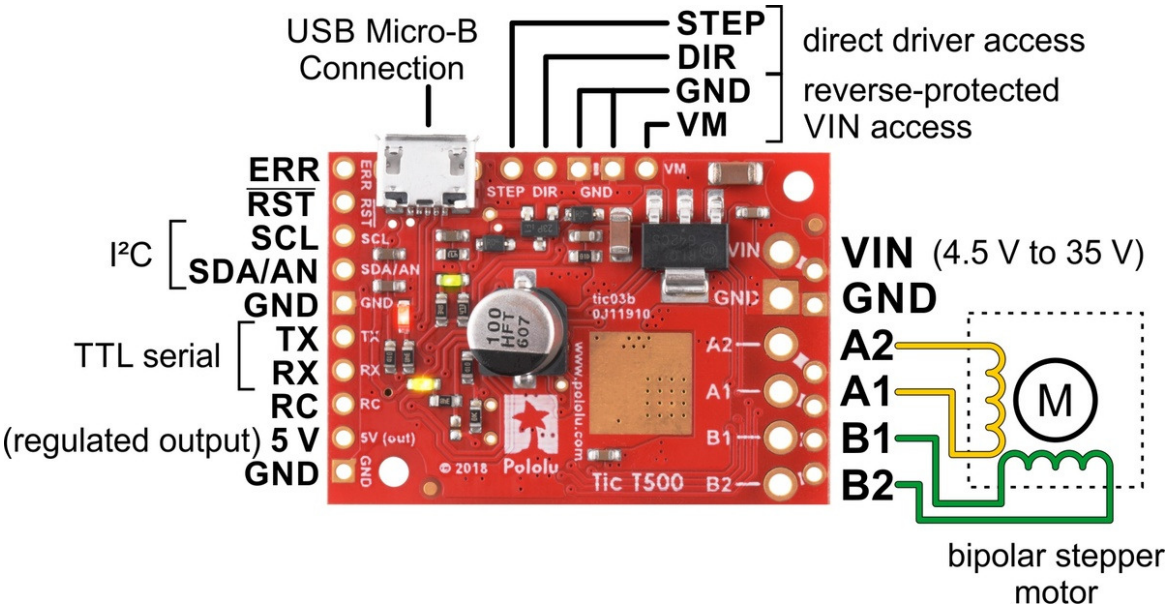


Obr. 14 Výkres použitého Raspberry Pi, kótovaný v milimetrech [24]



Obr. 15 Pin-out použitého Raspberry Pi [25]

3.2.2 Pololu Tic T500



Obr. 16 Driver Pololu Tic T500 [26]

Popis

Pololu Tic T500 je driver pro krokové motory, který disponuje rychlou konfigurací přes USB rozhraní pomocí bezplatného softwaru společnosti Pololu. Driver podporuje šest ovládacích rozhraní:

- USB;
- sériové TTL;
- I²C;
- analogové napětí (potenciometr);
- kvadraturní enkodér;
- rádiové ovládání (RC).

Tento driver využívá integrovaného obvodu MP6500, který může pracovat od 4,5 V do 35 V a může dodávat přibližně 1,5 A na fázi bez chladiče nebo nuceného proudění vzduchu. Případně až 2,5 A max., s dostatečným přídavným chlazením.

Pololu Tic T500 má několik režimů ovládání, v této práci budou použity tři: režim nastavení pomocí PC a USB (pro samotné řízení pomocí Raspberry Pi), režim komunikace pomocí I^2C a režim STEP/DIR pro synchronní pojezdy. [26]

Režim USB

Režim USB má nejširší možnosti nastavení (příkazů) a lze pomocí tohoto režimu driver konfigurovat. Připojení pomocí tohoto rozhraní má však nevýhodu, která je tím významnější, čím více zařízení má být řízeno současně. Je pak zapotřebí použití USB

rozbočovačů, což může nakonec vést k problémům s komunikací nebo časem k problémům s kompatibilitou ovladačů a toto rozhraní tedy není dlouhodobě příliš spolehlivé. [26]

Režim I²C

Protože Tic T500 považuje vstupní hodnotu 2,1 V na SCL nebo SDA za vysokou, lze tyto piny připojit přímo k 3,3 V mikrokontrolerům, aniž by byl třeba převodník napěťové úrovně. Po sběrnici I²C lze nastavit většinu parametrů jako u rozhraní USB. Některé nastavení je však nedostupné, jako například uložení nastavení do energeticky nezávislé paměti pro „stálé“ uložení nastavení i po vypnutí. Dále není k dispozici příkaz pro tovární nastavení driveru a start bootloaderu. Přes I²C lze nastavovat například tyto parametry:

- cílovou polohu;
- cílovou rychlost;
- zastavení motoru;
- interní funkci pro „homing“ (speciální metoda využitá při inicializaci);
- nabuzení a odbuzení cívek motoru;
- vymazání chyb;
- nastavení maximální rychlosti, zrychlení, zpomalení;
- reset driveru;
- nastavení proudového limitu;
- výběr krokového režimu;
- načítání proměnných s informacemi o stavu driveru atd. [26]

Režim STEP/DIR

V tomto režimu je nutné připojit mikrokontroler k pinům STEP a DIR na driveru. Piny STEP a DIR jsou připojeny přes ochranné rezistory 220 Ω nebo 470 Ω ke vstupním pinům STEP a DIR, integrovaného obvodu MP6500. Použitím tohoto režimu se obejde funkce pro omezení rychlosti a zrychlení. Driver tak nebude mít žádné informace o aktuální poloze nebo rychlosti motoru.

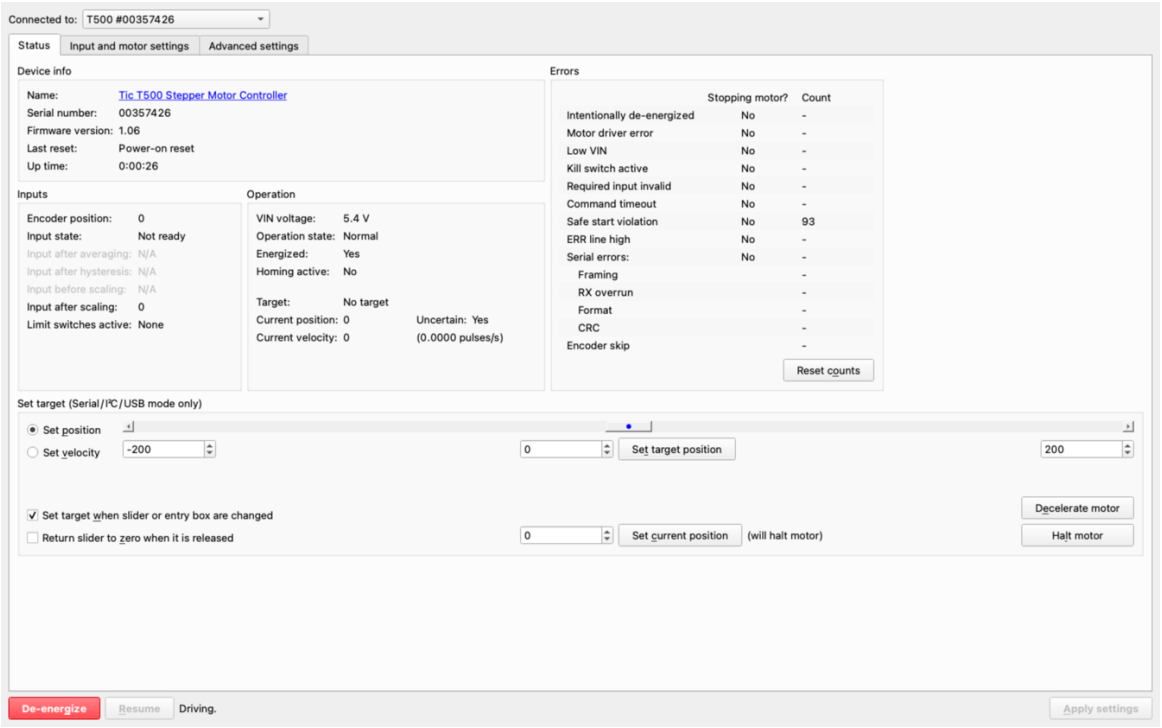
Piny STEP a DIR jsou ve výchozím nastavení v logické nule (nízká úroveň). Driver udělá jeden krok motoru, kdykoli přijde náběžná hrana na pin STEP ve směru kroku určeným pinem DIR. Podrobné specifikace rozhraní STEP/DIR jsou uvedeny v datovém listu MP6500.

V režimu STEP/DIR lze rozhraní USB, sériové TTL nebo I²C stále používat k nastavení proudového limitu, režimu kroku nebo například k deaktivaci.

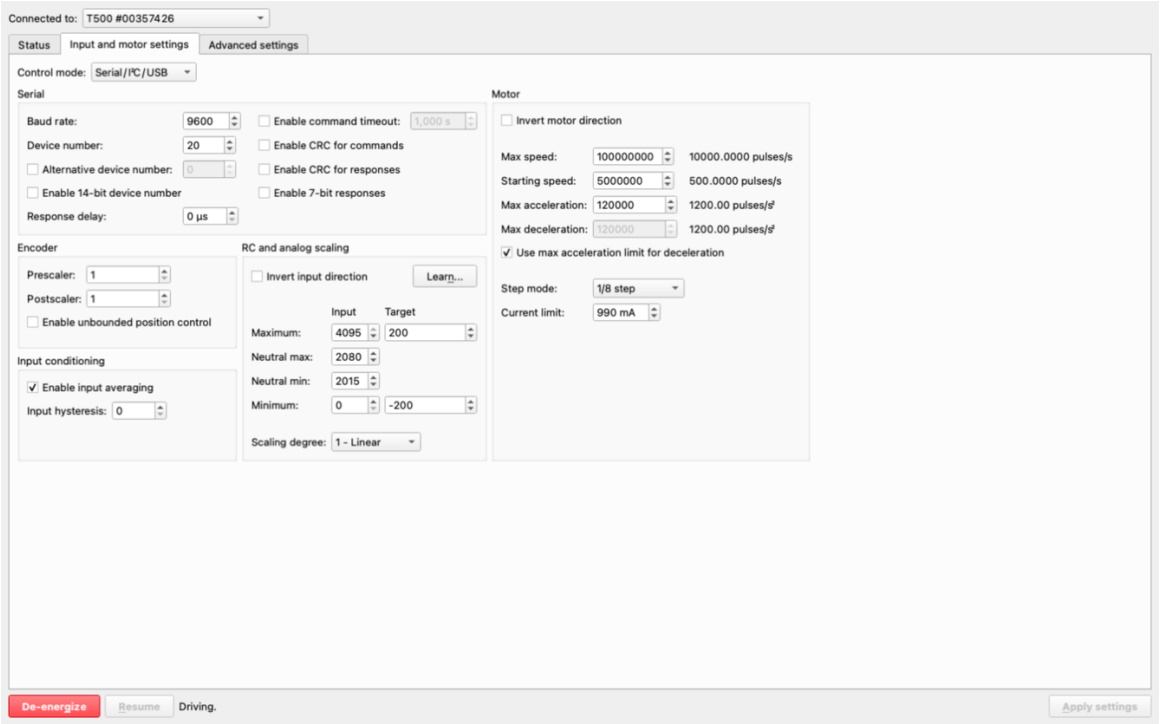
Když je režim ovládání nastaven na cokoliv jiného než na STEP/DIR, lze použít piny STEP a DIR jako výstupy pro ovládání dalšího driveru krokového motoru a zajistit tím tak synchronní pohyb více motorů. Této funkčnosti bude využito i v samotné práci, předejde se tak problémům různého časování příkazů, motory tak mohou být mechanicky spojeny pro synchronní pohyb. [26][27]

Koncové spínače

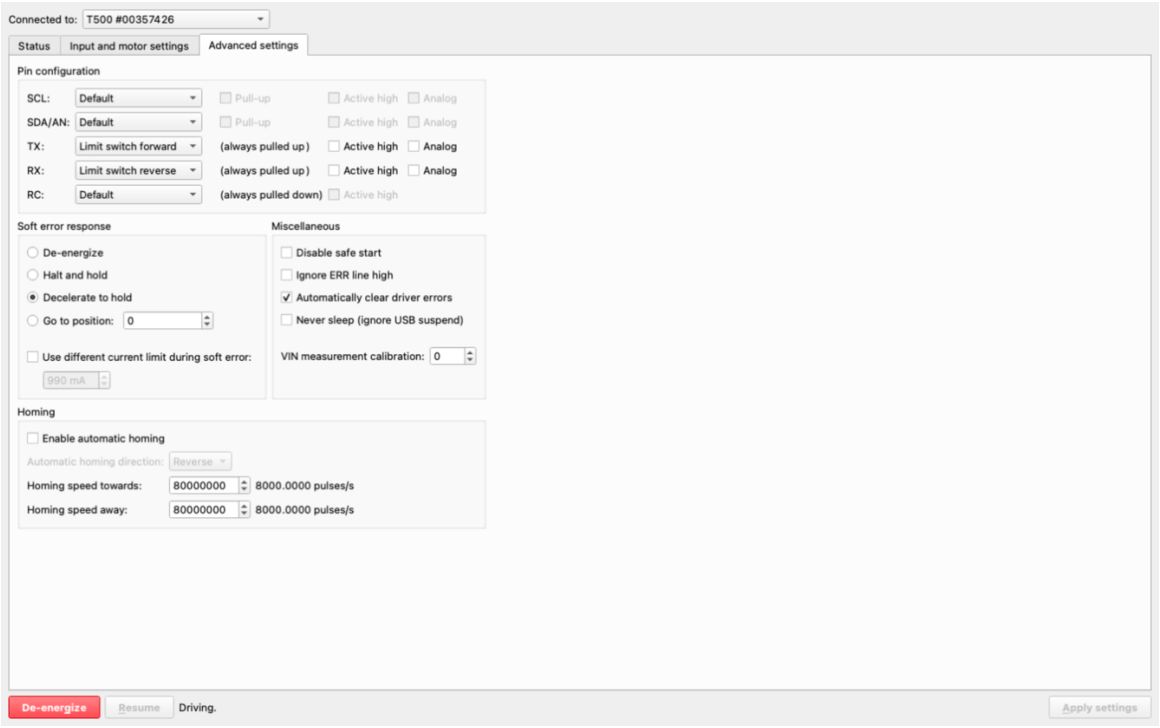
Jakýkoli z pinů driveru (SCL, SDA, TX, RX nebo RC) lze nakonfigurovat jako digitální vstup pro koncový spínač. Jednotlivé piny lze konfigurovat v originálním ovládacím softwaru od výrobce Pololu na kartě Pokročilá nastavení. [26][27]



Obr. 17 Nastavení T500 na kartě STATUS [26]



Obr. 18 Nastavení T500 na kartě Vstup a nastavení motorů [26]

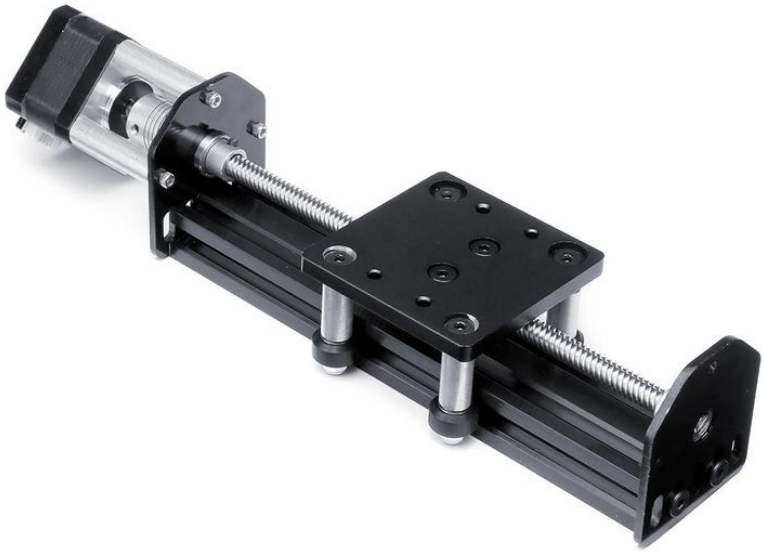


Obr. 19 Nastavení T500 na kartě Pokročilá nastavení [26]

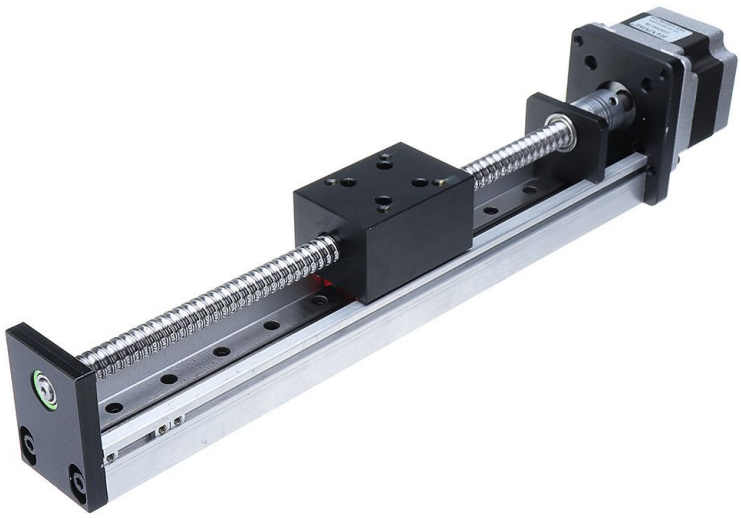
3.2.3 Motory s pojezdy

Tabulka 7 Důležité parametry použitých motorů [28][29]

Parametr	HANSPOSE 17HS3401S	HANSPOSE 23HS4128
Počet fází:	2	2
Úhel kroku:	1,8°	1,8°
Přesnost kroku:	±5 %, v režimu full-step	±5 %, v režimu full-step
Počet vývodů:	4	4
Jmenovitý proud:	1,3 A	2,8 A
Maximální napětí:	500 VAC po 1 minutu	500 VAC po 1 minutu



Obr. 20 Motor HANSPOSE 17HS3401S s pojezdem [28]



Obr. 21 Motor HANSPOSE 23HS4128 s pojezdem [29]

4 Návrh zařízení

4.1 Popis návrhu

Navrhnuté zařízení využívá výše popsaných softwarových a hardwarových komponent. Celé řízení je postaveno na mikropočítači Raspberry Pi (RPI), na kterém běží systém OpenHABian, MQTT broker a Python skripty.

4.2 Nastavení

Všechny komponenty jsou nastaveny v režimu „localhost“, to znamená, že k ovládání není třeba internetového připojení a pro připojení musí být ovládací zařízení ve formě PC, mobilu nebo tabletu připojeno ve stejné síti. Pro snazší nastavení je na RPI nastavena pevná IP adresa na hodnotu 192.168.0.10, při připojení pomocí ethernetového kabelu. Při připojení na bezdrátovou síť WiFi se IP adresa nastaví dle přidělené adresy od DNS serveru. Všechny následující části jsou popisovány v režimu připojení přes ethernetový kabel s pevnou IP adresou.

4.3 Připojení k zařízení

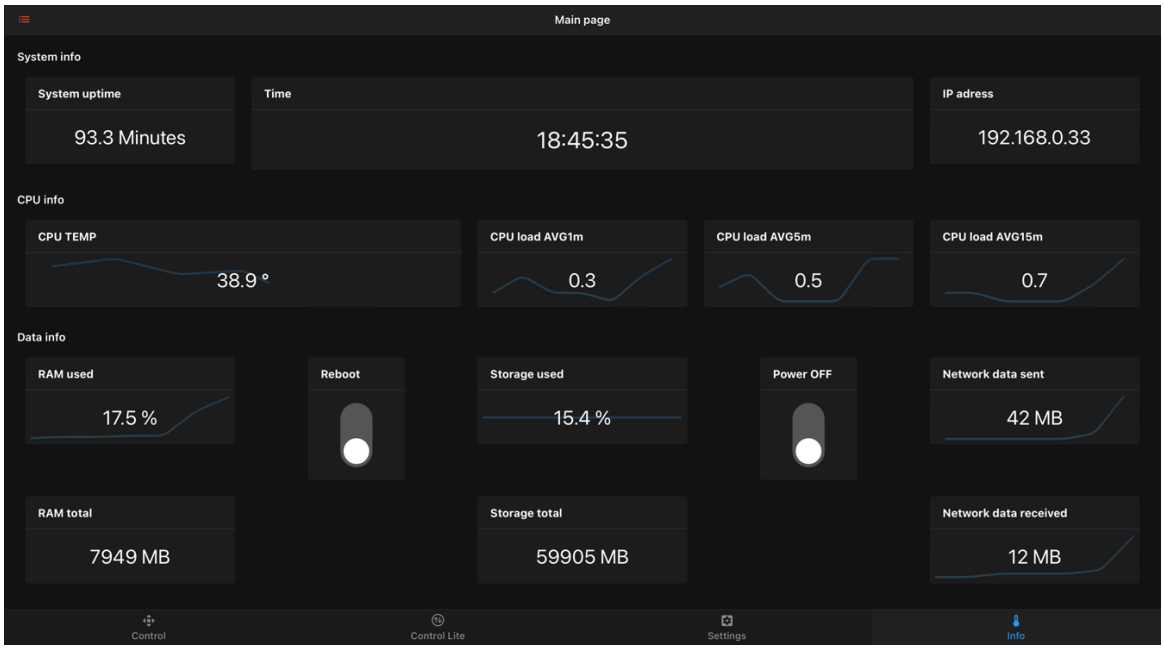
4.3.1 K Raspberry Pi pomocí SSH

Pro připojení k RPI pomocí SSH je třeba otevřít terminál a do příkazové řádky napsat **„ssh openhabian@192.168.0.10“,** následně nás terminál vyzve k zadání hesla. Po úspěšném připojení získáme výpis, který je vidět na obrázku č. 22.

Obr. 22 Ukázka výstupu připojení přes SSH do Raspberry Pi

4.3.2 K uživatelskému rozhraní pomocí webového prohlížeče

Pro připojení k uživatelskému rozhraní je třeba být ve stejné síti a zadat do vyhledávače příslušnou IP adresu s portem 8443. Výsledný příkaz zadaný do vyhledávače při připojení ethernetovým kabelem je tedy **„192.168.0.10:8443“.** Po připojení nás přivítá uvítací obrazovka s důležitými údaji o systému, která je vidět na obrázku č. 23 a je totožná s obrazovkou info.



Obr. 23 Uvítací a info obrazovka v Open HAB

4.4 Filozofie návrhu

Filozofie postaveného zařízení je rozdělena na 3 hlavní části:

- **Backend –** hlavní řídící program napsaný v programovacím jazyce Python, který obstarává samotné řízení všech připojených driverů;
- **Komunikační rozhraní –** jako komunikační rozhraní mezi backend a frontendem je použit MQTT broker, díky kterému lze obousměrně komunikovat;
- **Frontend –** pro uživatelské rozhraní byl vybrán software OpenHAB, který je široce modifikovatelný a zároveň řeší spoustu nízko úrovňových věcí za designéra.

Program je psaný tak, aby nebylo třeba zadat jakoukoli vstupní hodnotu od uživatele. Vše stojí na inteligentním procesu inicializace, kdy software sám rozpozná připojené motory a podle nich provede příslušnou inicializaci. Pro systém není překážkou ani použití různě dlouhých os, nebo naopak pokaždé úplně jiného zapojení s různými rozměry. Jediné, co je potřeba dodržet, je adresa driveru motoru, která pevně přísluší každé ose, driveru a podle ní

software pozná, jakou má provést inicializaci a co je připojeno. Problém není ani přepojování motorů při zapnutém napájení. Avšak pokud se počet připojených driverů liší od počtu inicializovaných driverů, program přestane pohybovat motory a je nutné provést inicializaci znovu nebo zajistit původní zapojení připojených zařízení. Toto je ochranný prvek. Bez použití ochranného prvku a výpadku napájení by hrozilo poškození mechanické konstrukce. V uživatelské části nastavení lze zadat rozsah jednotlivých os v milimetrech pro přepočet procentuální absolutní hodnoty polohy na jednotky vzdálenosti. Minimální krok, který je možné udělat je 0,05 % z rozsahu každé osy. Tento limit jde jednoduchým nastavení v administrátorské části SW Open HAB změnit. Zařízení lze ovládat z jakéhokoliv místa pomocí počítače, tabletu, mobilu atd. Jediná podmínka je, aby zařízení bylo připojené ve stejné síti jako Raspberry Pi, na kterém ovládací software běží. Při využití zařízení, které podporují multi- touch, jde ovládat poloha u všech os najednou. Uživatelské rozhraní také obsahuje přepínač „Control“ u každé osy, a to právě z důvodu použití na dotykových zařízeních, kdy by mohlo dojít k nechtěnému posunutí osy nevhodným přejetím přes slider polohy. V případě vypnutého vypínače „Control“ u některé osy a posunutí slideru polohy se automaticky slider vrátí do aktuální polohy a chybný příkaz na změnu polohy se nevykoná.

4.5 Rozbor ovládacího softwaru

4.5.1 Python

V následující kapitole je proveden rozbor zajímavých částí programu. Jelikož je kód obsáhlý, jsou rozebrány jen ty základní části a zbytek kódu je obsažen v příloze. Celý program obsahuje celkem 11 skriptů a cca 1189 řádků kódu. Celý program je psán v prostředí Visual Studio Code.

Seznam vytvořených python skriptů:

- MM_Axis_class.py
- MM_byte_parser.py
- MM_initialization.py
- MM_main.py
- MM_mqtt.py
- MM_PoffReboot.py
- MM_range_calculator.py
- MM_slave_counter.py
- MM_switcher_axis.py

- MM_switcher_system.py
- MM_system_info_class.py

Rozbor zajímavých částí programu

Skript MM_main.py

Následující funkce rozdělí příchozí téma na jednotlivá slova a vybere, kam má zprávu poslat na zpracování:

```
# rozparsování topicu příchozí zprávy
def function(topic, value):
    topic = topic.split("/")
    if len(topic) != 5: return 0
    try:
        d = {
            "main": topic[0],
            "type": topic[1],
            "classification": topic[2],
            "name": topic[3],
            "variable": topic[4]}
    except:
        pass

    # spuštění příslušného switcheru dle typu tématu
    if d.get("type") == "System":
        switcher_system(d.get("variable"), d.get("name"), value)
    elif d.get("type") == "Control":
        switcher_axis(d.get("variable"), d.get("name"), value)
```

Následující funkce kontroluje aktuální polohu všech os a posílá jí do MQTT serveru:

```
# sledování aktuální polohy ve vlastním vlákně, nezávisle (asynchronně) na běhu hlavního programu
def get_actual_position(client: mqtt.Client, axis: Axis):
    axis.clear_axis_error() # průběžné mazání chyb, které zastavují motory (zejména serial error)
    current_position = axis.get_current_position() # zjištění aktuální pozice
    current_position_mm = round((int(axis.state_var["RangeMM"])/2) * (current_position/100),2) # přepočet aktuální
pozice na vzdálenost v mm

    var_topic_ap = f'{topic_path_main}/{topic_path_control}/{topic_path_classification}/{axis.name}/ActualPosition'
    var_topic_ap_mm =
f'{topic_path_main}/{topic_path_control}/{topic_path_classification}/{axis.name}/ActualPositionMM'
    client.publish(var_topic_ap, str(current_position)) # odeslání aktuální pozice do MQTT
    client.publish(var_topic_ap_mm, str(current_position_mm)) # odeslání aktuální pozice do MQTT
    # Přiřazení Actual Position
    axis.state_var["ActualPosition"] = float(current_position)  #záznam aktuální pozice do parametru dané osy
    axis.state_var["ActualPositionMM"] = float(current_position_mm) # záznam aktuální pozice do parametru dané osy
    threading.Timer(0.5, get_actual_position, (client, axis)).start() # vytvoření nového vlákna, které spustí tuto metodu
každých 500ms
```

Tato funkce průběžně kontroluje stav připojených zařízení na I²C sběrnici:

```
# sledování připojených zařízení na i2c, nezávisle (asynchronně) na běhu hlavního programu
def slave_info(client):
    device = slave_counter_i2c(0)
    msg = str(device).replace("{","").replace("}","").replace("'","").replace(",",";")
    client.publish("MIXANM/System/I2C/Info/SlaveI2CCounter",str(msg))
    slave_driver_control()
    threading.Timer(1, slave_info, (client,)).start() # vytvoření nového vlákna, které spustí tuto metodu každou 1s
```

Skript MM_byte_parser.py

Následující funkce je vytvořena pro naformátování vrácených informací z driveru na jednotlivé bity:

```
# naformátování vráceného několikabajtové hexa čísla na jednotlivé bity, pro možnost porovnání
def byte_parser(data):
    try:
        data = [data[i:i + 1] for i in range(0, len(data), 1)]
        new_data = []
        for i in range(0, len(data), 1):
            new_data.append([1 if x == '1' else 0 for x in "{0:08b}".format(int(hex(ord(data[i]))[2:], 16))])
        return list(chain.from_iterable(new_data))
    except:
        return -1

# definovaný formát dle datasheetu pro misc flag
def misc_flag_format(data):
    data = byte_parser(data)
    if data == -1:
        return -1
    keys = [
        "reserved7",
        "reserved6",
        "reserved5",
        "HomingActive",
        "ReverseLimitActive",
        "ForwardLimitActive",
        "PositionUncertain",
        "Energized"]
    return dict(zip(keys, data))
# definovaný formát dle datasheetu pro error status
def error_status_format(data):
    data = byte_parser(data)
    if data == -1:
        return -1
    keys = [
        "SafeStartViolation",
        "CommandTimeout",
        "SerialError",
        "RequiredInputInvalid",
        "KillSwitchActive",
```

```
    "LowVIN",
    "MotorDriverError",
    "InternationallyDeEnergized",
    "ERRLineHigh_or_reserved7",
    "ERRLineHigh_or_reserved6",
    "ERRLineHigh_or_reserved5",
    "ERRLineHigh_or_reserved4",
    "ERRLineHigh_or_reserved3",
    "ERRLineHigh_or_reserved2",
    "ERRLineHigh_or_reserved1",
    "ERRLineHigh_or_reserved0"]
  return dict(zip(keys, data))
```

Skript MM_Axis_class.py

Tento skript obsahuje kompletně nadefinovanou třídu osy se všemi parametry, která implementuje jednotlivé drivery. Pro velkou obsáhlost je celý skript pouze v příloze.

Skript MM_range_calculator.py

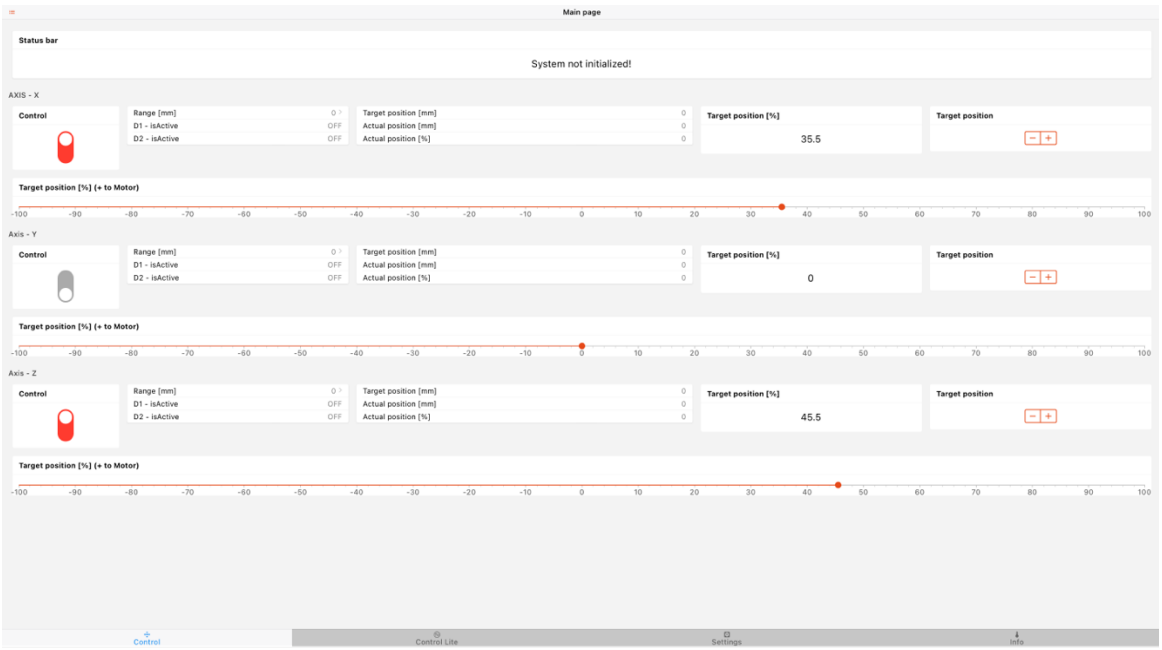
Následující skript je nejdůležitější v procesu inicializace. Tento skript slouží pro výpočet rozsahu připojené osy a odstraňuje velké množství chyb, které při inicializaci vznikají na I^2C komunikaci:

```
import statistics
# program pro výpočet rozsahu osy, který opravuje přijaté chyby a vybere správnou hodnotu
def range_calculator(data):
  try:
    if type(data) !=  type(list()): return 0 # vypni, pokud není správný typ
    if len(data) == 0: return 0 # vypni, pokud nejsou žádná data

    # převedení vstupních dat na absolutní hodnotu
    data_c = []
    for x in range(len(data)):
      data_c.append(abs(data[x]))

    data_c.sort() # seřazení vstupních dat podle velikosti
    data_c.remove(0) # odstranění nuly

    # vytvoří rozdíly sousedících hodnot pro vytvoření mediánu rozdílů
    rozdily = []
    for x in range(len(data_c)-1):
      rozdily.append(data_c[x+1] - data_c[x])
    median_rozdilu = statistics.median(rozdily) # medián rozdílů, pro zjištění trendu

    treshold = 2
    index_to_pop = []
    for x in range(len(rozdily)):
      if rozdily[x] > (median_rozdilu * treshold): # pokud jsou nějaké hodnoty větší než medián * treshold(2), označ je za
chybné pro odstranění
        index_to_pop.append(x+1) # zapíše index, který se má vymazat z data_c
```

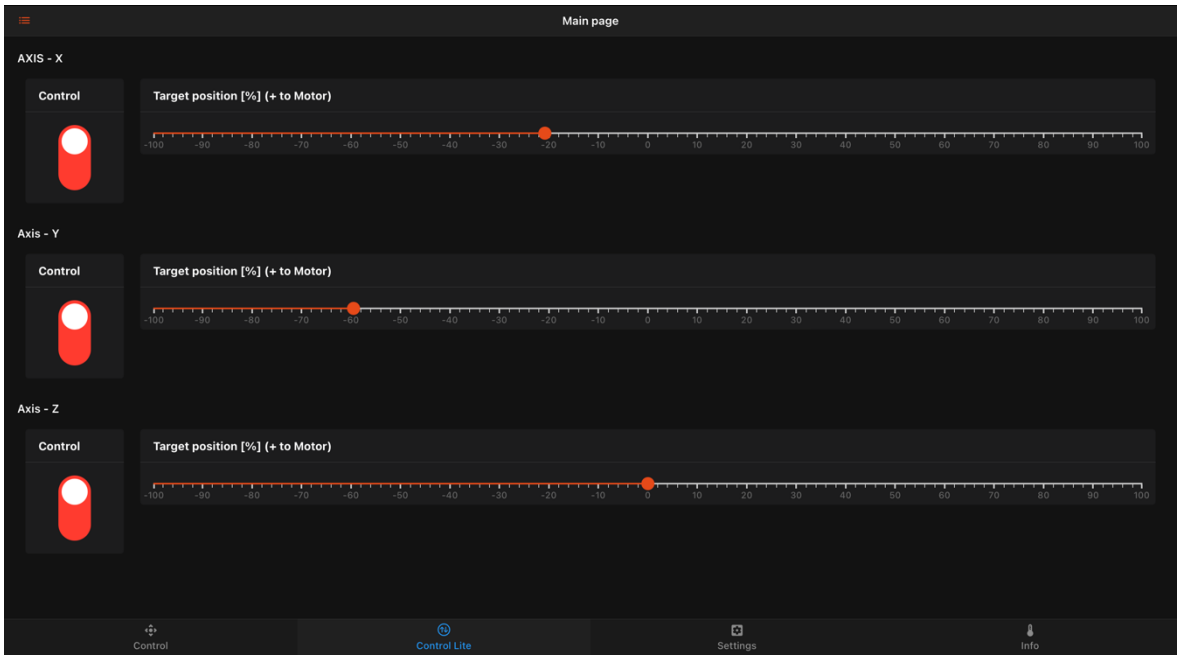
```
        tr = 0.8 # treshold pro počet zbývajících dat 80%
        if len(index_to_pop) != 0:
            burden = index_to_pop[-1]/(len(data_c)-1) # výpočet "zatížení", zbývající data po vymazání všech dat od
posledního extrému
            if burden < tr:
                return 0
        # smazání chybných dat
        for i in range(len(index_to_pop)-1, -1, -1):
            if i == len(index_to_pop)-1:
                for j in range(len(data_c)-1,index_to_pop[i] -1, -1):
                    data_c.pop(j)
                continue
            data_c.pop(index_to_pop[i])
        data_c = max(data_c)
        return data_c
    except:
        return 0
```

4.5.2 OpenHAB

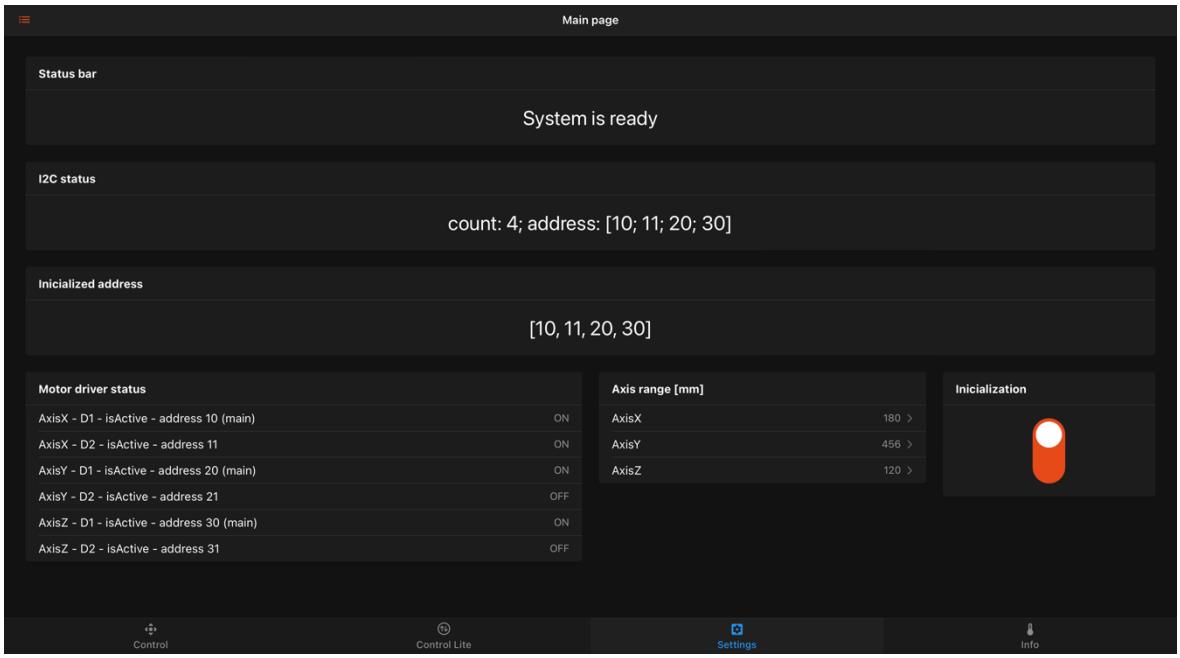
V softwaru OpenHAB bylo kompletně vytvořeno uživatelské rozhraní pro ovládání zařízení. V tomto rozhraní se nachází několik stránek, které jsou vidět na obrázcích č. 24, 25 a 26. OpenHAB běží jako samostatný server a obousměrně komunikuje přímo s MQTT brokerem.



Obr. 24 Open HAB, hlavní ovládací stránka, světlý režim



Obr. 25 Open HAB, zjednodušené ovládání, tmavý režim

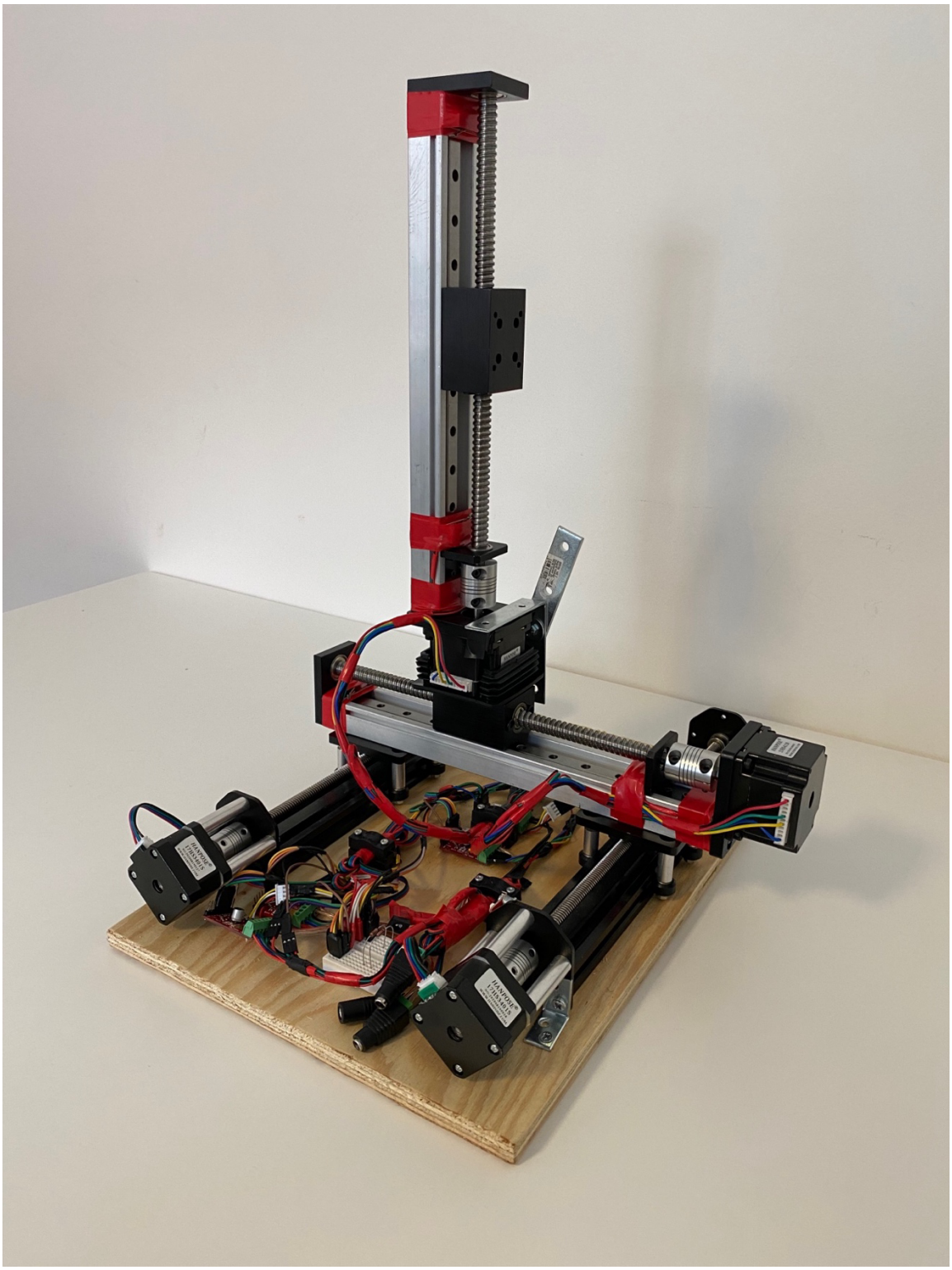


Obr. 26 Open HAB, stránka pro nastavení a inicializaci

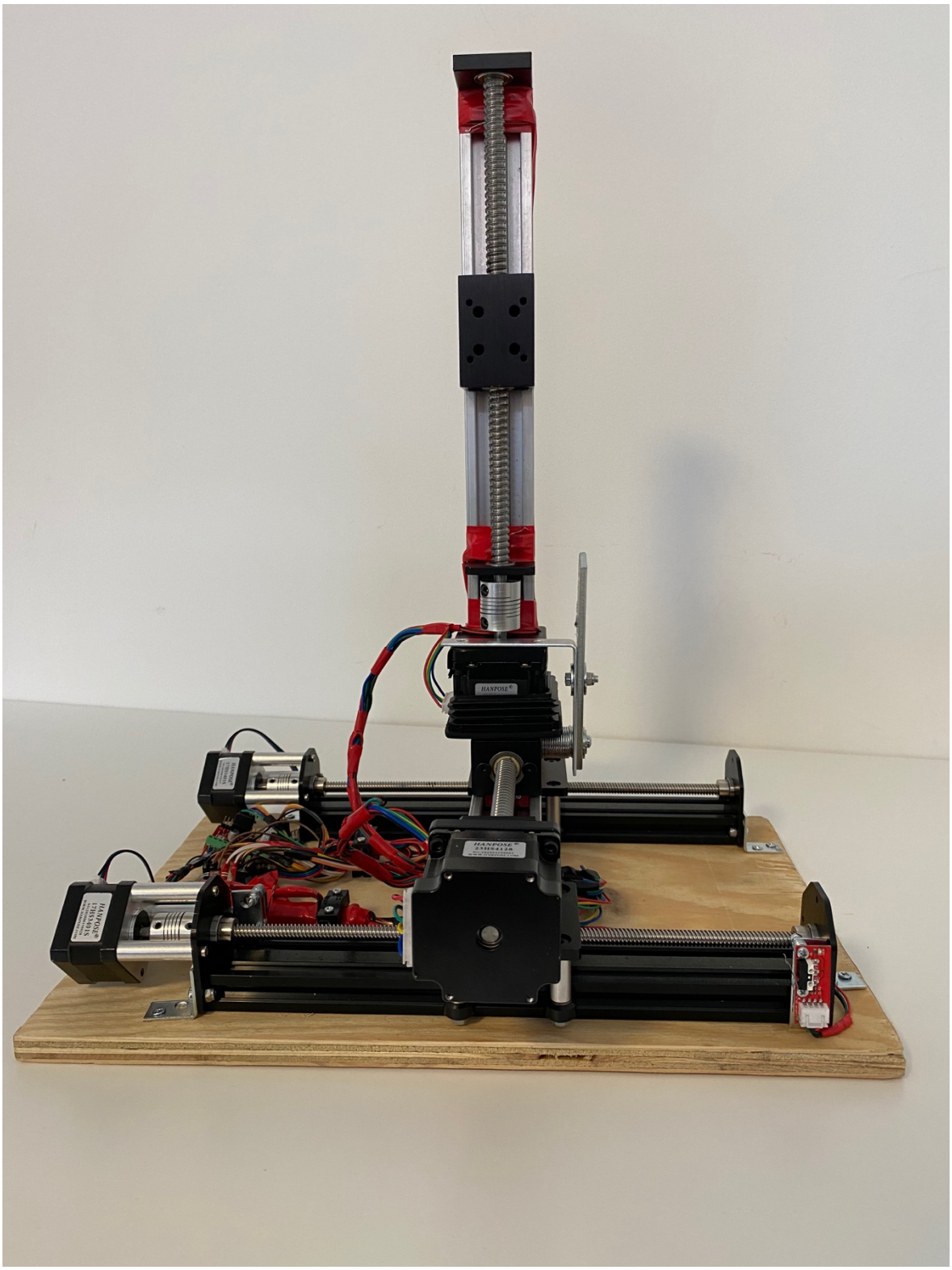
4.6 Mechanická konstrukce

Prototyp samotné konstrukce má celkem 4 motory s pojezdy, 2x každý výše popsaný model z tabulky č. 7. Jedna osa je synchronní (současně řízené 2 motory), a to osa X, pro větší tuhost. Sestavení bylo provedeno jen jako prototypové na dřevěnou desku o rozměru 42,5 cm podél osy X a 28,5 cm podél osy Y. Osa X i osa Y jsou horizontální a osa Z je vertikální. Celková výška konstrukce je 49,5 cm. Každý samostatný motor má 2 koncové spínače na každé straně, které slouží především pro inicializaci. Díky tomuto postupu lze

připojit libovolně dlouhé osy bez úpravy softwaru, neboť si program vždy při inicializaci změří rozsah každé osy automaticky. U motoru, který je synchronní není potřeba žádný koncový spínač, protože je naprosto závislý na hlavním motoru, který koncové spínače má (za předpokladu užití stejného motoru).



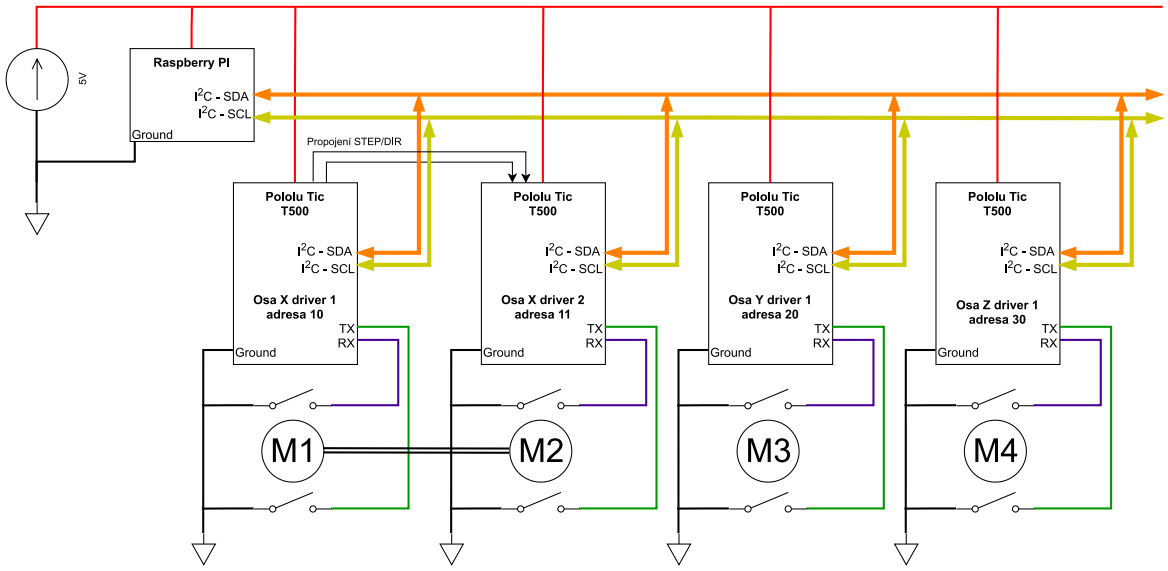
Obr. 27 Prototyp konstrukce víceosého pozicionéru, pohled 1



Obr. 28 Prototyp konstrukce víceosého pozicionéru, pohled 2

4.7 Blokové schéma zařízení

Následující blokové schéma zobrazuje zapojení zařízení s RPI a čtyř driverů s motory a koncovými spínači. Motory M1 a M2 jsou pevně mechanicky spojeny (dvojitá čára mezi M1 a M2). Aby vše fungovalo, musejí být piny STEP/DIR driverů s adresou 10 a 11 propojeny 1:1 (STEP -> STEP, DIR -> DIR). Motor M3 a M4 je vždy samostatná nezávislá osa. Piny na RPI, které byly použity pro připojení I^{2}C jsou 27 pro SDA a 28 pro SCL a lze je najít na obrázku č. 15. Linky SCL a SDA mezi RPI a drivery jsou propojeny 1:1 tzn. SDA -> SDA a SCL -> SCL. V tomto zapojení je vysoká úroveň linek I^{2}C určena akceptovatelným napětím na vstupech RPI a to je 3,3 V. Drivery mohou pracovat i na 5 V, ale tím by se poškodily GPIO vstupy na RPI. Jako zdroj u RPI je použit originální zdroj od Raspberry Pi s výstupním napětím 5,1 V a výstupním proudem 3 A s USB-C konektorem. Pro napájení motorů a driverů bylo použito čtyř 10 W adaptérů s napětím 5 V a proudem 2 A. Při testování bylo zkoušeno i odpojení a opětovné připojení I^{2}C sběrnice a vše fungovalo bez problémů. Lze tedy jako první připojit k napájení RPI a následně připojit drivery dle potřeby, bez nutnosti restartu nebo odpojování od napájecího napětí. Po přepojení je nutné provést inicializaci, pokud se změnilo uspořádání. Při stejném uspořádání není potřeba dělat nic navíc.



Obr. 29 Blokové schéma zapojení zařízení

5 Zhodnocení

Výsledkem této práce je hotový a plně funkční prototyp víceosého pozicionéru pro vzdálené ovládání detektoru ionizujícího záření. Dle zadání byl navrhnut celý systém pro vzdálené ovládání a po prozkoumání možností, bylo cíleně odstoupeno od použití komunikace s jednotlivými drivery pomocí rozhraní USB, protože se toto neukázalo jako dlouhodobě spolehlivé a funkční. Bylo zjištěno, že v čase může docházet ke špatné kompatibilitě ovladačů a zařízení se tak stane nefunkční. Na základě těchto zkušeností byla vymyšlena nová filozofie řízení za použití prvků IoT. Řešení s IoT prvky se při testování osvědčilo a umožnilo použití dalších funkcionalit v budoucnu, které by při použití USB komunikace byly komplikovaně dosažitelné. Při řešení bylo diskutováno možné zpoždění ovládání o několik sekund, z důvodu použití MQTT, které se po testování zamýšleného použití ukázalo jako zanedbatelné. Reakce doby odezvy na podnět v uživatelském prostředí nehraje roli a ovládání je uživatelsky příjemné. Větší zpoždění při ovládání je způsobeno mechanicky, kdy řízení motorů má nastavené určité maximální zrychlení, rychlost a zpomalení, aby byl pohyb co nejvíce plynulý.

Díky použití stejných integrovaných obvodů a jejich vstupně/výstupních pinů STEP/DIR pro řízení synchronních motorů, má samotné řízení zanedbatelnou latenci mezi jednotlivými motory. Ta vznikne samotným propojením, při použití obyčejného vodiče dlouhého cca 10 cm, je zpoždění mezi signály přibližně 0,5 ns (při rychlosti šíření signálu 65 % rychlosti světla). Prakticky to funguje tak, že hlavní driver dostane příkaz na změnu polohy a v reálném čase posílá identický řídicí signál, směru (DIR) a kroku (STEP), do druhého synchronního driveru. Toto je použito na základě doporučeného zapojení výrobce.

Použitý systém OpenHAB pro ovládání má uživatelsky přívětivé předdefinované prvky, které byly použity a jsou funkční. Pro větší variabilitu nastavení těchto prvků by bylo nutné napsat vlastní uživatelské rozhraní komunikující s MQTT serverem. Toto lze v budoucnu nadále rozvíjet díky použití MQTT serveru a připraveného API. V případě dalšího vývoje tímto směrem tedy nebude žádné omezení a lze používat i více uživatelských rozhraní, dle aktuální potřeby. Ovládací software napsaný v Pythonu odpovídá zadání a je plně funkční. Výsledné řešení umožňuje plnohodnotné pozicování a uplatnění může najít zejména pro řešení s úsporou finančních prostředků při zachování běžné funkcionality.

Při budoucím vývoji zařízení by bylo zajímavé přidat další funkce a věnovat se následujícím tématům, na které z časových důvodů nebyl prostor, zejména: uložení polohy,

uložení nastavení po vypnutí, automatické programy pozic a pohybů, rozšíření na více os, včetně os rotačních, možnost zpracovávat kódy pro CNC, rozšířit ovládání, optimalizace inicializačního procesu, optimalizace výkonu, optimalizace zatížení řídicího RPI, rozšíření ovládání na cloudové MQTT servery, vylepšit bezpečnostní procesy při krizových stavech zařízení, dokončit zařízení do fáze komerčního produktu, přidat motory se zpětnou vazbou pro přesnější ovládání, plnohodnotná konfigurace z uživatelského rozhraní bez nutnosti dalšího softwaru, automatické určení polohy z manuálního pohybu, vylepšení samotné konstrukce prototypu na komerční produkt atd.

Literatura

[1] Co jsou to CNC obráběcí stroje? [online]. [cit. 2022-05-21]. Dostupné z: https://www.profika.cz/co-jsou-to-cnc-obrabeci-stroje

[2] CNC stroje [online]. [cit. 2022-05-21]. Dostupné z: https://cs.wikipedia.org/wiki/CNC_stroje

[3] Základní školení 3D tisku [online]. [cit. 2022-05-21]. Dostupné z: https://www.youtube.com/watch?v=0UQyRxaBYQ4&t=1s

[4] TOP 5 NEUVĚŘITELNÉ VĚCI VYTIŠTĚNÉ NA 3D TISKÁRNĚ [online]. [cit. 2022-05-21]. Dostupné z: https://www.youtube.com/watch?v=3LTpuzXl7Yc&t=337s

[5] Visual Studio Code [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://code.visualstudio.com

[6] SSH – bezpečné používání vzdáleného počítače a kopírování dat [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.dsl.cz/jak-na-to/jak-na-ssh

[7] Lekce 1 - SSH – Konfigurace a připojení [online]. 2015 [cit. 2022-05-21]. Dostupné z: https://www.itnetwork.cz/site/ssh/ssh-konfigurace-a-pripojeni

[8] Python v ČR [online]. n.d. [cit. 2022-05-21]. Dostupné z: https://python.cz

[9] Python [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.python.org

[10] Python Tutorial [online]. W3schools, 2022 [cit. 2022-05-21]. Dostupné z: https://www.w3schools.com/python/

[11] INTERNET VĚCÍ (IOT): DEFINICE, PŘÍKLADY VYUŽITÍ, PRODUKTY [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.rascasone.com/cs/blog/iot-internet-veci-definice-produkty-historie

[12] What is the internet of things (IoT)? [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT

[13] MQTT: The Standard for IoT Messaging [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://mqtt.org

[14] Protokol MQTT: komunikační standard pro IoT [online]. 2016 [cit. 2022-05-21]. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/

[15] Co je MQTT a k čemu slouží ve IIoT? Popis protokolu MQTT [online]. n.d. [cit. 2022-05-21]. Dostupné z: https://ipc2u.cz/blogs/news/mqtt-protokol?_pos=1&_sid=6c335f472&_ss=r

[16] MQTT Broker [online]. n.d. [cit. 2022-05-21]. Dostupné z: http://www.whitesoft.cz/reseni/telekomunikace/mqtt-broker

[17] OpenHAB empowering the smart home [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.openhab.org

[18] Seriál OpenHAB v inteligentním domě [online]. n.d. [cit. 2022-05-21]. Dostupné z: https://www.root.cz/serialy/openhab-v-inteligentnim-dome/

[19] OpenHABian – Hassle-free openHAB Setup [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.openhab.org/docs/installation/openhabian.html

[20] Instalace openHAB na Raspberry Pi 3 [online]. n.d. [cit. 2022-05-21]. Dostupné z: https://chytra-obec.cz/wp-content/uploads/2018/01/01-instalace-na-rpi3.pdf

[21] UM10204 I2C-bus specification and user manual Rev. 7.0–1 [online]. 2021 [cit. 2022-05-21]. Dostupné z: https://www.pololu.com/file/0J435/UM10204.pdf

[22] Sériová rozhraní SPI, Microwire, I2C a CAN [online]. 2002 [cit. 2022-05-21]. Dostupné z: http://home.zcu.cz/~dudacek/NMS/Seriova_rozhrani.pdf

[23] I²C [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://cs.wikipedia.org/wiki/I²C

[24] DATASHEET Raspberry Pi 4 Model B Release 1 [online]. 2019 [cit. 2022-05-21]. Dostupné z: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[25] Raspberry Pi 4 Model B Default GPIO Pinout with PoE Header - Documents - Raspberry Pi - element14 Community element14 Community Raspberry Pi 4 Model B Default GPIO Pinout with PoE Header - Documents - Raspberry Pi - element14 Community [online]. [cit. 2022-05-21]. Dostupné https://community.element14.com/products/raspberrypi/w/documents/4317/raspberry-pi-4-model-b-default-gpio-pinout-with-poe-header

[26] Tic T500 USB Multi-Interface Stepper Motor Controller (Connectors Soldered) [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.pololu.com/product/3134

[27] 4.13. Setting up STEP/DIR control [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://www.pololu.com/docs/0J71/4.13

[28] HANPOSE 17HS3401S 34mm-Nema 17 Stepper-Motor [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://usa.banggood.com/HANPOSE-17HS3401-S-34mm-Nema-17-Stepper-Motor- 42-Motor-42BYGH-1_3A-28N_cm-4-lead-for-CNC-Laser-3D printer-p- 1415422.html?cur_warehouse=CN

[29] HANSPOSE 23HS4128 34mm-Nema 17 Stepper-Motor [online]. 2022 [cit. 2022-05-21]. Dostupné z: https://usa.banggood.com/HANPOSE-23HS4128-S-34mm-Nema-23-Stepper-Motor- 42-Motor-42BYGH-1_2,8A-28N_-1435462.html?cur_warehouse=CN

Seznam obrázků

Seznam tabulek

Elektronické přílohy

Příloha A – Skripty_Python.zip