



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH SYSTÉMU PRO DÁLKOVÉ SPOUŠTĚNÍ DOPRAVNÍKŮ

DESIGN OF A SYSTEM FOR REMOTE STARTING OF CONVEYORS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. David Strašák

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Martin Formánek

BRNO 2025

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	Bc. David Strašák
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	Ing. Martin Formánek
Akademický rok:	2024/25

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh systému pro dálkové spouštění dopravníků

Stručná charakteristika problematiky úkolu:

Tato práce je zpracovávána pro oddělení společnosti Honeywell, které se zaměřuje na dopravníkové systémy ve skladech. Součástí každé zakázky je kontrola, že jsou dopravníky mechanicky správně nainstalované. Zaměstnanci, kteří kontrolu provádí často nemají dostatečné znalosti potřebné pro ovládání dopravníků.

Podstatou této práce je tvorba systému, který umožní zaměstnancům provádějící kontroly ovládat dopravníky fyzicky i na dálku (v řádech desítek metrů). Systém bude obsahovat zařízení, bude ovládat dopravníky skrz mikrokontroler pomocí vstupně–výstupních signálů ovládacího panelu frekvenčního měniče. Tenhle mikrokontroler bude komunikovat s nadřazenou jednotkou (mobilní aplikací) pomocí bezdrátového připojení.

Cíle bakalářské práce:

1. Navrhněte obvod, který umožňuje lokální i dálkové ovládání dopravníků pomocí mikrokontroleru.
2. Navrhněte a realizujte desku plošných spojů s mikrokontrolerem.
3. Vytvořte uživatelskou aplikaci, která umožní ovládání mikrokontroleru na dálku.
4. Ověřte funkčnost navrženého systému.

Seznam doporučené literatury:

VALÁŠEK, M.: Mechatronika, Vydavatelství ČVUT 1995.

Santos, R.A. & Block, A.E.. (2012). Embedded systems and wireless technology.

ZÁHLAVA, Vít. Metodika návrhu plošných spojů. 1. Praha: Vydavatelství ČVUT, 2000. ISBN 80-01-2193-9

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2024/25

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc.
ředitel ústavu

doc. Ing. Jiří Hlinka, Ph.D.
děkan fakulty

Abstrakt

...Abstrakt...

Summary

...anglicky...

Klíčová slova

...klíčová slova...

Keywords

...anglicky...

Bibliografická Citace

STRAŁÁK, D. *Návrh systému pro dálkové spouštění dopravníků*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2025. 59 s., Vedoucí diplomové práce: Ing. Martin Formánek.

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práce (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

David Strašák

Brno

Tímto bych chtěl poděkovat všem, kteří se radou nebo jakoukoliv pomocí podíleli na vzniku této bakalářské práce. Především panu Ing. Martinu Formánkovi, za odborné vedení, odpovědi na mé dotazy a za cennou zpětnou vazbu při návrhu a tvoření této bakalářské práce. Velké díky za podporu také patří mé partnerce, rodině a všem přátelům.

David Strašák

Obsah

Úvod (co znamenají jednotlivé barvy v osnově plus)	9
1 Rešerše	10
1.1 Frekvenční měniče a jejich role v řízení dopravníků	10
1.1.1 Jak frekvenční měniče fungují	11
1.1.2 Sinamics G120D	13
1.1.3 Nastavení ovládacího panelu	14
1.1.4 Bezpečnostní aspekty práce s ovládacím panelem	16
1.2 Open-source vývojové desky	16
1.2.1 Proč WEMOS vývojové desky	17
1.2.2 Technické specifikace WEMOS D1 Mini Pro	17
1.2.3 Arduino framework pro ESP8266	19
2 Návrh zařízení	23
2.1 Princip funkce navrženého systému	23
2.1.1 Požadavky na systém	25
2.2 Hardware	26
2.2.1 Ovládání relé	28
2.2.2 LCD display s I2C převodníkem	29
2.3 Firmware ve vývojové desce	30
2.3.1 Třída a objekt ConveyorController	30
2.3.2 Stavový diagram logiky systému	34
2.4 Software v mobilní aplikaci ($\Sigma = 5$ stran)	38
2.4.1 Architektura aplikace	38
2.4.2 Použité knihovny a technologie v aplikaci (0.5 strany)	40
2.4.3 Princip komunikace s NodeMCU servery	41
2.4.4 Funkčnost aplikace (hodně stran)	44
2.4.5 Design aplikace (1 strana)	45
2.4.6 Konvertování webové aplikace do mobilní aplikace (0.5 strany)	45
2.5 Vytvoření schránky pro desku (1 strana)	47
2.6 Kompletace řešení (2 strany)	48
3 Ověření funkčnosti návrhu ($\Sigma = 4$ strany)	50
3.1 Ověření funkčnosti lokálního ovládání (0.5 strany)	50
3.2 Ověření funkčnosti mobilní aplikace (1 strana)	50
3.3 Ověření funkčnosti zapojení více desek (0.5 strany)	51
3.4 Posouzení z hlediska bezpečnosti (1 strana)	51

Závěr (1 strana)	53
Seznam zkratek a symbolů	54
Seznam zdrojů	55
Seznam obrázků	57
Seznam tabulek	58
Seznam příloh	59

Úvod (co znamenají jednotlivé barvy v osnově plus)

Zde jsou vysvětlivky barev v textu:

Modrou barvou jsou moje otázky na zkušenější akademiky - vedoucí nebo kdokoliv kdo tomu rozumí :)

Zelenou barvou jsou obecný popis o tom co v dané kapitole bude

Červenou barvou jsou informace o tom, kde seženu zdroje pro tuhle kapitolu. Budu rád když mi dáte vědět jestli jsou tyhle zdroje v pořádku, nebo ne.

Černou barvou jsou moje myšlenky co souvisí s kapitolami - je to základ textu bakalářky bez nějaké větší editace (ale snažil jsem se aby dávaly smysl).

Co znamená ta suma za nadpisy

Σ znamená kolik stran předpokládám, že bude mít daná kapitola až bude napsaný všechn text práce. Je to počet i s obrázky. Obrázků tolik není takže se nebojím že bych měl málo textu.

Názvy kapitol

Názvy kapitol jsou teď jenom nastřelené a nejspíš nezní dobře. Snažil jsem se spíš aby reflektovaly myšlenku kapitoly.

Vysvětlivka značení:

požadavek na systém je **tučný**

proměnné z kódu, metody a funkce jsou **typewriter**

1 Rešerše

1.1 Frekvenční měniče a jejich role v řízení dopravníků

Dopravníkové systémy byly kdysi pouze robustní mechanické konstrukce s jednoduchým asynchronním motorem napojený na jednu hodnotu síťového napětí a s nemotornou regulací rychlosti například pomocí přidáním odporu do sekundárního vinutí. V dnešní době jsme v éře průmyslu 4.0. a s tím je v každém mechatronickém systému důraz na automatizaci a digitalizaci spojených procesů. Díky velkému pokroku v oblasti výkonové elektrotechniky a řídících systémů vznikly nové možnosti precizního řízení otáček asynchronních motorů. Důležitým prvkem této transformace se staly frekvenční měniče - zařízení které umí na vstupu brát síťové napětí a na výstupu poskytovat jinou amplitudu a frekvenci napětí, což umožňuje efektivně řídit otáčky jakýchkoli asynchronních motorů. Tohle umožnilo vznik dopravníkových systémů u kterých je možné přesně a efektivně řídit otáčky. Když se k tomuto systému přidají ještě řídící systémy, je možné znát v každé chvíli polohu balíků na lince a inteligenčně tento tok balíků řídit.

Dopravníkové systémy, které společnost Honeywell vytváří jsou přesně takové inteligenční dopravníkové systémy. Cílem těchto systémů je pro zákazníky (většinou dopravní společnosti nebo například supermarkety) vytvořit systém, na který stačí vložit balík na jednom místě a tento balík už doputuje na místo kde má skončit. Řídící systém se postará o zbytek činností jako je třeba naskenování QR kódu na balíku, identifikace koncového bodu a řízení všech linek tak, aby nedošlo ke kolizím nebo nebezpečným událostem.

V současné době jsou tyhle jednotlivé dopravníky poháněné třífázovými asynchronními motory, které pomocí složitých převodů roztáčí celý dopravník (všechny jeho válečky nebo páš). Tyhle asynchronní motory jsou poháněné frekvenčními měniči a ty jsou pro většinu Honeywell dopravníkových systémů v dnešní době model G120D od značky Sinamics. Frekvenční měnič poskytuje výstup do asynchronního motoru, ale jedná se pouze o výkonovou část. Aby bylo možné frekvenční měnič řídit, je potřeba na něj připojit i ovládací panel, které je v běžné sestavě Honeywell dopravníkových systémů model CU240D-2 od značky Sinamics. Při běžném provozu je na tenhle ovládací panel připojená komunikační sběrnice PROFINET, která dává frekvenčnímu měniči ovládací příkazy. PROFINET je naprogramovaný přes Siemens TIA Portal (Total Integrated Automation Portal), což je prostředí vyvinuté od Siemens právě pro řízení různých frekvenčních měničů pomocí Siemens programovatelných logických automatů (PLC). V tomto programu je modelovaný tok na lince a pomocí toho PLC automaticky řídí dopravníkový systém. [1]

Zařízení, které v je v této bakalářské práci navrženo se ale nekoncipuje pro standardní provoz dopravníkových systémů, protože tam je systém už řízený PLC. Tento systém je navrhovaný pro zjednodušení procesu kontroly kvality instalace a funkčnosti dopravníků, který je konaný hned po instalaci dopravníků (které instaluje externí firma) v prostorách zákazníků. Jedná se hlavně o dynamické kontroly kvality mechanické instalace, kdy se na

1 REŠERŠE

1.1 FREKVENČNÍ MĚNIČE A JEJICH ROLE V ŘÍZENÍ DOPRAVNÍKŮ

každém dopravníku musí zkontolovat, že je schopný pohybu bez přílišného házení nebo vibrací kvůli špatné instalaci.

V kontextu těchto zkoušek není nezbytné, aby frekvenční měniče komunikovaly prostřednictvím řídicího systému Siemens PLC. Opak je pravdou - zde je inicializace PLC sítě mezi dopravníky spíše extra úkol, což jsou schopnosti které často zaměstnanci kontrolující mechanickou instalaci dopravníků nemají. Při těchto zkouškách se stává prioritní mít kontrolu nad individuálními dopravníky a mít možnost je ovládat a nastavovat na nich rychlosť dle libosti. Výhodou by zde při takovém ovládání byla i možnost implementace dálkového ovládání dopravníků.

1.1.1 Jak frekvenční měniče fungují

Jak již bylo naznačeno v kapitole 1.1, frekvenční měniče se pro řízení asynchronních motorů teoreticky používat nemusí, ale poté by měl dopravník jenom omezený výběr z nastavitelných rychlostí a celé řízení by bylo mnohem složitější. V dnešní době jsou frekvenční měniče technicky nejvhodnější způsob regulace motorů jak z hlediska technických parametrů (regulační rozsah a přesnost), tak i z energetického hlediska (regulace je bezeztrátová). Kvůli témtoto důvodům jsou frekvenční měniče tak časté. [2]

Frekvenční měnič Sinamics G120D je sice vektorově řízený, ale princip funkce frekvenčního měniče se dá lépe vysvětlit na měniči se skalárním řízením.

Rozdíl mezi těmito dvěma způsoby řízení spočívá v efektivitě. Vektorové řízení cíleně reguluje proud v cívkách asynchronního motoru tak, aby statorové magnetické pole bylo prostorově optimálně natočené vůči poli rotorovému (úhel závisí na počtu pólů). Díky tomu je dosaženo efektivnějšího pohonu rotoru požadovanou rychlostí a směrem. Skalární řízení naopak tento vzájemný úhel nesleduje, a proto není z hlediska řízení optimální. Vektorové řízení je zkrátka složitější, ale efektivnější a má další výhodu že umožňuje přímé řízení momentu. [2]

Funkce frekvenčního měniče vychází přímo z principu funkce asynchronního motoru. Při návrhu asynchronního motoru se navrhuje velikost sycení motoru které je určeno spřaženým magnetickým tokem statorového vinutí Ψ_S který je definován jako:

$$\Psi_S = N\Phi_S \quad (1.1)$$

kde N je počet závitů cívky na statoru a Φ_S je magnetický tok jednoho závitu cívky.

Aby frekvenční měnič mohl fungovat, musí být spřažený magnetický tok statorového vinutí konstantní. Tomu se říká **Podmínka konstantního sycení**. Statorové vinutí motoru je napájeno nějakým harmonickým napětím vycházející z frekvenčního měniče o tvaru:

$$U_S(t) = U_{max} \sin(\omega_s t) \quad (1.2)$$

kde U_S je napětí na statoru, U_{max} je amplituda statorového napětí a ω_s je úhlová frekvence napájecího napětí. [3]

Pokud zanedbáme statorový odpor a budeme tedy uvažovat, že celé statorové napětí u_L bude na indukčnosti motoru, bude maximum spřaženého magnetického toku ve statoru rovné: [3]

$$\Psi_S = \int_0^{T/2} u_L dt \quad (1.3)$$

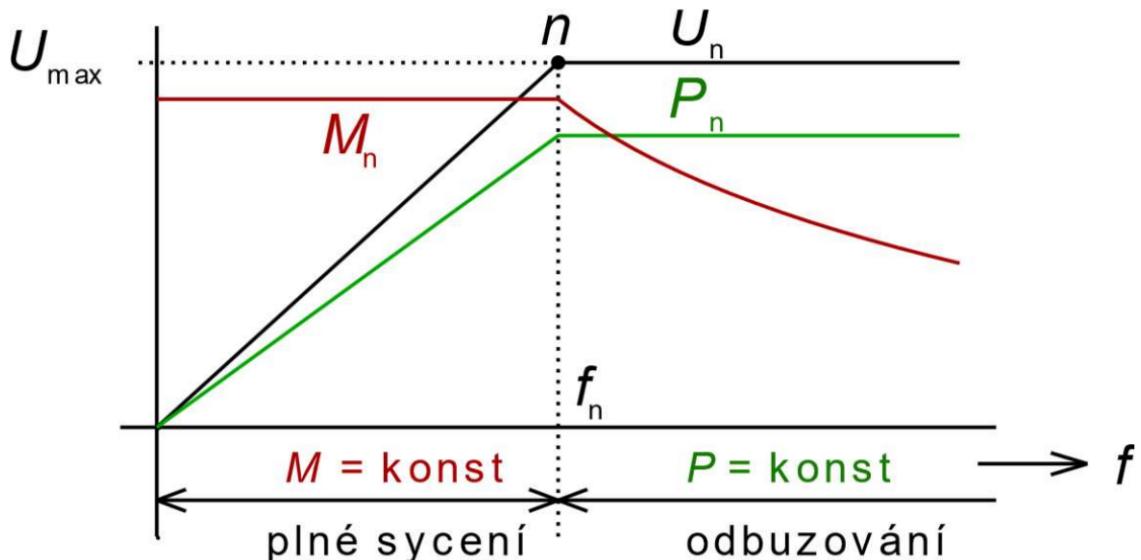
1 REŠERŠE

1.1 FREKVENČNÍ MĚNIČE A JEJICH ROLE V ŘÍZENÍ DOPRAVNÍKŮ

Princip podmínky konstantního sycení tedy spočívá v tom, že chceme mít konstantní spřažený magnetický tok. Tohle se dělá z důvodu, že na sycení motoru závisí například magnetizační proudy. Křivka sycení není lineární a má bod zvratu, kdy se menší změna sycení projeví ve mnohem větším zvýšení magnetizačního proudu než tomu bylo před bodem zvratu. Konstantní sycení je nastaveno proto, abychom zůstali před bodem zvratu a díky tomu bude magnetizační proud růst pomaleji a rozumně.

Režimy funkce frekvenčního měniče

Asynchronní motor, který je napájen z frekvenčního měniče má dva provozní režimy ve kterých se může nacházet. Těmi jsou oblast konstantního momentu a oblast konstantního výkonu zobrazené v grafu 1.1.



Obrázek 1.1: Závislost napětí, momentu a výkonu na frekvenci [3]

V levé části grafu je oblast konstantního momentu s plným sycením motoru. Zde platí podmínka definovaná v rovnici 1.3 o konstantním spřaženém magnetickém toku ve statoru. Díky tomu je moment na motoru konstantní a postupně motoru roste výkon, který je definovaný jako:

$$P = M\omega \quad (1.4)$$

až do maximální hodnoty výkonu která je v bodě n - jmenovitý bod motoru.

Je také dobré podotknout, že levá část grafu nemůže jít takto od nulového napětí (tentotograf je spíše idealizovaný případ), ale jde zpravidla od 10% jmenovité hodnoty napětí, jelikož se musí pokrýt ztráty které vznikají na odporu statorového vinutí R_S . [3]

V pravé části grafu je oblast konstantního výkonu ve kterém se motor odbuzuje. Zde už není splněna podmínka z rovnice 1.3 a tak motoru klesá moment. Vzhledem k tomu, že frekvence statorového napětí stále roste, tak rostou stále i otáčky rotoru.

1.1.2 Sinamics G120D

Sinamics G120D je decentralizovaný frekvenční měnič designovaný pro buzení motorů od dopravníkových systémů po elektrické monoraily. Slovo decentralizovaný zde znamená, že frekvenční měnič není jeden centralizovaný, ale že je víc menších frekvenčních měničů blízko u motorů, které ovládají. Kvůli tomu má i certifikaci IP65, která zaručuje dostatečnou kvalitu zpracování, aby bylo možné mít tento frekvenční měnič v náročných prostředí skladů zákazníků firmy Honeywell. [1]

Frekvenční měnič obsahuje funkce jako je přesné nastavení polohy motoru, bezpečnostní funkce a dobře konfigurovatelné digitální a analogové vstupy a výstupy. Je to standardní frekvenční měnič který je používán v různých aplikacích hlavně firmami které fungují jako systémoví integrátoři. Běžná podoba tohoto frekvenčního měniče (s kontrolním panelem CU240D-2) je na obrázku 1.2. [1]



Obrázek 1.2: Frekvenční měnič Sinamics G120D [1]

Sinamics G120D se v rámci systémů společnosti Siemens dá používat s třífázovými asynchronními motory řad Simotics GP (General Purpose) a Simotics SD (Severe Duty). Jak už název napovídá tak v případě společnosti Honeywell je jako motor nejčastěji používán Simotics GP. Napájecí napětí frekvenčního měniče je také třífázové od 380V do 500V dle konfigurace motoru a frekvenční měniče se vyrábí s výkonem 0,75kW až 7,5kW.

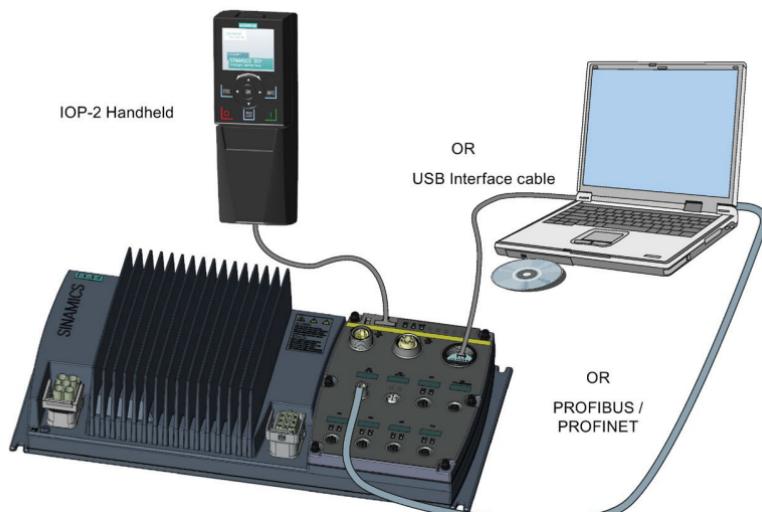
Tento frekvenční měnič je tvořen dvěma hlavními částmi - výkonová část a ovládací panel. Ovládací panel ovládá a monitoruje výkonovou část frekvenčního měniče pomocí několika kontrolních systémů na bází uzavřených smyček a díky tomu může kontrolovat bezpečný stav frekvenčního měniče a taky znemožnit ovládání, pokud by s měničem bylo něco v nepořádku. Také je schopný rekuperace energie z brzdění linek a vracet ji do sítě, což zákazníkům snižuje náklady na provoz. [1]

Všechny tyto funkcionality je možné ovládat přes průmyslové sběrnice PROFINET, PROFIBUS anebo běžnou sběrnici EtherNet. Tímto způsobem dává PLC příkazy frekvenčnímu měniči v běžném režimu ovládání dopravníků.

1.1.3 Nastavení ovládacího panelu

Jak bylo dříve zmíněné frekvenční měnič má vždy nějaký ovládací panel. V případě Honeywell instalací je ovládací panel většinou typu Sinamics CU240D-2. Ovládací panely tohoto typu lze za chodu seřizovat třemi způsoby (zobrazené i v obrázku 1.3):

- Připojení USB z notebooku
- Použití PROFINET nebo PROFIBUS
- Použití zařízení IOP-2 Handheld



Obrázek 1.3: Způsoby seřizování ovládacího panelu frekvenčního měniče [4]

Vzhledem k předpokladu, že navrhovaný systém mají používat osoby, které nejsou inženýři specializovaní na kontrolní systémy, je nejvhodnější způsob jak seřizovat ovládací panel použít zařízení IOP-2 Handheld. Ostatní dvě metody vyžadují specializovaný software pro který by bylo zapotřebí instalovat a udržovat si pro něj licence. Naproti tomu IOP-2 Handheld představuje samostatné zařízení schopné provést veškerá potřebná nastavení a je dodáváno s optickým kabelem pro přímé připojení k ovládacímu panelu.

Pro účely navrhovaného systému je potřeba ovládací panel vyresetovat a nastavit do jednoho z možných výchozích nastavení. Zvolené výchozí nastavení ovlivňuje celý systém, protože ten ovládá dopravník tím, že pomocí relé spíná digitální vstupy ovládacího panelu frekvenčního měniče - tímto způsobem dává systém příkazy na spuštění dopravníku, zrychlení a zpomalení. Nesprávné nastavení ovládacího panelu by vedlo k tomu, že ačkoliv by systém generoval správné signály na digitálních vstupech, panel by je interpretoval chybně.

Navržený systém je optimalizován pro výchozí nastavení číslo 9, ve kterém jsou funkce digitálních vstupů definovány tímto způsobem:

- Digitální vstup 0: ON/OFF dopravníku
- Digitální vstup 1: Zrychlení dopravníku
- Digitální vstup 2: Zpomalení dopravníku

1 REŠERŠE

1.1 FREKVENČNÍ MĚNIČE A JEJICH ROLE V ŘÍZENÍ DOPRAVNÍKŮ

- Digitální vstup 3: Kvitování chyby

Systém bude konektory připojený k ovládacímu panelu a pomocí relé na desce plošných spojů bude ovládat dopravník rozpojování a zkratováním těchto digitálních vstupů. Digitální vstup č. 3 nebude v rámci systému využíván, protože kvitování případných chyb je vyžadováno pouze jednorázově při resetování do výchozích nastavení a lze jej provést přímo pomocí zařízení IOP-2 Handheld. [4].

Resetování ovládacího panelu do tohoto výchozího nastavení je možné provést přímo na místě pomocí zařízení Sinamics IOP-2 Handheld. Pro resetování stačí pouze připojit IOP-2 k ovládacímu panelu frekvenčního měniče, vybrat možnost pro zresetování nastavení ovládacího panelu a vybrat výchozí nastavení číslo 9. Zbytek hodnot na ovládacím panelu, jako je například zrychlující rampa, může zůstat na výchozích hodnotách, protože je není potřeba v rámci testování kvality instalace mít na správných hodnotách (frekvenční měnič funguje i tak). Poté je možné odpojit IOP-2 od ovládacího panelu, který tímto způsobem zůstane nastavený nadále.

Alternativní výchozí nastavení ovládacího panelu

Při návrhu systému byla kromě výchozího nastavení číslo 9 zvažována i další relevantní výchozí nastavení, konkrétně nastavení č. 8 a č. 12.

Výchozí nastavení č. 8 by definovalo chování systému následovně:

- Digitální vstup 0: ON/OFF dopravníku
- Digitální vstup 1: Zrychlení dopravníku
- Digitální vstup 2: Zpomalení dopravníku
- Digitální vstup 3: Kvitování chyby
- Digitální vstup 4: Při přerušení nouzově zastaví (E-STOP)
- Digitální vstup 5: Při přerušení nouzově zastaví (E-STOP)

Tohle výchozí nastavení nabízí stejnou funkcionalitu jako zvolené nastavení č. 9, ale vyžaduje připojení dalšího kabelu na kterém bude v obvodu umístěné bezpečnostní tlačítko E-STOP, které by také muselo mít dostatek místa ve schránce systému. Tohle výchozí nastavení bylo zamítnuto právě kvůli tomu, že E-STOP tlačítko vyžaduje neúměrně příliš mnoho místa a tak by to výrazně zvětšilo rozměry systému, což by zmenšovalo přenositelnost a jednoduchost používání. Bezpečnost systému je přitom zajištěna jinými bezpečnostními prvky přímo u dopravníků, jak je podrobněji popsáno v kapitole 3.4. [4]

Další zvažovanou alternativou bylo výchozí nastavení č. 12, které by definovalo funkce vstupů tímto způsobem:

- Digitální vstup 0: ON/OFF dopravníku
- Digitální vstup 1: Reverzace směru otáčení
- Digitální vstup 2: Kvitování chyby
- Analogový vstup: Nastavení rychlosti

Tohle nastavení by umožnilo připojení potenciometru k analogovému vstupu ovládacího panelu pro přímé nastavení rychlosti. Díky tomuto by bylo o jedno tlačítko méně na schránce systému, ale zároveň by to vyžadovalo speciální kabely pro připojení (vzhledem k tomu, že se analogové vstupy ovládacích panelů v instalacích Honeywell běžně nevyužívají) a také by to zahrnovalo modifikaci desky plošných spojů, například s využitím integrovaného obvodu digitálně řízeného potenciometru. [4]

Po zhodnocení těchto možností bylo vybráno výchozí nastavení č. 9.

1.1.4 Bezpečnostní aspekty práce s ovládacím panelem

Vzhledem k tomu, že frekvenční měnič pracuje ve výkonové části s usměrněným trojfázovým napětím, je velmi důležité dbát na bezpečnost při práci s frekvenčním měničem. Z tohoto důvodu je vedle každé instalace frekvenčního měniče v Honeywell umístěn bezpečnostní vypínač, který vypojuje napájení výkonové části frekvenčního měniče. Pro bezpečné zacházení s frekvenčním měničem je potřebné mít tento vypínač ve stavu rozpojeno.

Po rozpojení napájení výkonové části zůstává na ovládacím panelu napětí 24V. Tohle nízké napětí je ale chráněno šroubovacími gumovými krytkami, které zakrývají veškeré digitální vstupy a výstupy ovládacího panelu kde se toto napětí nachází.

1.2 Open-source vývojové desky

Mikrokontroller, mozek praktické části bakalářské práce, není integrován přímo na desce s výkonovou částí zařízení. Využita byla možnost open-source vývojové desky, přičemž toto označení je dnes často synonymem pro Arduino, značku s největším přínosem v této oblasti.

Myšlenka vývojových desek jako jsou Arduino desky začala myšlenkou minimalismu - desky nebyly nikdy převratné, ale obsahovaly přesně to, co je potřeba. Na jedné vývojové desce je obsažený mikrokontroller, převodník z USB do sériové komunikace a dle desky obsahuje další užitečné součástky. Je zde možné tedy nejenom využívat periferie použitého mikrokontrolleru, ale i další hardware, jako jsou externí krystaly, napájení mimo USB kabel a další na základě specifického návrhu vývojové desky. [5]

Rychlou adaptaci veřejnosti umožnil nejenom kvalitní design desek, ale i software pro programování Arduino desek na počítači. Na rozdíl od předchozího proprietárního softwaru, který nebyl dostupný pro všechny operační systémy, je programovací prostředí Arduina open-source a spustitelné na všech systémech s podporou Java aplikací. [5]

Kromě výhod ekosystému Arduino existují obecné důvody pro použití hotových vývojových desek namísto přímé integrace mikrokontrolleru v prototypování a vývoji. Mezi hlavní výhody patří možnost připojení vývojové desky pomocí kolíkových lišť, což umožňuje snadné vyjmutí pro přeprogramování nebo výměnu. Přímá integrace by v případě poruchy vyžadovala odpájení. Díky tomu vývojová deska celkově usnadňuje prototypování a opravitelnost.

Nakonec je důvodem zvolení open source vývojových desek do systému i jejich dostupnost a flexibilita kterou nabízejí. Jelikož jsou schémata zapojení desek veřejně dostupná, může je vyrábět jakýkoliv výrobce. Navíc je také možné používat veřejně dostupná schémata zapojení desek při návrhu vlastních desek plošných spojů do kterých jsou vývojové desky integrované a díky tomu známá přesná propojení jednotlivých komponentů v celé navržené desce plošných spojů.

1.2.1 Proč WEMOS vývojové desky

Arduino v dnešní době není jediná firma, která vyrábí open-source vývojové desky. Pro tuhle bakalářskou práci byla zvolena vývojová deska od společnosti WEMOS, která je výrobce vývojových desek které jsou podobné Arduino deskám, ale jejich zaměření je specificky ve vytváření kompaktních desek které mají integrovanou bezdrátovou konektivitu (WiFi a bluetooth) pomocí populárních mikrokontrolérů ESP32 a ESP8266 od společnosti Espressif Systems.

Mít možnost používat WiFi je důležitý požadavek, který musí vývojová deska splňovat. Pokud bude systém možné ovládat přes WiFi, je možné pro ovládání použít jakékoli zařízení, které má WiFi technologii, což je v dnešní době většina chytrých zařízení. Kvůli tomu lze dopravník ovládat širokým spektrem chytrých zařízení a tak není potřeba aby s sebou uživatelé nosili dedikovaný vysílač.

Společnost WEMOS nabízí několik modelů vývojových desek s různými mikrokontroléry a periferiemi. Mezi známé varianty patří například:

- **WEMOS D1 Mini:** Kompaktní deska postavená na mikrokontroléru ESP8266EX. Poskytuje 11 digitálních GPIO pinů (z toho 10] s podporou PWM a podporou přerušení), 1 analogový vstup, I2C rozhraní, 4MB Flash paměti a integrovanou PCB anténu pro WiFi. Napájení a programování se provádí přes USB-C konektor. Deska je oblíbená pro své malé rozměry a širokou podporu, ale omezuje ji malý počet GPIO pinů, což desku nedělá dobrou pro prototypování nebo rozsáhlejší aplikace.
- **WEMOS C3 Mini:** Novější varianta využívající mikrokontrolér ESP32-C3. Tento čip integruje WiFi i Bluetooth konektivitu. Deska disponuje USB-C konektorem, 4 MB Flash paměti a 12 GPIO pinů.

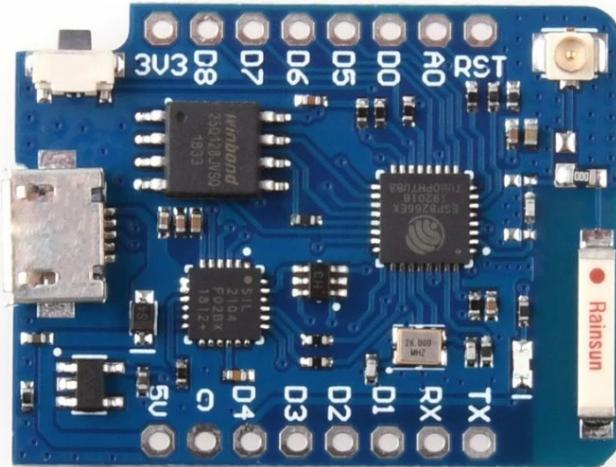
Tenho systém je ale navrhován pro industriální prostředí a je velmi důležité aby byla bezdrátová komunikace co nejspolehlivější. Proto je potřebné mít na vývojové desce k dispozici externí anténu, která významně zvýší dosah WiFi komunikace díky lepšímu umístění antény. Proto byla do systému vybrána vývojová deska WEMOS D1 Mini Pro kterou lze vidět na obrázku 1.4. Tato deska sdílí většinu vlastností s modelem D1 Mini, ale narozdíl od levnějšího modelu má 16MB Flash paměti, které jsou také velmi důležité vzhledem k tomu, že na kód pro systém by 4MB Flash paměti nestačilo.

Všechny vývojové desky WEMOS s mikrokontroléry ESP8266 a EPS32 lze programovat pomocí prostředí Arduino IDE (nebo alternativy jako PlatformIO) s využitím Arduino jazyka založeného na C++, nebo alternativně pomocí MicroPython. Celá aplikace je programována pomocí Arduino framework pro ESP8266. Webové ovládání je prováděno pomocí knihovny WebServer, která umožňuje běh nenáročných síťových aplikací (více je popsáno v kapitole 1.2.3).

1.2.2 Technické specifikace WEMOS D1 Mini Pro

Na základě požadavků projektu a srovnání s alternativami je pro realizaci hardwarového návrhu nejfektivnější vývojová deska WEMOS D1 Mini Pro. Tato sekce popisuje její technické parametry s jejich využitím v návrhu. Samotná deska je postavena na mikrokontroléru ESP-8266EX a je velmi kompaktní, zatímco ale stále poskytuje dost funkcionality a vstupních a výstupních pinů pro provoz zařízení.

Technické specifikace této vývojové desky jsou následující: [7, 8]



Obrázek 1.4: Použitá varianta vývojové desky WEMOS D1 Mini Pro [6]

- Mikrokontrolér: ESP-8266EX
- Napájecí napětí: 5V
- Provozní napětí: 3, 3V
- Počet digitálních I/O pinů (GPIO): 11. Tyto piny slouží pro digitální vstupní a výstupní signály které budou ovládat dopravník a na základě kterých se bude approximovat rychlosť dopravníku.
- Podpora periferií na GPIO pinech: Většina digitálních pinů podporuje funkce jako:
 - Přerušení (Interrupt): Umožňuje reakci mikrokontroléru na externí události.
 - PWM (Pulse Width Modulation): Pro generování semi-analogového signálu nebo snížení střední hodnoty napětí. Až 10 pinů má podporu PWM.
 - I2C: Dvouvodičová sériová sběrnice používaná pro komunikaci s periferiemi, jako je v tomto projektu použitý LCD displej. Deska disponuje dedikovanými piny pro tuto sběrnici na pinech D1 a D2.
 - One-wire: Sériová sběrnice pro komunikaci s některými typy senzorů.
- Analogový vstupní pin: 1. Tento pin umožňuje měřit analogové napětí, například z některých typů senzorů. Maximální vstupní napětí pro tento pin je 3.2V.
- Paměť:
 - Flash paměť: 16 MB. Tato velká kapacita Flash paměti je důležitá pro uložení aplikačního kódu, rozšiřujících knihoven (jako je WebServer) a webových souborů co jsou hostované na serveru.

- RAM: 50 kB. Slouží pro běh programu a ukládání proměnných.
- Bezdrátová konektivita: Integrovaná WiFi na frekvenci 2.4 GHz.
- Anténa: Možnost připojení externí antény prostřednictvím IPEX1 / SMA konektoru nebo využití vestavěné keramické antény pro testování. Pro zvýšení spolehlivosti a dosahu v industriálním prostředí je využita možnost externí antény.
- Napájení a programování: Micro USB konektor. Deska může být napájena přes USB nebo přes 5V pin.
- Napájení z baterie: Rozhraní pro připojení lithiové baterie s nabíjecím proudem až 500mA. Toto rozhraní ale v tomto projektu není využíváno a navržená deska plošných spojů není pro používání baterie uspořádána.
- Kompatibilita: Deska je kompatibilní s vývojovými prostředími a firmwary jako Arduino, MicroPython a NodeMCU, což poskytuje flexibilitu při vývoji firmwaru.

Zapojení vývojové desky do navrhnuté desky plošných spojů je popsáno v kapitole 2.2.

1.2.3 Arduino framework pro ESP8266

Pro programování mikrokontrolerů z řad EPS8266 je možné využít jejich nativního software development kitu (SDK) anebo takzvaný "Arduino framework" pro tuto platformu. Tato knihovna je portace SDK pro platformu Arduino a díky tomu je možné používat prostředí jako je Arduino IDE nebo Platformio pro programování ESP8266 mikrokontrolerů. Tohle všechno je možné i přesto, že mají ESP8266 i Arduino přirozeně různé základy, které spolu ve výchozím stavu nejsou kompatibilní. Tato podpora umožňuje velkému množství hobby i profesionálním programátorům programovat ESP8266 mikrokontrolery ve stejném prostředí jako ve kterém programovali své Arduino projekty. To všechno bez ztráty hardwarových nebo síťových periferií.

"Arduino frameworck pro ESP8266 je podpora ESP8266 mikrokontroleru pro Arduino prostředí. To vývojářům umožňuje používat známé Arduino funkce a knihovny, které je možné spouštět přímo na ESP8266 mikrokontrolleru." [9]

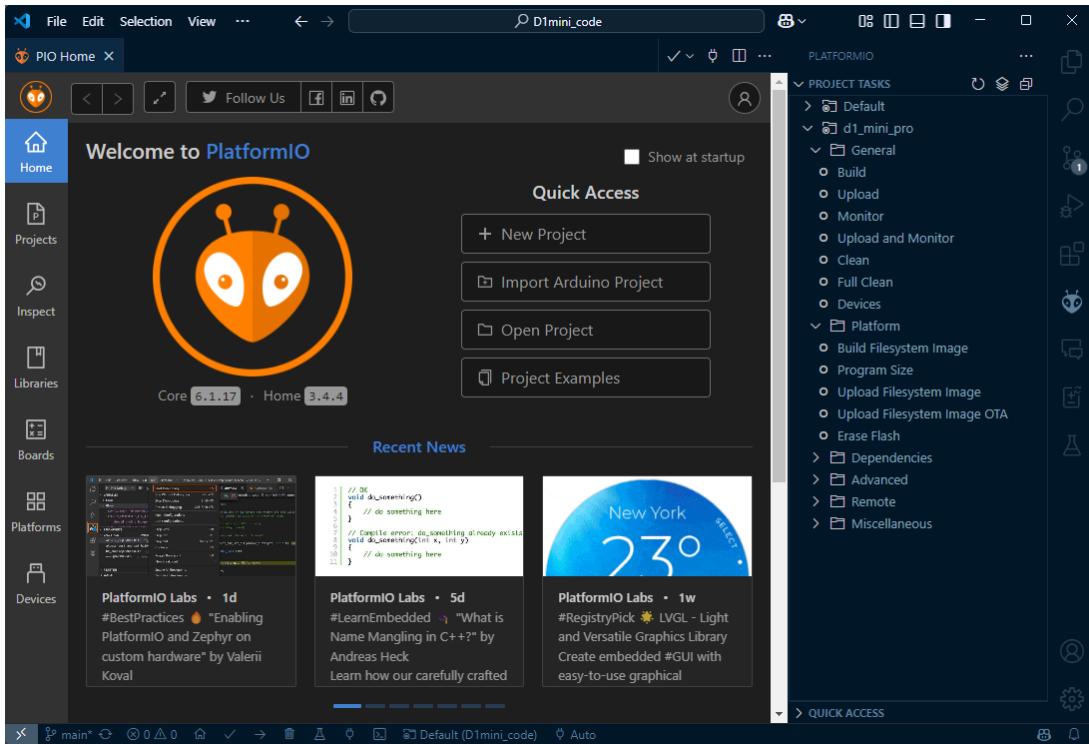
V této práci byl kód mikrokontroleru programován uvnitř programovacího prostředí PlatformIO (ukázané na obrázku 1.5) ve kterém byl Arduino framework pro ESP8266 využitý. Byla využitá i kompatibilita s existujícími Arduino knihovnami, jako v případě knihovny Ticker. To je knihovna, která v rámci Arduino zařízení umožňuje nastavit časovač, který spouští některou funkci pravidelně přesné časové intervaly.

Knihovny pro webové funkce

Využití Arduino frameworku pro ESP8266 plně umožňuje využít i WiFi funkcionality tohoto mikrokontroleru. Knihovny co jsou využité jsou knihovny **ESP8266WiFi**, **ESP8266WebServer** a **ESP8266mDNS**.

Knihovna **ESP8266WiFi** je základní stavební kámen veškeré síťové komunikace. Bez této knihovny by se mikrokontroler nemohl připojit k existující bezdrátové síti nebo si vytvořit vlastní hotspot. V kódu se implementuje pomocí příkazů jako je *WiFi.begin(jmeno,heslo)*.

Funkcionalitu web serveru, která je velmi důležitá pro tento projekt implementuje knihovna **ESP8266WebServer**. Tato knihovna poskytuje funkce pro definování plně

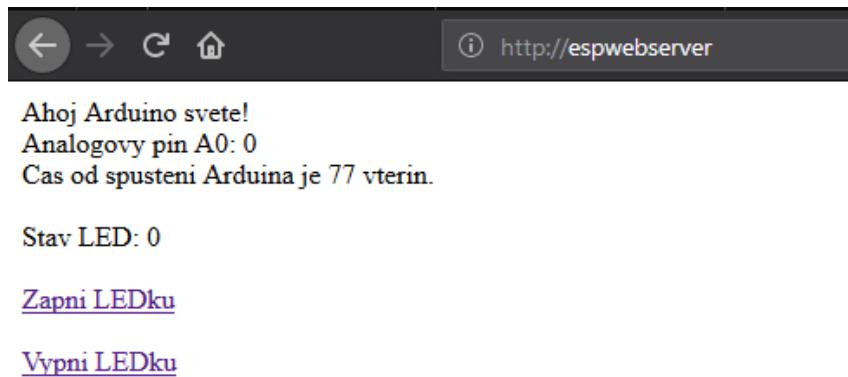


Obrázek 1.5: Ukázka rozhraní vývojového prostředí Platformio

funkčního HTTP serveru a odpovědi na jeho webové požadavky (např. GET pro získání dat, POST pro desílání dat a další). Tento webový server umí servírovat statické webové stránky, ale umí v rámci jejich provádění i spouštět jakékoli další funkce mikrokontroleru. Tohle umožňuje ovládat mikrokontroler skrz odpovědi na webové adresy.

Knihovna **ESP8266mDNS** neboli ESP8266 multicast DNS je knihovna, která umožňuje ukládat IP adresu mikrokontroleru na jakýchkoliv předem definovaných adresách která stačí zadat do prohlížeče na zařízení, které má server mikrokontroleru k dispozici.

Tyto tři knihovny se dohromady dají nastavit například tak, aby se pomocí jedné webové stránky dostupné na IP adresu *espwebserver* dala ovládala LED dioda na mikrokontroleru.



Obrázek 1.6: Ukázka nastavení WebServeru pro ovládání LED diody [10]

```
1 #include <ESP8266WiFi.h>
2 #include <ESP8266WebServer.h>
3 #include <ESP8266mDNS.h>
4
5 const char* nazevWifi = "Ardwifi";
6 const char* hesloWifi = "arduino1234";
7
8 ESP8266WebServer server(80);
9
10 #define LEDka LED_BUILTIN
11 #define analogPin A0
12
13 void zpravaHlavni() {
14     String analog = String(analogRead(analogPin));
15     String cas = String(millis() / 1000);
16     String ledStatus = digitalRead(LEDka) ? "ZAPNUTO" : "VYPNUTO";
17
18     String zprava =
19     "<h1>ESP8266 WebServer</h1>"
20     "Hodnota analogoveho pinu A0: " + analog + "<br>"
21     "Cas od spusteni: " + cas + " vterin.<br><br>"
22     "Stav LED (pin " + String(LEDka) + "): " + ledStatus + "<br><br>"
23     "<a href=\"/ledON\">Zapni LEDku</a><br>"
24     "<a href=\"/ledOFF\">Vypni LEDku</a>";
25
26     server.send(200, "text/html", zprava);
27 }
28
29 void setup() {
30     pinMode(LEDka, OUTPUT);
31     digitalWrite(LEDka, LOW);
32
33     WiFi.begin(nazevWifi, hesloWifi);
34
35     while (WiFi.status() != WL_CONNECTED) {
36         delay(50);
37     }
```

```

38
39     MDNS.begin("espwebserver");
40
41     server.on("/", zpravaHlavni);
42     server.on("/ledON", []() {
43         digitalWrite(LEDka, HIGH);
44         zpravaHlavni();
45     });
46     server.on("/ledOFF", []() {
47         digitalWrite(LEDka, LOW);
48         zpravaHlavni();
49     });
50
51     server.begin();
52 }
53
54 void loop() {
55     server.handleClient();
56     \delay(10)
57 }
```

Ukázka kódu 1.1: Nastavení ESP8266 WebServeru pro ovládání LED diody [10]

V ukázce kódu 1.1 lze z webových funkcionalit vidět hlavně globální objekt typu `ESP8266WebServer` s názvem `server`, pomocí kterého lze odpovídat na webové požadavky definované ve funkci `setup()`. Funkce `zpravaHlavni` je funkce kterou se odpovídá na většinu webových GET požadavků a tak je lépe definovaná a obsahuje dodatečné informace o stavu LED diody a času od spuštění. Tato funkce také rovnou obsahuje kód v jazyce HTML, ve kterém se píšou webové stránky.

Kód dále obsahuje připojení na WiFi pod zadaným názvem a heslem s tím že čeká na připojení a až poté bude provádět zbytek kódu. Zbytek `setup()` funkce obsahuje odpovědi na webové požadavky. Každá odpověď na webový požadavek odpoví tím, že zpět pošle HTML kód z funkce `zpravaHlavni` (který zobrazuje informace jako jsou v obrázku 1.6), ale v případě adres `/ledON` a `/ledOFF` stránka ještě nastaví vysokou nebo nízkou hodnotu na LED diodu na vývojové desce.

Nakonec je zde funkce `loop()`, která se každých 10 milisekund snaží odpovídat na webové požadavky. Na základě zadaných adres odpovídá prováděním bloků kódu které byly nastaveny v `setup()` funkci.

2 Návrh zařízení

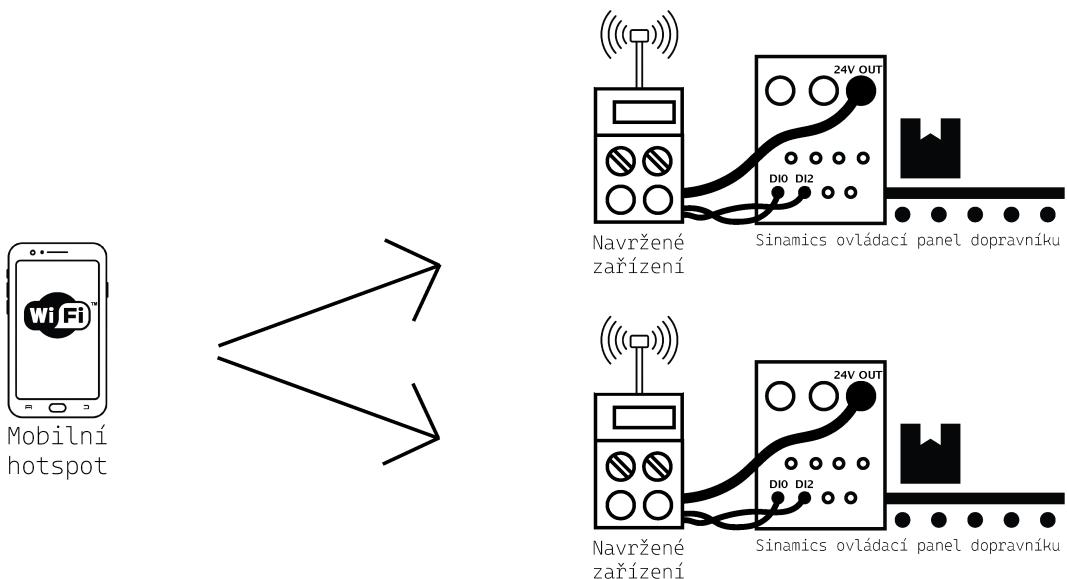
2.1 Princip funkce navrženého systému

Celý systém je primárně navržen kolem mobilní aplikace, jelikož požadavek na dálkové řízení dopravníků je jeden ze základních požadavků navrženého systému.

Aplikace bude obsahovat stránku pro plynulou WiFi komunikaci s vývojovou deskou WEMOS D1 Mini Pro. Vývojová deska je připevněna k desce plošných spojů, která je třemi kably připojena na ovládací panel (2 datové kably a 1 napájecí). Tímto způsobem může deska nastavovat takové digitální vstupy, k ovládání dopravníku.

Ovládací panel frekvenčního měniče si následně dle jeho nastavení podle své konfigurace interpretuje tyto příkazy a řídí výkonovou část frekvenčního měniče pro ovládání asynchronních motorů.

Na obrázku 2.1 je schéma základního principu.



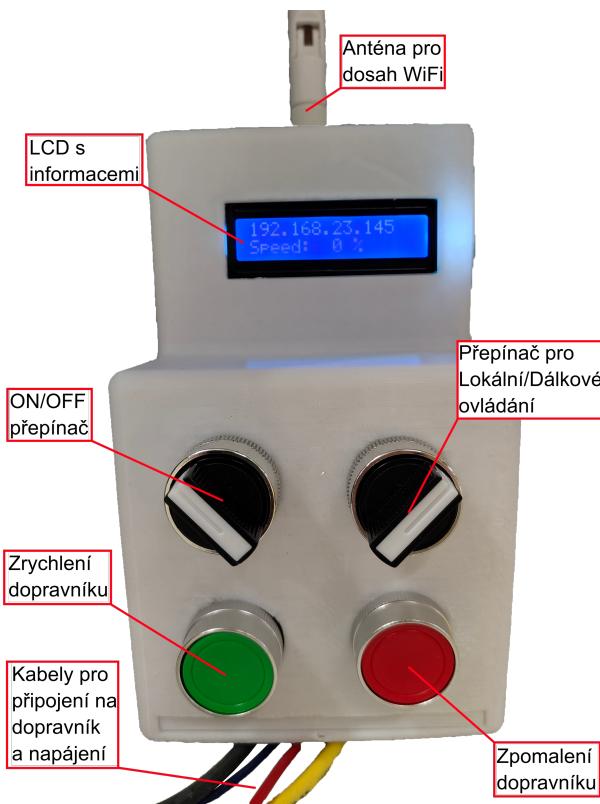
Obrázek 2.1: Schéma principu jak navržený systém funguje

Jelikož mobilní aplikace komunikuje s vývojovými deskami pomocí WiFi, je potřebné aby se buďto mobilní zařízení připojilo na přístupové místo vývojové desky, anebo se může vývojová deska připojit na hotspot mobilního zařízení. WebServer umožňuje obě varianty. Pro účely tohoto systému se více hodí ta druhá možnost, protože přirozeně umožňuje mít jeden hotspot na mobilním zařízení a na ten se může připojit více vývojových desek. Tohle umožňuje jednoduše ovládat více dopravníků zároveň. Další výhoda je, že vývojové desky připojené na hotspot vůbec nevyužívají toho, že je mobilní telefon připojený k internetu

a tak nijak nezatěžují rychlosť připojení - jediná limitace počtu takto připojených vývojových desek je tedy limitace maximálního počtu co může mít mobilní telefon připojené přes hotspot. Nevýhoda tohoto způsobu komunikace je ovšem taková, že se musí nastavit jednotné jméno a heslo WiFi komunikace, které bude zadáno přímo ve firmwaru vývojové desky a pokud bude potřebné tyhle údaje změnit, bude se muset přehrát kód všech vývojových desek (pro všechny pět používaných zařízení).

Díky tomu, že je tento systém navržený tak, aby přes 5-pinové kabely spínal digitální vstupy ovládacího panelu, je tento systém možné použít i na frekvenční měniče jiných značek než je Sinamics. Kabely, které se používají pro komunikaci s ovládacím panelem frekvenčního měniče (M12 5-pinové kabely) jsou v dnešní době u frekvenčních měničů časté. Jediné co je tedy potřeba pro používání systému s jiným frekvenčním měničem jsou správné konektory a dále aby bylo možné vyresetovat ovládací panel do podobného výchozího nastavení jako má Sinamics CU240-2.

Na obrázku 2.2 je finální vzhled schránky na desku plošných spojů s tlačítka, LCD displejem a dalšími funkcemi. Deska je dále popsána v kapitole 2.2.



Obrázek 2.2: Popis zařízení co ovládá dopravník

Na obrázku 2.3 je finální vzhled mobilní aplikace. Jsou zde vidět tři hlavní strany aplikace - Nastavení, Pomoc a Ovládání. Ovládání komunikuje s vývojovou deskou tím že posílá příkazy pro ovládání dopravníku, ale také získává průběrná data o rychlosti dopravníku a typu ovládání (lokální nebo dálkové). Aplikace je dále popsána v kapitole 2.4.

(a) Vstupní stránka aplikace (b) Část stránky nastavení (c) Část stránky s častými chybami

Obrázek 2.3: Vysílat příkazy zařízení bude mobilní aplikace připojená přes hotspot

2.1.1 Požadavky na systém

Pro zajištění funkčnosti a smysluplnosti návrhu je nutné si stanovit některé požadavky, které by systém měl splňovat. Tyto požadavky reflektují nejenom požadavky od společnosti Honeywell, ale i požadavky na základní spolehlivost, jednoduchost a bezpečnost ovládání dopravníků tímto způsobem.

Zde jsou požadavky, které by implementace navrženého zařízení měla splňovat:

- Lokální a dálkové ovládání**

Systém bude schopný ovládat dopravníky nejenom lokálně ale i bezdrátově.

- Ovládání více dopravníků zároveň**

Systém by měl jednoduše zprostředkovat ovládání více dopravníků zároveň.

- Ovládání z mobilního zařízení**

Aby se minimalizoval počet potřebných zařízení se systém musí dát provozovat z mobilního zařízení pomocí WiFi hotsporu.

- Napájení z ovládacího panelu**

Systém musí být navržený tak aby jeho rozšířené funkce bylo možné napájet připojením na 24V výstupní port v ovládacím panelu.

- Systém musí mít ovládání které je čistě analogové**

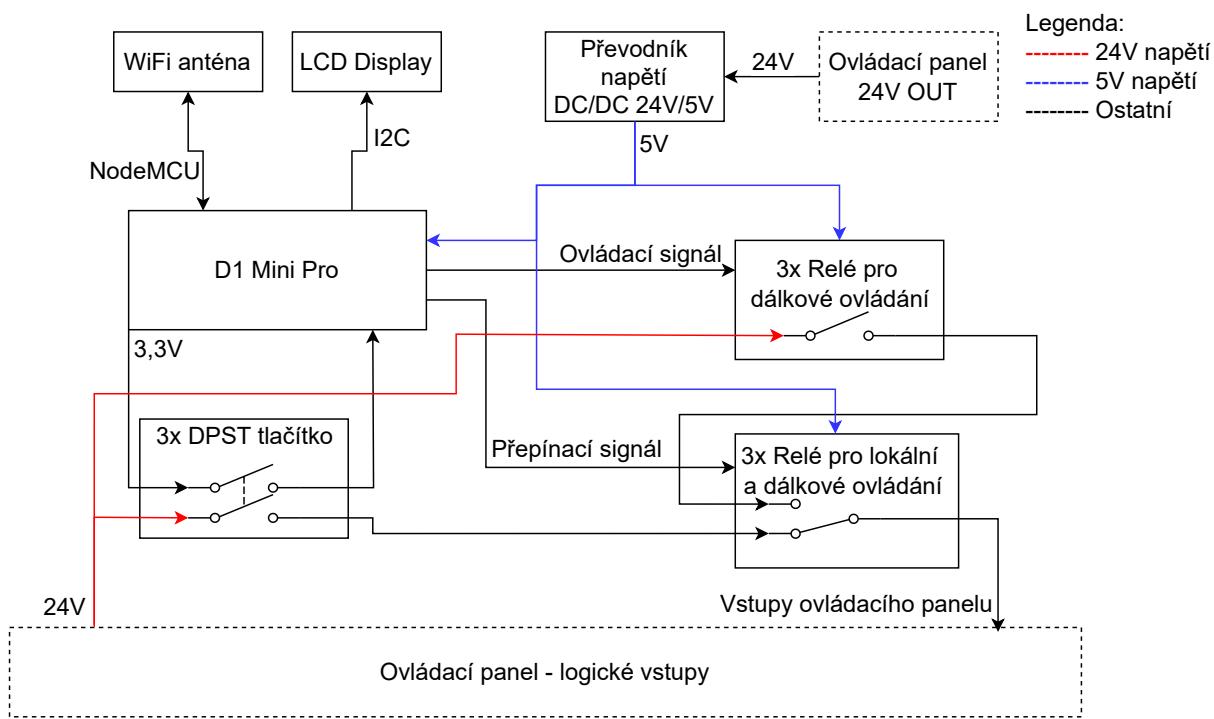
Systém by měl být navržen tak, aby bylo stále možné dopravníky ovládat i pokud by něco zamezovalo napájení vývojové desky.

- **Uživatelská přívětivost systému**

Systém by měl být uživatelsky přívětivý a jeho nastavení by mělo být jednoduše dostupné pro uživatele spolu se všemi informacemi jak s ním pracovat.

2.2 Hardware

Základní stavební kámen celého systému je jeho hardware společně s deskou plošných spojů (DPS) ve které je umístěn. Celé zapojení bylo nejdříve navržené jako elektrické schéma. To bylo postupně vylepšováno, později předěláno do schématu v programu pro tvorbu DPS a nakonec bylo vytvořené rozložení komponentů přímo na desce. Pozdější verze návrhu byly provedeny v počítačovém programu na návrh desek plošných spojů jménem KiCAD. Blokové schéma funkčnosti desky lze vidět na obrázku 2.4.



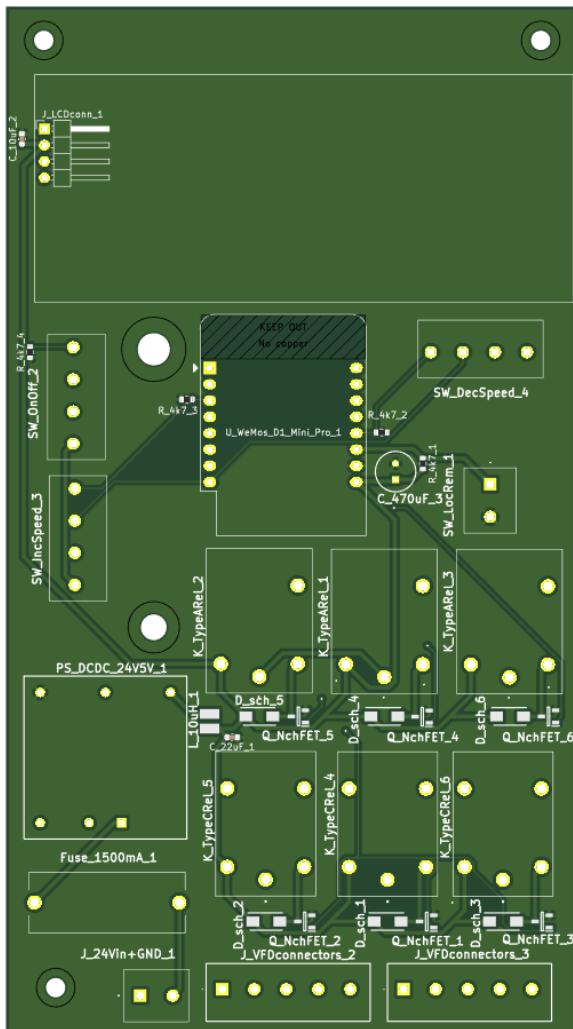
Obrázek 2.4: Blokové schéma desky plošných spojů

Při tvorbě desky se vycházelo z požadavků na systém. Nejdřív se vycházelo z toho, že desku musí být možné napájet z ovládacího panelu, který má výstupní port na kterém je 24V a který je schopný dodat maximálně 8A. Ze začátku se tedy počítalo s 24V napětím, pro které bylo potřeba zvolit dobrý převodník napětí co je schopný napětí přeměnit na 5V kterým se napájí vývojová deska. Nakonec byl zvolený převodník napětí s galvanicky oddělenou zemí aby se minimalizovala šance, že by kvůli nějaké chybě v návrhu desky byl zničený ovládací panel frekvenčního měniče. [4]

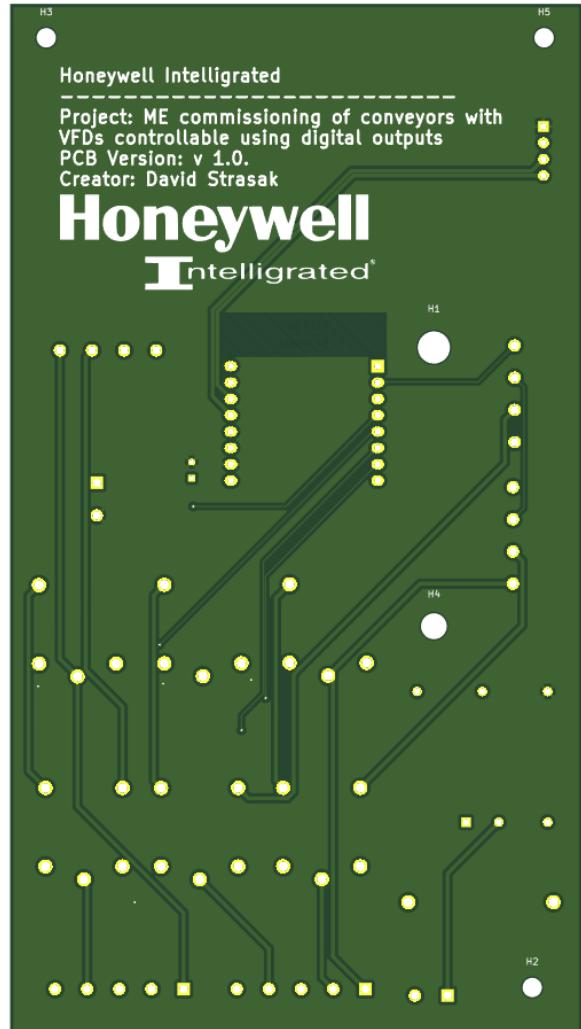
Dále se při návrhu vycházelo z myšlenky že systém musí mít i lokální i dálkové ovládání, s tím, že lokální ovládání musí být dostupné i bez napájení mikrokontroleru. Tohle bylo vyřešeno návrhem dvou přepínačů. *Relé pro lokální a dálkové ovládání* je přepínač, který je normálně v poloze lokálního ovládání (aby byl splněný požadavek čistě analogového ovládání), ale pokud se na něj přidá napětí, přepne se do stavu dálkového ovládání. Následně je v desce *Relé pro dálkové ovládání*, což je běžné SPST relé které sepne kontakty pokud je na něj přivedeno napětí.

Po dokončení schématu se přešlo na návrh umístění součástek na DPS. Finální návrh lze vidět v obrázku 2.5. Při rozmístování součástek po desce byl kladen důraz na několik kritérií:

- Aby měla deska co nejmenší rozměry a měla jen dvě vrstvy
- Aby byly všechny součástky na jedné straně desky (jednodušší pájení komponentů na desku)
- Aby byly trasy co nejkratší
- Aby měla deska co nejméně vertikálních cest (pro zmenšení efektů parazitní kapacity a parazitní indukčnosti)
- Aby byly filtrační kondenzátory co nejbliž filterovaných napájecích vstupů



(a) Přední strana DPS



(b) Zadní strana DPS

Obrázek 2.5: Návrh desky plošných spojů v KiCAD

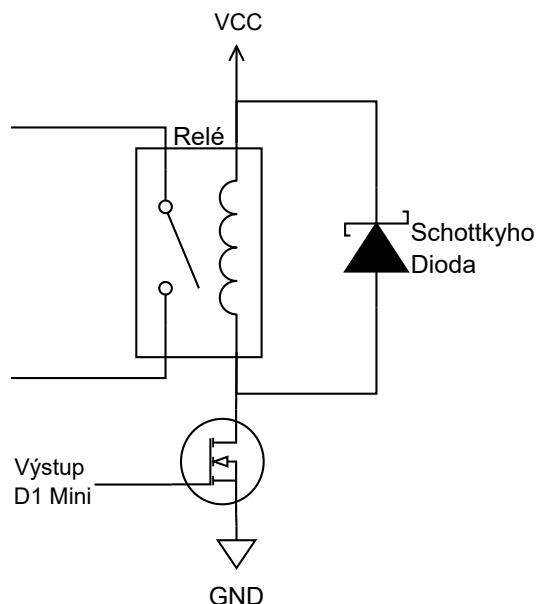
2.2.1 Ovládání relé

Jedna z věcí které bylo potřeba vyřešit při návrhu schématu elektrického obvodu je taková, že bylo cílem ovládat relé pomocí výstupních pinů vývojové desky. Pro spuštění relé je ovšem potřeba brát na paměť, že vyžaduje nejenom napětí $5V$ na ovládacím pinu, ale také vyžaduje proud do cívky $133mA$. Vývojová deska je schopná svými výstupními piny poskytnout napětí $5V$, ale její výstupní proud je pouze $10mA$. Tento problém je možné vyřešit pomocí tranzistorů.

Nejjednodušší řešení pro navrženou desku je řešení s N kanálovým NPN tranzistorem MOSFET, jelikož v případě MOSFET tranzistorů obecně stačí aby signál byl napěťový (nemusíme tedy řešit malý proud vycházející z vývojové desky). Nejdřív se relé zapojí tak, aby cívka měla na jedné straně $5V$ napětí a na druhé straně zem. Následně se MOSFET tranzistor dá mezi zem a cívku a na bázi tranzistoru se přivede signál z mikrokontroleru. V tomto zapojení bude skrz relé téct nominální proud, pokud je výstup vývojové desky vysoký, anebo bude obvod rozpojený pokud bude výstup z vývojové desky na nízké hodnotě napětí.

Dále je potřeba ještě paralelně s relé zapojit diodu kvůli ochraně tranzistoru od vybijecího proudu co cívka generuje po rozpojení obvodu. Tato dioda musí mít v závěrném směru hodnotu napětí vyšší než je $5V$ napětí zdroje a musí se zapojit tak, aby byla otevřená když se otočí polarita napětí na cívce v relé. Pro tuto desku byla zvolena Schottkyho dioda kvůli její rychlosti přepínání a kvůli malému napětí které se na diodě v otevřeném stavu nachází.

Na obrázku 2.6 je schéma zapojení tohoto způsobu ovládání relé pomocí vývojové desky.



Obrázek 2.6: Elektrické schéma ovládání relé

2.2.2 LCD display s I2C převodníkem

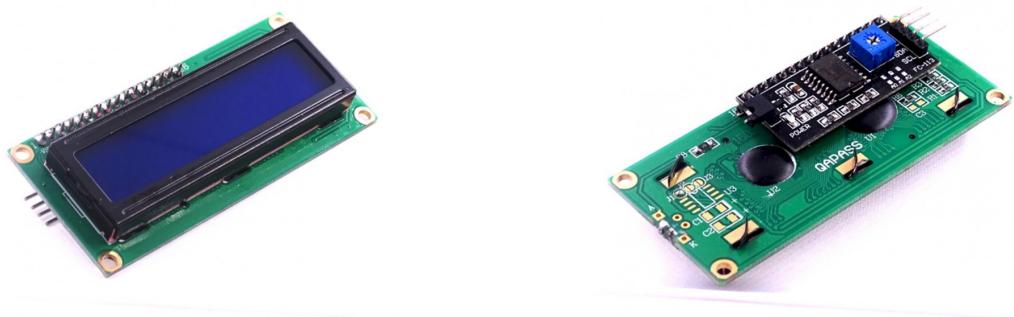
Aby bylo možné celý systém ovládat přes ESP8266 WebServer, je potřebné nějakým způsobem komunikovat s uživatelem IP adresu, kterou má vývojová deska připojená na hotspot mobilního zařízení. Tohle by bylo možné udělat například skrz nastavení multicast DNS na specifickou adresu a tu potom fyzicky napsat na schránku desky. Toto řešení by sice umožnilo přístup k WebServeru, ale neposkytovalo by to další funkce jako systém může mít se zabudovaným LCD displejem do desky plošných spojů.

Použití LCD displeje v systému může uživateli dodávat tyto informace:

- Stav připojení mikrokontroleru k hotspotu (před navázáním spojení systém nemůže reagovat na požadavky přes WebServer).
- Název (SSID) a heslo hotspotu, které mikrokontroler očekává.
- Aktuální rychlosť dopravníku.
- IP adresu pro přístup k WebServeru.

Na základě uvedených důvodů a požadavků na rozsah poskytovaných informací byl pro komunikaci s uživatelem zvolen LCD displej.

Konkrétně byl použit 16x2 znakový LCD displej (obrázek 2.7) zakoupený v internetovém obchodě LaskaKit [11]. Tento model je vybaven připájeným I2C převodníkem, což zjednodušuje jeho připojení na pouhé čtyři vodiče: dva pro I2C sběrnici (Serial Data (SDA) a Serial Clock (SCL)), jeden pro napájení 5V a jeden zemnící vodič (GND). Výhodou zvoleného displeje je také dostupnost knihovny pro Arduino framework, což usnadňuje jeho softwarovou implementaci ve firmwaru mikrokontroleru. [11]



(a) Přední strana

(b) Zadní strana

Obrázek 2.7: LCD displej s I2C převodníkem [11]

Vzhledem k citlivosti komunikace po I2C sběrnici na elektromagnetické rušení a přeslechy byly při návrhu desky plošných spojů dodrženy zásady pro minimalizaci smyčkové plochy mezi signálovými cestami SDA a SCL. Tento postup snižuje indukované napětí a přispívá ke spolehlivosti přenosu dat na sběrnici. [12]

Pro jednoduchost výměny LCD displeje bylo také zvoleno, že nebude přímo připájený v desce, ale podobně jako vývojová deska bude připojený skrz kolíkové lišty. Tohle zjednoduší výměnu nebo upravení parametrů LCD displeje jako je jas zobrazených písmen.

2.3 Firmware ve vývojové desce

Jak bylo již napsáno v kapitole 1.2.3 tento projekt byl programovaný v prostředí PlatformIO. Tohle programovací prostředí má tu výhodu v tom, že pro programování vývojových desek si stačí vybrat typ vývojové desky, kterou programuji a dané prostředí se nakonfiguruje tak, aby to bylo možné (V případě WEMOS D1 Mini Pro byl implementován Arduino framework pro ESP8266). Z tohoto prostředí tedy lze programovat jakékoli vývojové desky a programovat je jakýmkoliv jazykem mezi které patří například Arduino jazyk anebo MicroPython. Pro firmware v tomto projektu byl zvolený Arduino framework a tak je programovací jazyk Arduino verze C++. [13]

Honeywell je anglicky mluvící firma a tak je celý firmware i software v mobilní aplikaci programovaný anglicky.

V C++ programovacím jazyku se běžně objevují dva hlavní typy souborů. Jsou to hlavičkové soubory (s koncovkou `.h`) a zdrojové soubory (s koncovkou `.cpp`). Zatímco hlavičkové soubory obsahují deklarace, jako jsou prototypy funkcí nebo definice globálních proměnných, zdrojové soubory obsahují implementace a kód který je následně kompilován a prováděn na mikrokontroleru. V tomto projektu jsou dva hlavičkové a dva zdrojové soubory.

Jeden hlavičkový soubor obsahuje definice alternativních názvů pro výstupních a vstupních piny GPIO vývojové desky aby bylo jednodušší se orientovat v kódu. Dále je zde soubor **main.cpp**, který kompilátor přirozeně vyhledává aby v něm nastavil začátek provádění kódu. Nakonec je zde zdrojový a hlavičkový soubor s názvem **ConveyorController** který definuje třídu a zdrojový kód objektu který celý systém řídí.

2.3.1 Třída a objekt ConveyorController

C++ je objektově orientovaný program a tak má rozsáhlé funkce pro definici vlastních tříd. Kvůli tomu je důležité rozlišovat mezi pojmy třída a objekt. Třída v C++ slouží jako abstraktní definice pro vytvoření nového uživatelsky definovaného datového typu. Objekt je na druhou stranu konkrétní instance třídy a tak má alokované místo v RAM paměti a je sledovaný jeho unikátní stav během provádění kódu. [14]

Hlavní výhodou proč byl v práci použity objektově orientovaný přístup je kvůli zavedení takzvané **enkapsulace** uvnitř kódu. Jinými slovy, umožnuje to separaci řídící logiky dopravníku od zbytku aplikačního kódu. Během implementace firmwaru vyvstala potřeba, aby určité proměnné, jako například proměnná *conveyorSpeed* reprezentující approximaci rychlosti dopravníku či proměnná *remoteLocalState* indikující stav řízení (vzdálené/lokální), byly přístupné napříč několika funkcemi. Běžně by bylo možné tyto proměnné řešit pomocí globálních proměnných, ale to je v komplexnějších firmwarech považováno za rizikové. [15]

V takovém případě se pro bezpečnější provádění kódu může přejít k využívání objektů. Samotný objekt je sice definovaný globálně, ale uvnitř má tři definice přístupu: **veřejný**, **privátní** a **chráněný**. Tímto způsobem se může pro jakékoli proměnné uvnitř objektu určit, v jakých částech kódu budou dostupné, s tím, že pokud je proměnná definovaná jako veřejná, dá se získat i mimo daný objekt a na druhou stranu pokud je proměnná privátní, je možné ji získat pouze uvnitř metod objektu. Metoda je taková funkce, kterou je možné uvnitř objektu se stejnými přístupy definovat a je to kus kódu co bude provedený pokud bude metoda zavolaná. Metody které jsou veřejné je tedy možné spouštět ze skriptu *main.cpp* ve kterém existuje globální instance objektu, ale privátní metody už není možné

z hlavního skriptu spouštět.

Kvůli této funkcionality bylo od začátku rozhodnuto, že bude kód programovaný tímto způsobem. Většina kódu bude obsažena v conveyorController objektu a v hlavním zdrojovém kódu budou pouze volány veřejné metody této globální instance třídy ConveyorController.

Aby bylo možné představit jaké funkce jsou v ConveyorController třídě obsaženy, zde je její definice v hlavičkovém souboru *ConveyorController.h*:

```
1 #ifndef CONVEYORCONTROLLER_H
2 #define CONVEYORCONTROLLER_H
3
4 #include <Arduino.h>
5 #include <ESP8266WebServer.h>
6 #include <ESP8266WiFi.h>
7 #include <ESP8266mDNS.h>
8 #include <LiquidCrystal_I2C.h>
9 #include <WiFiClient.h>
10 #include "pinDefinitions.h"
11 #include <Ticker.h>
12
13 class ConveyorController {
14 public:
15     ConveyorController(const char* wifiNetworkName,
16                         const char* wifiNetworkPassword);
17
18     void initIO();
19     void initLCD();
20     void initWeb();
21     void assignRoutes();
22     void startWebServer();
23     void startTicker();
24     void handleClient();
25     void updateLCD();
26     void updateState();
27
28 private:
29     // WiFi credentials
30     const char* wifiNetworkName;
31     const char* wifiNetworkPassword;
```

```
32
33 // Web server
34 ESP8266WebServer webServer = ESP8266WebServer(80);
35
36 // LCD
37 LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27, 16, 2);
38
39 // Ticker
40 Ticker inputCheckingTicker;
41 Ticker LCDUpdatingTicker;
42
43 // Speed of the conveyor
44 int conveyorSpeed = 0;
45
46 // TRUE or FALSE state if the conveyor is controlled locally or remotely
47 // remoteLocalState ? "local" : "remote"
48 bool remoteLocalState = false;
49
50 // TRUE or FALSE state if the conveyor is speeding up or no
51 bool locIncSpeedState = false;
52 bool remIncSpeedState = false;
53
54 // TRUE or FALSE state if the conveyor is slowing down or no
55 bool locDecSpeedState = false;
56 bool remDecSpeedState = false;
57
58 // TRUE or FALSE state if the conveyor is ON or OFF
59 bool locOnOffState = false;
60 bool remOnOffState = false;
61
62 // Route handler for the main page
63 void mainRoute();
64
65 // Route handler for unknown pages
66 void unknownRouteResponse();
67
68 void LCDWaitingForConnection(bool condition);
69
```

```

70     void writeValue(int pin, int value);
71 };
72
73 #endif

```

Ukázka kódu 2.1: Header soubor ConveyorController Objektu

Z této definice lze vidět, že pro přehlednost je celkový kód rozdělený do různých metod tak, aby měla každá metoda svůj účel. Tyto metody jsou seřazeny podle pořadí provedení až do funkce startTicker, po které se už funkce provádějí periodicky. Účely které zajišťují metody jsou:

- **initIO** - Nastavuje piny vývojové desky na vstupy a výstupy pomocí PinMode příkazu a dále inicializuje výstupní piny na nízkou hodnotu.
- **initLCD** - Inicializuje používání LCD displeje.
- **initWeb** - Začne hledání webové sítě, která by odpovídala parametrům nastaveného jména (SSID) a hesla. Také informuje o hledání sítě na LCD displeji a přes serial komunikaci, kterou se hodí mít při debugování firmware.
- **assignRoutes** - Přiřadí WebServeru odpovědi na jednotlivé adresy. Tyto odpovědi obsahují nejdříve nastavení *<head>* části odpovědi (dále vysvětlené v kapitole 2.4.6), poté provádění C++ kódu a nakonec odesílání odpovědí na požadavky.
- **startWebServer** - Spustí WebServer, který bude nyní přijímat požadavky.
- **startTicker** - Nastaví Ticker (časovač) pro periodické spouštění hlavní funkce **updateState** a funkce pro aktualizaci LCD.
- **handleClient** - Sleduje webové požadavky.
- **updateLCD** - Aktualizuje na LCD displeji hodnotu IP adresy a hodnotu rychlosti dopravníku.
- **updateState** - Tato funkce řídí relé na desce plošných spojů a tak i ovládá celý dopravník. Chová se dle stavového diagramu, který je popsaný v kapitole 2.3.2.

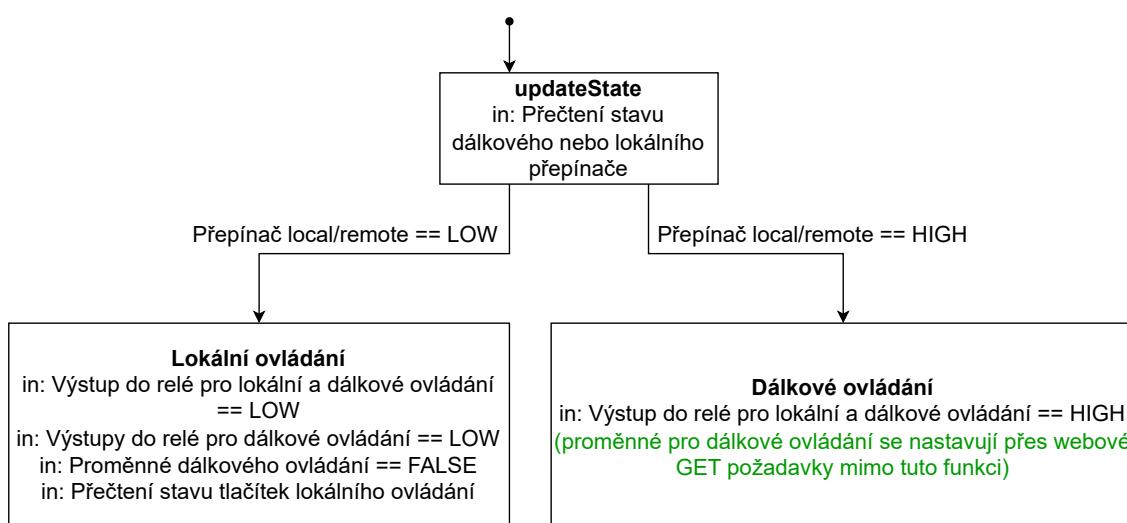
Hlavíčkový soubor dále obsahuje i několik prototypů privátních metod které urychlují psaní kódu při opakovaných úkonech a proměnných které jsou používány uvnitř metod (všechny proměnné používané uvnitř třídy jsou privátní). Specifické privátní metody které je potřeba poukázat jsou metody **mainRoute** a **unknownRouteReponse**. Tyhle metody zahrnují vytváření HTML řetězce kterým se odpovídá na webové požadavky, co přichází na WebServer. Je to podobný způsob jak odpovídat na webové požadavky jako byl v ukázce kódu 1.1.

2.3.2 Stavový diagram logiky systému

Aby bylo možné řídit dopravník pomocí ConveyorController objektu, je potřebné implementovat v kódu logiku, která sleduje stav ovládání dopravníku a podle stavu ovládání bude spínat tranzistory co spouští relé v desce plošných spojů. Do téhle funkcionality se navíc dalo přidat i některé další funkce jako je approximace rychlosti dopravníku blíže popsaná v kapitole 2.3.2. Tato veřejná metoda objektu ConveyorController se jmenuje updateState a je pomocí knihovny Ticker prováděna každých 300 milisekund.

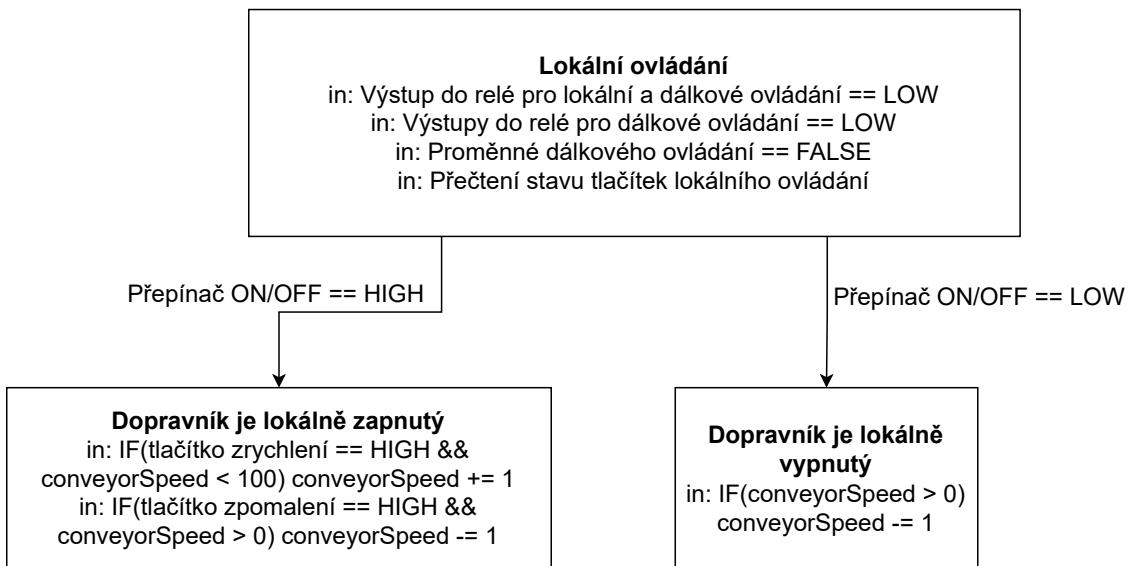
V tomhle systému je na tuto část kódu nahlíženo jako na stavový diagram Harelova typu. Pro přehlednost je zde stavový diagram rozdělený do tří různých částí (byl ale navrhován jako jeden celek):

- **Začátek stavového diagramu** - kde se rozhoduje hlavně jestli je dopravník řízený lokálně nebo dálkově.
- **Dálkové ovládání dopravníku**
- **Lokální ovládání dopravníku**



Obrázek 2.8: Začátek stavového diagramu

Zde je vstup do stavového diagramu funkce updateState. Na začátku se přečte hodnota vstupního pinu vývojové desky ke kterému je připojený přepínač pro lokální nebo dálkové ovládání. Na základě této hodnoty se určí, jestli je systém ve stavu lokálního nebo dálkového ovládání. Podle této hodnoty se stavový diagram větví do diagramů pro lokální nebo dálkové ovládání.



Obrázek 2.9: Strana stavového diagramu s lokálním ovládáním

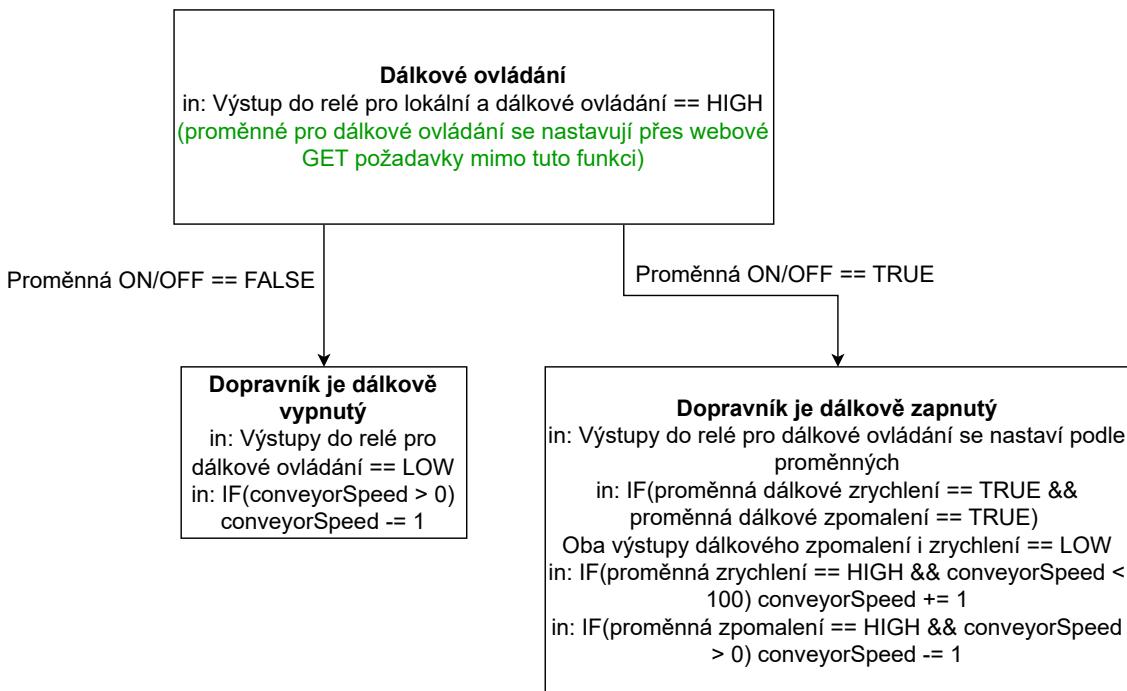
Ve stavu s lokálním ovládáním se nastaví výstupní pin vývojové desky ovládající tranzistor, který vypne relé, které přepíná mezi lokálním a dálkovým ovládáním. Následně se rovněž nastaví tranzistor ovládající dálkové ovládání, jelikož není potřeba toto relé mít zapnuté, když je tato větev obvodu deaktivována. Nakonec se nastaví proměnné dálkového ovládání na FALSE, aby byly vynulovány nastavené příkazy z WebServeru. Na konci provedení těchto bloků kódu se přečtem stavu tlačítek a přepínačů umístěných fyzicky na desce a na základě těchto stavů se pokračuje v provádění kódu.

V tomto stavu působí mikrokontroler spíše jako pouhý pozorovatel stavu tlačítek, než aby přímo spínal nějaké z relé. V blokovém schématu systému na obrázku 2.4 lze vidět, že tomu tak je protože je relé pro přepínání lokálního nebo dálkového ovládání připojené přímo k tlačítku, které bez žádného digitálního zpracování spíná nebo rozepíná 24V linku napětí pro ovládací panel. Toto vychází z požadavku na systém **Systém musí mít ovládání které je čistě analogové**.

Ze začátku se může pokračovat do větve **Dopravník je lokálně vypnuty**, a to v případě, že je hodnota přepínače, který lokálně ovládá stav ON/OFF, vyhodnocena LOW. V tomto stavu se pouze odečte hodnota rychlosti dopravníku, pokud je rychlosť vyšší než 0¹.

Alternativně je možné pokračovat do větve **Dopravník je lokálně zapnutý**, pokud je hodnota přepínače ON/OFF vyhodnocena jako HIGH. V tomto případě se sleduje, jakým způsobem je dopravník ovládán, a na základě toho se přičte nebo odečte hodnota rychlosti dopravníku.

¹rychlosť dopravníku sledovaná v proměnné conveyorSpeed je pouze approximace, která se zobrazuje v mobilní aplikaci a na LCD displeji. Není to vstup do ovládacího panelu.



Obrázek 2.10: Strana stavového diagramu s dálkovým ovládáním

Ve stavu s dálkovým ovládáním se nastaví výstupní pin, který ovládá relé pro dálkové nebo lokální ovládání, na úroveň HIGH. V rámci tohoto stavu už není nic dalšího provedeno, jelikož se proměnné, na základě kterých se ve stavovém diagramu pokračuje, nastaví při zpracování GET požadavků, které přicházejí na WebServer.

WebServer tedy nastavuje proměnné, na základě kterých se v tomto stavovém diagramu pokračuje v provádění kódu. Tyto proměnné jsou inicializovány na hodnotu FALSE, a proto se po přepnutí do režimu dálkového ovládání dopravník vypne. Během provádění stavového diagramu dálkového ovládání nejsou tyto hodnoty vynulovány, a je proto nutné přes WebServer nastavit proměnné jak na hodnotu TRUE, tak na hodnotu FALSE.

Pokud přes WebServer bude proměnná ON/OFF nastavena na hodnotu FALSE, poté se vypne relé, které ovládá dálkové ovládání, a sníží se hodnota rychlosti dopravníku až na 0.

Pokud se přes WebServer proměnná ON/OFF nastaví na hodnotu TRUE, sepne se tranzistor, který přivede proud do relé pro dálkové ovládání. Podobně se spustí relé, které zrychluje nebo zpomaluje dopravník přes ovládací panel, pokud jsou jejich specifické proměnné nastavené na hodnotu TRUE. Také zde se zvyšuje nebo snižuje hodnota approximace rychlosti.

Jak aplikace approximuje rychlosť dopravníku

V předchozí kapitole byla zmíněna proměnná *conveyorSpeed* která si udržuje approximovanou hodnotu rychlosti dopravníku. Approximace rychlosti vychází z předpokladu, že frekvenční měnič zrychluje asynchronní motory dopravníku lineárně. Tento předpoklad je smysluplný, jelikož bývají frekvenční měniče v praxi často konfigurovány s lineárními rampami pro zrychlení a zpomalení ovládaných motorů. Tento předpoklad byl ověřen přímo

na frekvenčním měniči v hale společnosti Honeywell a bylo tedy zkонтrolováno, že platí.

Při testování bylo také zjištěno, že frekvenční měnič zrychluje o 500 otáček za 10 sekund. Zrychlení frekvenčního měniče je tedy $50\text{ot}/\text{min}$ za sekundu, což je z maximální hodnoty $1500\text{ot}/\text{min}$ (definovaná při nastavení ovládacího panelu v kapitole 1.1.3) zrychlení $3.33\%/\text{s}$ neboli 1% za 300ms . Dopravník stejnou rychlosť i zrychluje i zpomaluje.

Samotná approximace rychlosti dopravníků je tedy provedena tím způsobem, že se veřejná metoda `updateState` provádí každých 300ms a během každého provedení končí každá větev stavového diagramu aktualizací hodnoty `conveyorSpeed`. Tato aktualizace je vždy provedena tím způsobem, aby hodnota patřila do intervalu $<0, 100>\%$ z maximální rychlosti.

Volat funkci `updateState` každých 300ms by mělo být pro approximaci dostatečně pravidelné. Jediný způsob jak by approximace mohla nabývat špatných hodnot je pokud by byl systém ovládaný v lokálním režimu ovládání, tlačítka zrychlení by bylo sepnuté po dobu těsně menší než 300ms a zrovna tato doba vyšla na dobu mezi volání funkce `updateState`. Tohle je obecně chyba která může nastat u vzorkování. Pokud by se tak stalo i opakovaně, stejně by byla chyba v rádu jednotek procent, což je zanedbatelné.

Naopak častější volání funkce `updateState` než 300ms by mělo vyšší spotřebu energie a bralo by to zbytečně výpočetní výkon vývojové desky, která ho spíše může využít pro odpovídání na webové požadavky co přichází na WebServer.

Způsob jakým je funkce `updateState` spuštěna každých 300ms je pomocí nastavení časovače skrz framework jménem `Ticker`. `Ticker` je open-source knihovna, která může být vložena do jakéhokoliv projektu založeném na Arduino framework. Zde je příklad kódu pro nastavení `Ticker` knihovny:

```

1 #include "Ticker.h"
2
3 void blink() {
4     digitalWrite(LED_BUILTIN, ledState);
5     ledState = !ledState;
6 }
7
8 void setup() {
9     Ticker timer(blink, 500);
10    timer.start();
11 }
12
13 void loop() {
14     timer.update();
15 }
```

Ukázka kódu 2.2: Způsob používání `Ticker` knihovny

Používání `Ticker` knihovny tedy spočívá v tom, že si stačí zaobalit kód, který má být

vykonáván, do samostatné callback funkce. Následně je možné vytvořit **Ticker** objekt ve kterém jsou vstupy tato callback funkce a čas který určuje jak často má být kód vykonáván. Pro kontrolu se doporučuje ještě přidat **update** metodu vytvořeného **Ticker** objektu do **loop** funkce.

Díky možnosti nastavit pomocí knihovny **Ticker** metodu **updateState** vzniká dobrý způsob jak dávat uživatelům systému vědět, jakou má dopravník aktuální rychlosť i bez toho, aby bylo připojené zařízení Sinamics IOP-2 kterým se ovládací panel programuje (popsáno v kapitole 1.1.3).

2.4 Software v mobilní aplikaci ($\Sigma = 5$ stran)

Jak už bylo nastíněno v sekci 2.1, primární důvod proč bylo rozhodnuto, že bude systém ovládaný přes mobilní aplikaci je, že není potřeba mít další dedikované zařízení, které bude fungovat jako dedikovaný hardwareový ovladač systému. Díky tomu je celkový systém přenositelnější a jednodušší ovládat, jelikož mobilní zařízení musí uživatelé mít už i kvůli zapisování dat z kontroly dopravníků. Komunikace mobilní aplikace a vývojové desky je realizována pomocí bezdrátové technologie WiFi, což umožňuje komunikaci v řádech desítek metrů (u nejnovějšího WiFi protokolu do 90 metrů). [16]

Vzhledem k požadavku na systém **Uživatelská přívětivost systému** je navíc velmi výhodné mít systém ovládaný mobilní aplikací, jelikož je možné mít prakticky veškeré informace ohledně nastavování ovládacího panelu frekvenčního měniče na jednom místě. Tohle výrazně zlepšuje uživatelské používání systému, protože uživatelé nemusí hledat dokumentaci, když je na stejném místě jako ovládací prvky systému.

Mobilní aplikace není technicky vzato pro systém potřebná. WebServer je možné ovládat i přes webový prohlížeč. Poté by ale systém ztrácel další funkce, které mobilní aplikace nabízí. To je třeba zmínovaná dokumentace uvnitř aplikace, nebo možnost mít automaticky aktualizovanou hodnotu rychlosti dopravníku a stavu ovládání. Velká výhoda aplikace je také ovládání více dopravníků najednou.

2.4.1 Architektura aplikace

Navržená mobilní aplikace vůbec není mobilní aplikace v tradičním slova smyslu, ale spíše jde o webovou aplikaci. Tuto webovou aplikaci lze pomocí některých knihoven a rozšíření spolehlivě konvertovat do mobilní aplikace. Zde budou popsány důvody tohoto rozhodnutí.

Běžná webová aplikace má dvě hlavní části:

První část se jmenuje **front-end** a je to uživatelská část aplikace - obsahuje veškerý kód, který je prováděn u uživatele v prohlížeči. To většinou bývají jednodušší JavaScriptové bloky kódu, veškerá struktura a obsah aplikace (psaná v jazyce HTML) a veškeré formátování aplikace (běžné psané v jazyce CSS). Všechno ve front-end části jakékoli webové aplikace je možné si dohledat pomocí rozhraní jako jsou **Chrome Developer Tools**, které kterémukoliv uživateli ukážou celý front-end kód webové aplikace.

Druhá hlavní část webové aplikace se jmenuje **back-end** a je to část aplikace kterou běžný uživatel nevidí. Obsahuje data seřazená v databázích, které je možné si pomocí různých příkazů načíst do front-end části. Také umí využívat server pro provádění různých výpočtů, které jsou buďto moc náročné, anebo musí být z určitých důvodů schované před běžnými uživateli.

Z hlediska softwarové architektury byla aplikace koncipována jako statická webová

aplikace - tedy čistě front-endová aplikace bez back-end části. Aplikace byla postavená na knihovně jazyka JavaScript jménem **React**, což je dlouhodobě jedno z nejpopulárnějších rozhraní pro tvorbu webových aplikací (jak tvrdí studie nejpoužívanějších jazyků pro webové programování [17]). Specificky byla použita **Vite** verze Reactu, která je optimalizována pro rychlosť aplikace. [18]

Aby bylo možné programovat webové aplikace, je potřeba mít nainstalovanou aplikaci **Node.js**, která mimo jiné usnadňuje programování webové aplikace tím, že umí v reálném čase ukazovat jak každá změna React kódu ovlivnila aplikaci pomocí takzvaného "development"serveru co je dostupný na lokální síti počítače, na kterém je node.js spuštěný.

Každá verze webové aplikace byla testována tím způsobem, že se překonvertovala do statických souborů (obsahující pouze HTML, CSS a JS soubory) pomocí příkazu **vite build**. Ten přeloží syntax jazyka React do souborů, které jsou ekvivalentní, ale narozdíl od React syntaxu jim rozumí každý webový prohlížeč.

Když existuje tato "čistá" verze aplikace, je možné použít framework jménem **Capacitor**, který je vytvořený specificky pro to aby bylo možné ho pomocí node.js nainstalovat do knihoven pro webové programování jako je React. Následně je možné pomocí několika příkazů Capacitor frameworku přeložit tuto verzi aplikace do souborů mobilní aplikace. [19]

Nakonec je možné si tuto složku konvertované mobilní aplikace otevřít v prostředí **Android Studio** a tam si z daného projektu vytvořit soubor o příponě **.apk**, který si umí nainstalovat každé android zařízení. Pomocí capacitor je možné konvertovat i do mobilní aplikace pro zařízení firmy Apple. Na konvertování do **.apk** souboru je ale potřebné mít i stolní počítač značky Apple, kde je možné aplikaci konvertovat pomocí **Xcode**.

Proč webová aplikace?

Důvod proč byla webová aplikace programována jako webová a ne přímo mobilní je jelikož hlavní účel dané aplikace je posílat GET požadavky na IP adresu vývojové desky. To je úkol, na který je jakákoli knihovna pro tvoření webových aplikací přímo optimalizována.

Každá větší webová aplikace hostuje front-end část a v jejím rámci posílá webové požadavky na svůj server, kde je spuštěný back-end aplikace. Tyto požadavky jsou spojené s dynamicky objevujícím se obsahem, který může být:

- Načtení reklamy na sociální síti.
- Potvrzení objednávky na e-shopu.
- Provádění náročnějších kalkulací pomocí výkonu co poskytuje server.
- A další.

V případě navrženého systému sice nemá mobilní aplikace vlastní back-end server, ale zato má WebServer hostovaný na mikrokontroleru. Ten přijímá příkazy jako "zrychlit dopravník" a naopak vysílá informace jako je rychlosť dopravníku.

Z těchto důvodů je mnohem snažší navrhnut nejdříve webovou aplikaci, implementovat komunikaci obou částí webového systému a následně konvertovat webovou aplikaci do mobilní aplikace. Pokud by byla mobilní aplikace programovaná od základu v knihovnách

pro tvorbu mobilních aplikací, zmizel by jednoduše vyřešitelný problém konverze webové aplikace do mobilní, ale přibyl by problém implementace webové komunikace uvnitř mobilní aplikace.

Adresy webové aplikace

old text

Ta aplikace samotná je čistě frontendová (statická) webová aplikace postavená na Vite verzi Reactu. Capacitor tuhle aplikaci převádí do mobilní aplikace a kvůli tomu potřebuju používat některý balíčky spojený s capacitorem aby mi to umožnilo posílat webové požadavky na IP adresy nodeMCU serverů.

Webová aplikace má tři URL adresy:

- Vstupní stránka - Tato stránka obsahuje hlavní část aplikace která umožňuje ovládání dopravníků.
- Setup - Tato stránka obsahuje návod jak dopravník nastavit aby s tímto zařízením správně fungoval.
- Help - Tato stránka obsahuje časté chyby které můžou nastat při používání zařízení a při nastavování dopravníků a snaží se poskytnout rady aby pomohla s vyřešením problémů.

Tyto URL adresy jsou převedené i do adres dostupných na mobilní aplikaci. Navigaci na tyto adresy zajišťuje header s navbarem jelikož v android WebView nelze zadávat URL adresy a tak musí navigace probíhat přes tlačítka na webu.

Co je to React a proč se hodí na návrh této webové aplikace. Co je to HTML, CSS a JS. Co je to reaktivnost komponentů. Jak se react využívá. Možná sem přidat co je to tailwind a Lucide for React.

React je web developmentovej framework a hodí se na návrh, protože přes node package manager už existují frameworky, který mi umožní webový aplikace portovat do mobilní aplikace - tenhle framework se jmenuje capacitor. Capacitor používá WebView (pro Android) nebo WKWebView (pro iOS) a díky tomu umožňuje zobrazit si webové aplikace na zařízení.[20]

Webovou aplikaci navrhují, protože to, co chci aby dělala je aby jenom posílala GET požadavků na IP adresu mikrokontrolleru kterou hostuje NodeMCU, což je hodně jednoduchá věc na implementaci ve webové aplikaci.

React může mít zase jako zdroj nějakou odbornou literaturu zaměřenou na react (pokud něco takového najdu) anebo online dokumentaci.

2.4.2 Použité knihovny a technologie v aplikaci (0.5 strany)

Vysvětlit co za frameworky (knihovny) jsem při developování aplikace použil aby to fungovalo co nejlépe

HTML, CSS a JS

TypeScript

Tailwind

DaisyUI

Lucide for React

V rámci programování jsem používal několik rozšíření, které web developerům pomáhají v designu aplikací. Tím je **Tailwind**, což je rozšíření které umožňuje stylizovat kód webové aplikace pomocí vlastnosti `className`, kterou mají veškeré HTML prvky a tím není potřeba mít samostatné soubory v CSS (jazyk který webům dává jejich design). Na Tailwind navazuje rozšíření **DaisyUI**, které zjednodušuje celkový design aplikace tím, že mi umožňuje si nadefinovat barvy motivu aplikace v různých proměnných, díky čemuž je možné motiv měnit skrz změnu jedné proměnné nastaveného motivu. Také umožňuje mít vlastní styl aplikace pokud má mobilní zařízení temný režim nebo světlý režim. S designem aplikace ještě pomáhalo rozšíření **Lucide pro React**, které obsahuje velké množství ikon pro různé prvky uživatelského rozhraní, jako je třeba ve tlačítkách ON/OFF, Add Conveyor, zrychlení, atd.

Aplikace ještě používá TypeScript, což je nadstavba JavaScriptu, která umožňuje definovat typy proměnných, což zvyšuje bezpečnost programu a zlepšuje zážitek z programování, vzhledem k tomu, že programátoři upozorňuje na chyby v kódu, na které by JavaScript běžně neupozornil.

2.4.3 Princip komunikace s NodeMCU servery

Co je to GET požadavek a jak se dá implementovat v JavaScriptu.

Jak se GET požadavek mění když používám capacitor

Co je to GET požadavek

Vysvětlit obecně jak fungují GET požadavky z web developmentu

Aplikace bude ovládat nodeMCU servery přes posílání webových GET požadavků. GET požadavek je typicky například když do prohlížeče na PC zadám do URL adresy jakýkoliv název webu - tak to provádí GET požadavek na server, který je na IP adrese hostován. V tomhle případě jsou servery moje nodeMCU servery a místo PC zadávám požadavky přes webovou aplikaci.

V rámci nodeMCU se nastavují adresy na kterých server nějakým způsobem odpovídá. Během tohoto nastavení můžu nejenom určit jaká stránka se ukáže po zadání požadavku na získání informací z webové adresy, ale můžu tím spouštět i jakýkoliv jiný kód.

Typický test této funkce je pomocí GET požadavku na IP adresu rozsvítit LED diodu, která přes breadboard zapojená do výstupního pinu vývojové desky WEMOS D1 mini pro. Pokud je deska připojená ke stejné WiFi síti jako můj počítač, můžu na počítač zadat například adresu 192.168.0.144/ledON. NodeMCU server na tuto adresu odpoví tím, že změní stav LED diody v desce a až poté pošle zpátky HTML kód, který se mi zobrazí v prohlížeči. Takto je možné provádět jakýkoliv kód, který je nastaven aby se prováděl v

rámci GET požadavků na jakoukoliv adresu.

V reactu se běžně GET požadavky provádí pomocí asynchronní funkce fetch. Asynchronní funkce jsou speciální funkce, během kterých můžeme používat speciální slovo "await". Tohle slovo způsobí, že Javascript počká než se daný příkaz dokončí a až poté bude pokračovat v provádění funkce. Tohle zamezuje vznik chyb v kódu, které můžou vznikat pokud se snažím dělat operace s proměnnými, které jsou zatím prázdné (jelikož ty data ještě například neposlal server).

V běžné react aplikaci by bylo možné posílat GET požadavky na nodeMCU server tímto způsobem:

```

1  async function getData() {
2      const url = "https://example.org/products.json";
3      try {
4          const response = await fetch(url);
5          if (!response.ok) {
6              throw new Error(`Response status: ${response.status}`);
7          }
8
9          const json = await response.json();
10         console.log(json);
11     } catch (error) {
12         console.error(error.message);
13     }
14 }
```

Ukázka kódu 2.3: Základní způsob posílání GET požadavků v JavaScriptu

[MDN web docs](#)

Implementování GET requestů do aplikace

Vysvětlit proč normální GET requesty nefungují (Omezení z WebView) a jak je tedy implementovat. Taky zmínit že jsou implementované ve funkci sendCommand.

GET požadavky není kvůli capacitoru možné posílat běžným způsobem, protože v android aplikaci nefunguje fetch funkce tak, jak se od ní očekává. Je ale možné použít balíček vytvořený komunitou capacitoru který se jmenuje CapacitorHttp (což bylo nedávno přidané do základního capacitor balíčku). Je to balíček, který zjednodušuje posílání GET požadavků na servery v rámci Capacitor aplikací tím, že má metodu GET, která funguje jako běžný fetch požadavek typu GET, ale je upravený aby fungoval ve WebView mobilním prostředí.

Tímto způsobem se v aplikaci posílají GET požadavky:

```

1 // pred zacatkem komponentu:
```

```

2 import { CapacitorHttp } from "@capacitor/core";
3 import { HttpOptions } from "@capacitor/core/types/core-plugins";
4
5 // uvnitř komponentu se strankou aplikace:
6
7 // Send a command to a specific conveyor and update its status
8 const sendCommand = async (ip: string, command: string) => {
9   try {
10     const options: HttpOptions = {
11       url: `http://${ip}/${command}`,
12     };
13
14     const response: any = await fetchWithTimeout(
15       CapacitorHttp.get(options),
16       3000 // Fail fast if conveyor is unresponsive
17     );
18
19     setErrorMessage(null);
20     console.log(response);
21     try {
22       const responseData = await response.json();
23       console.log("Response data:", responseData);
24     } catch (error: any) {}
25     } catch (error: any) {
26       console.error("Command failed:", error);
27       setErrorMessage(`Command failed for ${ip}: ${error.message}`);
28
29     // Immediately mark the conveyor as offline when a command fails
30     setConveyors((prevConveyors) =>
31       prevConveyors.map((conv) =>
32         conv.ip === ip ? { ...conv, isOnline: false } : conv
33       )
34     );
35   }
}

```

Ukázka kódu 2.4: Funkce sendCommand dostupná uvnitř vstupní stránky aplikace

Nejdřív je nutné si importovat HttpOptions a CapacitorHttp z capacitoru. Následně pokračuje definice komponentu, který obsahuje celou vstupní stránku aplikace. Uvnitř

stránky aplikace je definovná funkce sendCommand.

Funkce sendCommand má jako vstupy IP adresu a adresu na kterou bude posílat GET požadavek. Princip je takový, že se do HttpOptions nastaví jako URL celá IP adresa i s adresou požadavku a to se pomocí CapacitorHttp.get funkce pokusí získat. Pokud byl požadavek úspěšně doručen, aplikace se bude pokoušet přeložit odpověď přes json syntaxi, ale to se často pokazí, protože aplikace v rámci některých odpovědí odpovídá i HTML kódem aby bylo možné ji ovládat i přes webový prohlížeč. Pokud se tedy přeložit odpověď nepovede, není to žádný problém a proto je v pořádku mít rádek s přeložením ze json souboru ve try catch bloku, který jakýkoliv error potlačí.

Pokud funkce nezíská odpověď do 3 sekund, kód aplikace pomocí fetchWithTimeout (moje vlastní funkce definovaná jinde v kódu) vyšle error, který do konsole vypíše, že požadavek z nějakého důvodu nebyl doručen a zároveň zobrazí i error v uživatelském rozhraní aplikace. Tohle je obzvlášt důležitá funkce, protože se uživateli v aplikaci ukáže error pokud v rámci času 3 sekund nodeMCU server neodpoví na požadavek - informuje to tedy o tom, že uživatel budto zadal špatnou adresu, anebo odešel z dosahu ve kterém je nodeMCU server schopný se připojit na WiFi hotspot jeho mobilního zařízení. Tato implementace také vypíná možnost mačkat tlačítka, které ovládají dopravník, vzhledem k tomu, že by tlačítko vůbec nic nedělaly.

2.4.4 Funkčnost aplikace (hodně stran)

Tady už vůbec neřešit jak ta aplikace vypadá, ale zaměřit se na to co ta aplikace dokáže a jak to dělá. Struktura hlavní stránky, správa seznamu dopravníků, ovládání dopravníků je zahrnuto v sendCommand, získávání a zobrazování stavu skrz sendCommand a stránky Setup a Help. **TOHLE BUDE HLAVNÍ MEAT TÉHLE KAPITOLE**

Základ hlavní stránky je využívání reaktivnosti Reactu. Jelikož je aplikace dělaná v Reactu, je možné plynule přidávat dopravníky do seznamu bez toho aby se aplikace musela načítat pořád znova. React nám také dává možnost ukládat IP adresy dopravníků do lokální paměti stránky a tak si aplikace vždy při spuštění načte data o dopravnících, které v aplikaci byly při posledním ukončení. React také umožňuje veškeré další funkce jako implementace GET požadavků a na základě těchto GET požadavků upravovat vzhled aplikace - jako například, že pokud GET požadavek nedostane úspěšnou odpověď do tří sekund, aplikace vyhodnotí daný nodeMCU server jako nefunkční a na základě toho uživatele vizuálně upozorní a vypne možnost se pokoušet o spojení s zařízením.

Co funkci sendCommand používá

Vysvětlit proč je funkce sendCommand tak důležitá

Funkci sendCommand používají všechny tlačítka aplikace, které je možné vidět na obrázku 2.3 (tlačítka ON/OFF, zrychlení a zpomalení dopravníku).

Funkce sendCommand se ale navíc sama provádí každé 2 sekundy pro každý dopravník přidaný do vstupní stránky aplikace. Provádí se tam GET požadavek na adresu /getData na adresu každého nodeMCU serveru. Tato adresa odpovídá s aktuální (approximovanou) rychlostí dopravníku a s aktuálním stavem jestli je dopravník ovládaný lokálně nebo dálkově. Tato adresa už odpovídá pouze json souborem a díky tomu je důležité se i v rámci sendCommand pokoušet o přeložení odpovědi do javascript objektu pomocí příkazu *response.json()*. V případě těchto požadavků už program neselže s chybou a díky tomu uživatelské rozhraní aplikace získává informaci o rychlosti a stavu dopravníků, kterou

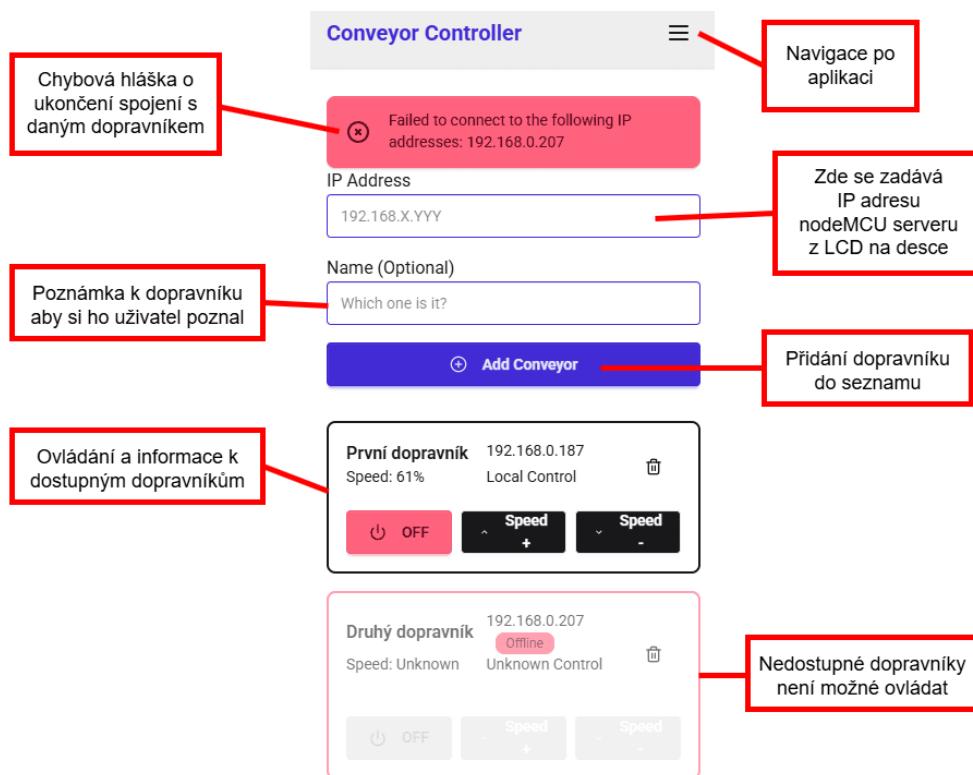
může zobrazovat ve vstupní stránce jak je vidět na obrázku 2.3.

2.4.5 Design aplikace (1 strana)

Zde vysvětlit jak jsem postupoval při designování aplikace. Vysvětlit co je to header aplikace a že mám nějaký styling kterej je pro každý React komponenty aplikovanej automaticky.

I přesto, že je aplikace programovaná jako webová aplikace jsem aplikaci designoval tak, aby vypadala v pořádku na mobilním zařízení. Vždy, když jsem aplikaci upravoval z grafické stránky, díval jsem se na ni přes *Chrome developer tools*, ve kterých lze nastavit aby se web zobrazoval jako na mobilním zařízení. Tohle usnadnilo programování uživatelského rozhraní.

Aplikace má motiv pro světlé rozhraní telefonu (light mode) i pro temné rozhraní (dark mode). Zde je zobrazeno světlé rozhraní.



Obrázek 2.11: Popis designu hlavní stránky aplikace

2.4.6 Konvertování webové aplikace do mobilní aplikace (0.5 strany)

Zmínit že je možné react aplikaci pomocí Android Studia portnout do .apk souboru který používá WebView a dále co je to CORS a že operační systém androidu defaultně zakazuje HTTP requesty.

V téhle části asi jenom zmíním, že je možné tuhle react aplikaci přes android studio portnout do .apk souboru, který se dá nainstalovat na android telefonech. Přes android studio je i možné aplikaci certifikovat a uploadnout na play store, ale to nemám zapotřebí, protože nechci aby tuhle aplikaci měli lidi co nejsou z Honeywellu.

Tady taky můžu zmínit co je to CORS (Cross Origin Resource Sharing), což je protokol který zamezuje serverům jako je nodeMCU v komunikaci s ostatními klienty kvůli bezpečnosti. Vzhledem k tomu, že IP adresu nodeMCU serverů může být úplně jakákoliv, musím CORS nastavit na možnost odpovídat na jakékoliv GET requesty z jakékoliv adresy, což teoreticky není doporučované kvůli snížení bezpečnosti proti útokům. V praxi to je ale úplně jedno, protože ten nodeMCU server je jenom na mého hotspoutu a odjinud z internetu není dostupný.

Druhá věc co můžu zmínit je, že operační systém android zařízení defaultně zakazuje všechny HTTP requesty a umožňuje jenom HTTPS requesty. HTTPS ale také nemá cenu zavádět na nodeMCU serveru protože je na lokální síti. V rámci aplikace musím ale HTTP requesty povolit tím, že to napišu do AndroidManifest.xml filu.

Tady budou zdroje asi jenom online - capacitor dokumentace, android studio dokumentace, CORS dokumentace

Požadavky na React aplikaci aby se dala konvertovat

Vysvětlit co je potřeba splnit aby se aplikace dala konvertovat do .apk

Abych tu react aplikaci mohl konvertovat pomocí kapacitoru, musí to být statická webová aplikace - tedy jen aplikace s frontendem bez serverových endpointů. Na každou routu musí existovat nějaký odkaz v navigaci aplikace (protože v capacitor aplikaci nemůžu jen tak zadat URL). A možná další věci.

Kvůli těm URL adresám musí v aplikaci i být header s navbarem. Header obecný název pro tu část aplikace co je hned nahore a většinou obsahuje navigační prvky - jako navbar, což je část aplikace která obsahuje tlačítka přes které se lze dostat na další stránky.

Ještě zmínit jakým způsobem mají být řešení odkazy v single page aplikaci aby bylo možný se skrz ni navigovat plynule - tedy že nepoužívat anchor tagy.

Asi capacitor docs.

Postup konvertování webové aplikace

Už nepsat teorii jak by se dala aplikace zkonzervovat jako ve kapitole 2.4.6 ale specificky napsat jak jsem postupoval při konvertování aplikace krok za krokem.

Pro konvertování jsem použil android studio verze x.x. atd.

Konvertování webové aplikace do mobilní aplikace je postup o několika krocích který využívá kapacitor inicializovaný v projektu a následně Android Studio, které je potřeba pro překonvertování projektu z kapacitoru do souboru instalovaného na android zařízeních o příponě .apk.

- Začíná se v nejnovější verzi React webové aplikace.
- Tuto aplikaci člověk musí nejdříve postavit do produkční verze pomocí příkazu `npm run build`.
- Dále je potřeba aplikaci zesynchronizovat s kapacitorem pomocí příkazu `npx cap sync`.
- Nyní je potřeba si otevřít nainstalovaný program android studio, co je možné udělat rovnou z konzole pomocí příkazu `npx cap open android`.

- Nakonec je v android studio potřeba znovu postavit aplikaci do produkční verze pomocí příkazu build.

Na konci tohoto procesu je dostupný soubor přípony .apk, který je možné si poslat na android mobilní zařízení a tam nainstalovat.

Aplikace nevyžaduje žádné další nastavování.

2.5 Vytvoření schránky pro desku (1 strana)

V téhle sekci bude popis jak jsem postupoval při návrhu schránky pro desku, která obsahuje i tlačítka.

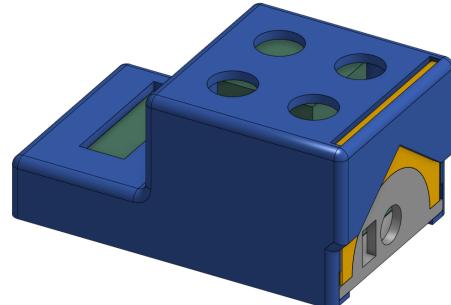
Myšlenka při návrhu schránek pro desku byla taková abych ji mohl vytisknout v běžných podmínkách pomocí 3D tiskárny Bambu Lab A1 Mini, kterou mám doma. Hlavní důvod proč jsem zvolil 3D tisk jako technologii bylo, že těchto schránek bude ve finále vytisknutých asi 5 kusů a tak není potřeba zajišťovat sériovou výrobu. Navíc je to pro schránky na desky plošných spojů levné řešení, které dosahuje dostatečné kvality provedení. Nároky na schránku jsou základní - je důležité mít možnost ji rozdělat aby byla možná údržba desky, ale jinak nemá zvláštní nároky, vzhledem k tomu, že bude vytažená pár týdnů do roku.

Modelovací software pro návrh schránky pro desku jsem zvolil online CAD Onshape. Do toho jsem si nahrál 3D model desky plošných spojů v .STL formátu který mi vygeneroval KiCAD pomocí 3D prohlížeče a na základě tohoto modelu jsem začal modelovat schránku na míru pro moji desku. Rozhodl jsem se, že schránka bude vytvořena ze dvou hlavních částí a že bude bez šroubů, aby bylo možné na ni jednodušeji prototypovat zařízení, ale zároveň bylo cílem aby stále držela dohromady během používání, pokud zrovna není potřeba ji otevřít.

To se mi povedlo pomocí dvou částí. Do dolní části je možné zajet desku plošných spojů a díky tomu deska ve schránce dobře drží. V dolní části je také otvor na kabely které jsou uchycené v desce plošných spojů. poté je možné vzít celou dolní část a zajet ji do horní části schránky, která obsahuje prostor pro tlačítka, LCD display a díru na našroubování antény pro zlepšení připojení k WiFi. Nakonec schránka obsahuje ještě jednu část a to je brána, který dolní část desky zajistí aby nevyjízděla z horní části.



(a) Pohled shora na schránku



(b) Pohled z boku na schránku

Obrázek 2.12: Model schránky pro desku plošných spojů

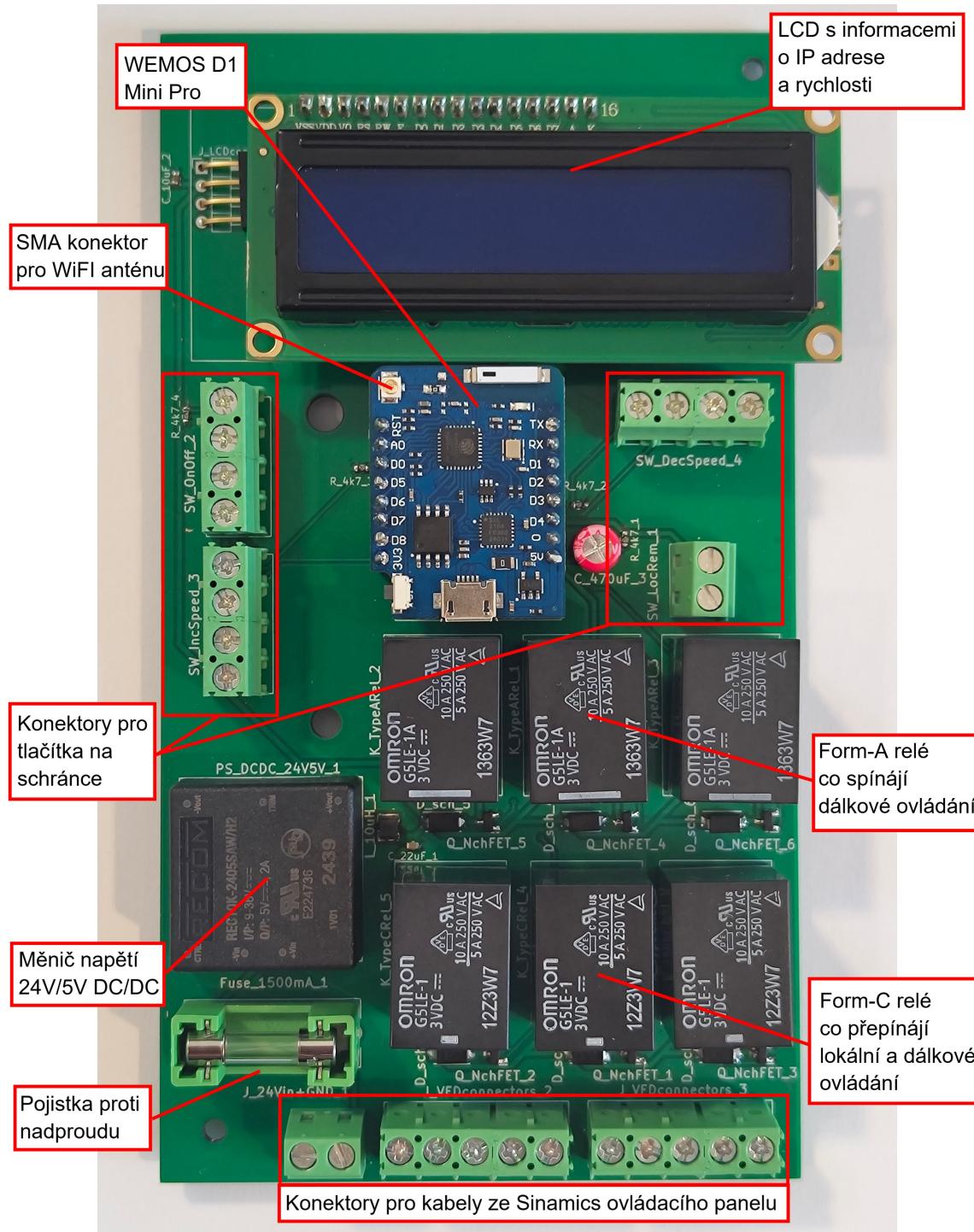
Po vytvoření modelů v Onshape jsem si pomocí Bambu Studio do tiskárny poslal desku a vytiskl jsem ji z bílého PETG materiálu značky SUNLU, který jsem vybral díky

jeho vyšší robustnosti než PLA, ale zároveň nepotřebuje složitější procesy a hardware pro tisknutí. Během tisknutí jsem filament sušil kvůli notoricky známým problémům PETG a vlhkosti filamentu.

2.6 Kompletace řešení (2 strany)

Tady bude postup kompletace celého zařízení a k tomu obrázky, které blíže popisují kde jsou jednotlivé části.

Při kompletaci jsem připájal všechny součástky na desku a vložil jsem do desky dvě součástky, které jsou vyjmoutelné - vývojovou desku WEMOS D1 Mini Pro a LCD Displej. Tyto součástky jsou vyjmoutelné protože to jsou složité součástky sestavené z více různých částí, ale nejsou ve formě uzavřených integrovaných obvodů. Kvůli tomu existuje riziko, že by nebyly správně kompletované a v tom případě se hodí mít možnost je jednoduše oddělat a odstranit na nich problémy. Vývojová deska se navíc musí často oddělávat aby bylo možné na ni nahrávat kód při tvoření a LCD má zezdola trimmer, kterým se mění viditelnost textu.



Obrázek 2.13: Finální podoba desky plošných spojů

Následně stačí jen přidělat kabely do desky na šroubovací konektory a složit schránku na desku. V tomto stavu by mělo zařízení být připravené na připojení na dopravník jako to je v obrázku 2.1. Desku je možné napájet buďto z ovládacího panelu frekvenčního měniče, což je doporučené, jelikož jsou do tohoto ovládacího panelu připojené i další kabely vycházející ze zařízení, ale je možné desku napájet i z jakéhokoliv zdroje napětí, který má napětí mezi 9 – 48V a je schopný dodávat proud do hodnot asi 0.5A.

3 Ověření funkčnosti návrhu ($\Sigma = 4$ strany)

Pro testování funkčnosti systému jsem zkoušel celý systém nastavit přesně podle instrukcí obsažených v Setup stránce dostupné v mobilní aplikace. Poté jsem připojil celý systém na digitální vstupy do ovládacího panelu frekvenčního měniče a připojil jsem systém na napájecí napětí 24V OUT vycházející z ovládacího panelu. Jak jsem zkontoval, že je všechno nastavené jak má být, otestoval jsem, jestli lokální ovládání spíná digitální vstupy ovládacího panelu. Po kontrole, že tomu tak je, je možné zapnout výkonovou část frekvenčního měniče (tedy část mimo ovládací panel). Tohle spustí dopravník.

Při zapínání výkonové části frekvenčního měniče je důležité si dát pozor, aby approximace rychlosti začínala na nule, protože jinak bude začínat approximaci na posunuté hodnotě.

Všechno testování bylo provedeno v Brněnské hale společnosti Honeywell. V té je rozšířeno několik různých typů dopravníkových linek které Honeywell Brno nabízí svým zákazníkům na fyzických i virtuálních prohlídkách. V době, kdy byla v hale testovaná funkčnost celého systému byla asi 80 metrů daleko v jiné části haly spuštěná jiná dopravníková linka, ale vzhledem k tomu, že vzdálenost dosahu WiFi signálu vyšla v rámci přijatelných mezí, lze předpokládat, že nebyla nijak druhou linkou zarušena. Je možné, že dosah bude v hale zákazníka jiný - to záleží podle dalších zdrojů rušení, hustoty dopravníků v dané oblasti a přítomnost dalších faktorů, které by mohly rušit WiFi vlny.

3.1 Ověření funkčnosti lokálního ovládání (0.5 strany)

Tady bych se chtěl zaměřit na to, jestli je možné dopravník ovládat přes tlačítka rovnou umístěná na desce.

Tohle jsem už testoval a funguje to jak má. Ty tlačítka spínají přímo těch 24V z ovládacího panelu dopravníku, takže fungují i když se vůbec nepřipojí ten mikrokontroller - tudíž deska funguje i bez připojení kabelu který vede na 24V OUT port, ale potom nefunguje LCD displej pro approximaci rychlosti dopravníku.

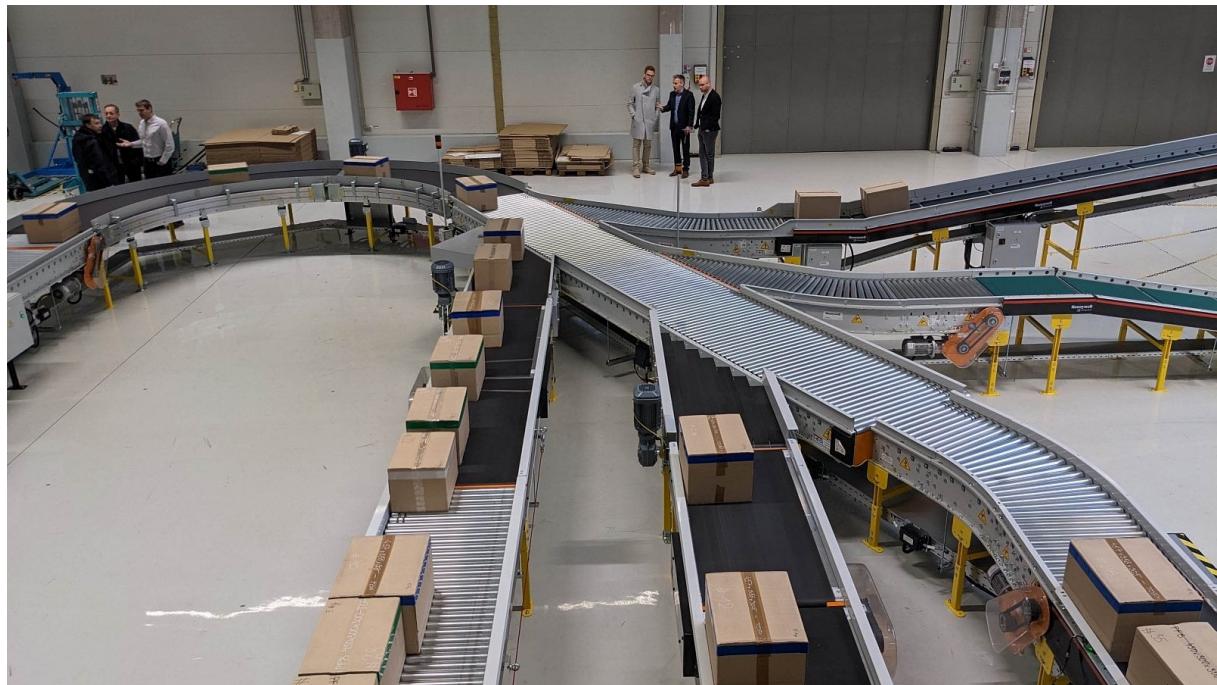
3.2 Ověření funkčnosti mobilní aplikace (1 strana)

Tady jenom potvrdím že mobilní aplikace opravdu funguje

Po zapojení dopravníku a otestování lokálního ovládání byla testována mobilní aplikace.

Pro používání mobilní aplikace jsem zapnul systém do stavu pro dálkové ovládání. Do mobilní aplikace následně stačí pouze přidat IP adresu která je vidět na LCD displeji desky a kliknout tlačítko "Add Conveyor". Po přidání dopravníku do aplikace je možné spustit dopravník a po spuštění dopravník zrychlovat i zpomalovat. Approximace rychlosti

3 OVĚŘENÍ FUNKČNOSTI NÁVRHNU(STĚ ZASPIRANÝ)VÍCE DESEK (0.5 STRANY)



Obrázek 3.1: Ukázka Brněnské haly pro testování dopravníků společnosti Honeywell [21]

lze úspěšně vidět nad tlačítka pro ovládání dopravníku. Při ovládání dopravníku na dálku je také typ ovládání v aplikaci viditelný jako "Remote".

V rámci tohoto testu jsem šel na X metrů daleko a zkontroloval že ta deska funguje i na nějakou vzdálenost.

3.3 Ověření funkčnosti zapojení více desek (0.5 strany)

Tady bych se rád zaměřil na nějaké testy, při kterých zkouším, jestli je opravdu možné ovládat více desek zároveň.

Pro otestování jsem nastavil a zapojil dvě desky podle návodu u dvou dopravníků několik metrů od sebe. Následně jsem testoval vzdálenost kde se mi podařilo si zachovat spojení s obouma deskami na X metrů. Tohle bylo znova testované v Brněnské Honeywell hale. Při testování jsem sledoval jestli se dopravníky hýbou tím, že jsem si na dopravník položil testovací balík a u toho bylo vidět, jestli se při dané vzdálenosti hýbe anebo už ne.

3.4 Posouzení z hlediska bezpečnosti (1 strana)

Tady bude nějaký moje zamýšlení nad bezpečností téhle desky.

Ta deska už ze své podstaty má umožňovat ovládat dopravník na dálku a někdy i třeba hodinu - aby si sedly všechny součásti a tak bylo vidět, jestli je dopravník opravdu v pořádku za chodu. Tohle může být inherentně nebezpečná věc pokud by operátor přišel o spojení s deskou ve špatný okamžik a nemohl tak dopravník zastavit. Je možný, že by měla obsahovat nějaké test (třeba každou sekundu), jestli je otevřená aplikace na mobilu, ale to by kazilo user experience uživatelů, který dělají ty testy, protože oni musí při používání aplikace ještě psát informace o zkouškách do excelu. Pokud by se aplikace zavřela, tak přestane komunikovat s deskou a ta by pak chtěla ten dopravník vypnout. Tenhle bezpečnostní mechanismus tedy nemá smysl implementovat, protože by příliš kazil používání systému jako takového.

3 OVĚŘENÍ FUNKČNOSTI POSVORNÉ Z HLEDISKY BEZPEČNOSTI (1 STRANA)

Myšlenku žádný takový mechanismus nevytvářet také podporuje fakt, že navržený systém přebírá bezpečnost z těch už nainstalovaných dopravníků. Dopravníky už při této fázi kontroly kvality mají kolem sebe E-STOP lanko a E-STOP tlačítka, které jsou nastavené aby dopravníku přikázali pohotovostní zastavení, jak vyžaduje bezpečnost na pracovišti.

V rámci bezpečnosti je také důležité správně vyškolit obsluhu systému pro bezpečné používání systému. Obsluha by měla například vědět, že je potřebné vypnout výkonovou část frekvenčního měniče pomocí nainstalovaného vypínače při veškeré manipulaci s ovládacím panelem. Obsluha by také měla vědět jak správně systém nastavit a jak řešit časté problémy - obě téma obsahuje mobilní aplikace na adresách Setup a Help.

Závěr (1 strana)

Seznam zkratek a symbolů

DPS Deska plošných spojů

CSS Cascading Style Sheets

HTML Hypertext Markup Language

JS JavaScript

PLC Programovatelný logický automat

GPIO General Purpose Input Output

SDK Software Development Kit

IoT Internet of Things

mDNS multicast Domain Name System

SDA I²C - serial data line

SCL I²C - serial clock line

Seznam zdrojů

- [1] SINAMICS G120D - Siemens Global. 2025. Dostupné také z: <https://www.siemens.com/global/en/products/drives/sinamics/low-voltage-converters/distributed-converters/sinamics-g120d.html>.
- [2] SKALICKÝ, CSc. Prof. Ing. Jiří. *Elektrické regulované pohony*. Fakulta Elektrotechniky a Komunikačních Technologií, Vysoké Učení Technické v Brně, 2007.
- [3] ČERVINKA, Ph.D. Ing. Dalibor. Řízení otáček asynchronních motorů Učební text do předmětu Elektrické pohony. 2025.
- [4] SIEMENS. *SINAMICS G120D Getting Started* [https://cache.industry.siemens.com/dl/files/509/109757509/att_951176/v1/G120D_getting_started_0418_en-US.pdf]. 2018. Datum přístupu: 2025-02-24.
- [5] EVANS, Brian. *Beginning arduino programming*. Apress, 2011. ISBN 978-1-4302-3777-8.
- [6] LASKAKIT.CZ. *WEMOS D1 Mini Pro klon*. [B.r.]. Dostupné také z: <https://www.laskakit.cz/lolin-d1-mini-esp8266-v3-1-0-wifi-modul/>. Datum přístupu: 2025-03-09.
- [7] SYSTEMS, Espressif. ESP8266EX datasheet. 2025. Dostupné také z: <https://www.espressif.com/en/subscribe..>
- [8] *D1 mini Pro — WEMOS documentation*. 2025. Dostupné také z: https://www.wemos.cc/en/latest/d1/d1_mini_pro.html.
- [9] *esp8266/Arduino: ESP8266 core for Arduino*. 2025. Dostupné také z: <https://github.com/esp8266/Arduino>.
- [10] DRATEK. *ESP WiFi Webserver / Návody Drátek*. 2025. Dostupné také z: <https://navody.dratek.cz/navody-k-produktum/esp-wifi-webserver.html>.

- [11] LASKAKIT.CZ. *16x2 LCD displej 1602 s I₂C převodníkem*. [B.r.]. Dostupné také z: <https://www.laskakit.cz/16x2-lcd-displej-1602-i2c-prevodnik/>. Datum přístupu: 2025-03-09.
- [12] WIT, Sjors de. *Jitter / Electronic Design and Consultancy*. 2020. Dostupné také z: <https://jitter.nl/blog/2020/09/29/think-in-current-loops-the-key-to-reliable-pcb-layout/>.
- [13] *Your Gateway to Embedded Software Development Excellence · PlatformIO*. 2025. Dostupné také z: <https://platformio.org/>.
- [14] *C++ Classes and Objects / GeeksforGeeks*. 2025. Dostupné také z: <https://www.geeksforgeeks.org/c-classes-and-objects/>.
- [15] *Encapsulation in C++ / GeeksforGeeks*. 2025. Dostupné také z: <https://www.geeksforgeeks.org/encapsulation-in-cpp/>.
- [16] *Wifi - draft 802.11n - NMS*. 2025. Dostupné také z: https://nms.fjfi.cvut.cz/wiki/Wifi_-_draft_802.11n.
- [17] *Technology / 2024 Stack Overflow Developer Survey*. 2025. Dostupné také z: <https://survey.stackoverflow.co/2024/technology>.
- [18] *Getting Started / Vite*. 2025. Dostupné také z: <https://vite.dev/guide/>.
- [19] *Capacitor - Cross-platform Native Runtime for Web Apps / Capacitor Documentation*. 2025. Dostupné také z: <https://capacitorjs.com/docs>.
- [20] CAPACITOR. *Frequently Asked Questions / Capacitor Documentation*. 2025. Dostupné také z: <https://capacitorjs.com/docs/getting-started/faqs>.
- [21] KEJDUŠ, Radomír. *Honeywell otevírá v Brně výzkumné centrum pro dopravníkové systémy - Cnews.cz*. 2025. Dostupné také z: <https://www.cnews.cz/clanky/honeywell-otevira-v-brne-vyzkumne-centrum-pro-dopravnikove-systemy/>.

Seznam obrázků

1.1	Závislost napětí, momentu a výkonu na frekvenci [3]	12
1.2	Frekvenční měnič Sinamics G120D [1]	13
1.3	Způsoby seřizování ovládacího panelu frekvenčního měniče [4]	14
1.4	Použitá varianta vývojové desky WEMOS D1 Mini Pro [6]	18
1.5	Ukázka rozhraní vývojového prostředí Platformio	20
1.6	Ukázka nastavení WebServeru pro ovládání LED diody [10]	20
2.1	Schéma principu jak navržený systém funguje	23
2.2	Popis zařízení co ovládá dopravník	24
2.3	Vysílat příkazy zařízení bude mobilní aplikace připojená přes hotspot	25
2.4	Blokové schéma desky plošných spojů	26
2.5	Návrh desky plošných spojů v KiCAD	27
2.6	Elektrické schéma ovládání relé	28
2.7	LCD displej s I2C převodníkem [11]	29
2.8	Začátek stavového diagramu	34
2.9	Strana stavového diagramu s lokálního ovládáním	35
2.10	Strana stavového diagramu s dálkovým ovládáním	36
2.11	Popis designu hlavní stránky aplikace	45
2.12	Model schránky pro desku plošných spojů	47
2.13	Finální podoba desky plošných spojů	49
3.1	Ukázka Brněnské haly pro testování dopravníků společnosti Honeywell [21]	51

Seznam tabulek

Seznam příloh