



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. 64 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	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	38
2.4.1 Architektura aplikace	38
2.4.2 Použité knihovny a technologie v aplikaci	40
2.4.3 Princip komunikace s WebServerem	41
2.4.4 Implementování GET požadavků s Capacitor	43
2.4.5 Design aplikace	45
2.4.6 Konvertování webové aplikace do mobilní aplikace	47
2.5 Vytvoření schránky pro desku	48
2.6 Kompletace řešení	49
3 Ověření návrhu	52
3.1 Ověření lokálního ovládání	53
3.2 Ověření mobilní aplikace	53
3.3 Spolehlivost dálkové komunikace	54
3.4 Ověření zapojení více desek	54

3.5 Posouzení z hlediska bezpečnosti	54
Závěr (1 strana)	56
Seznam zkratek a symbolů	58
Seznam zdrojů	59
Seznam obrázků	62
Seznam tabulek	63
Seznam příloh	64

Úvod

Vysvětlivka značení:

Pokud zmiňuju požadavek na systém tak to 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]

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

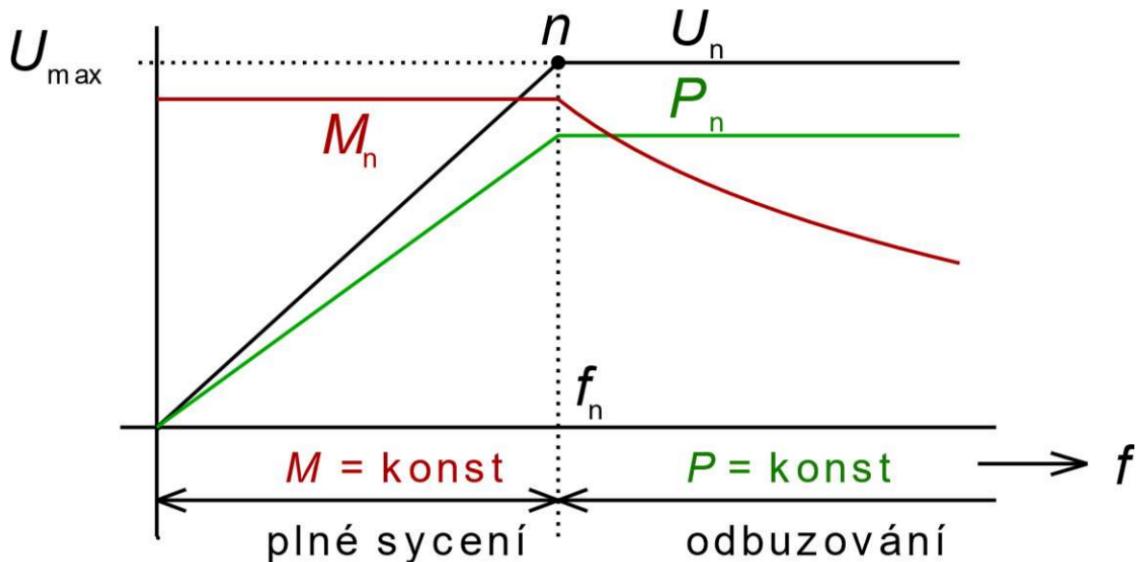
1 REŠERŠE

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.

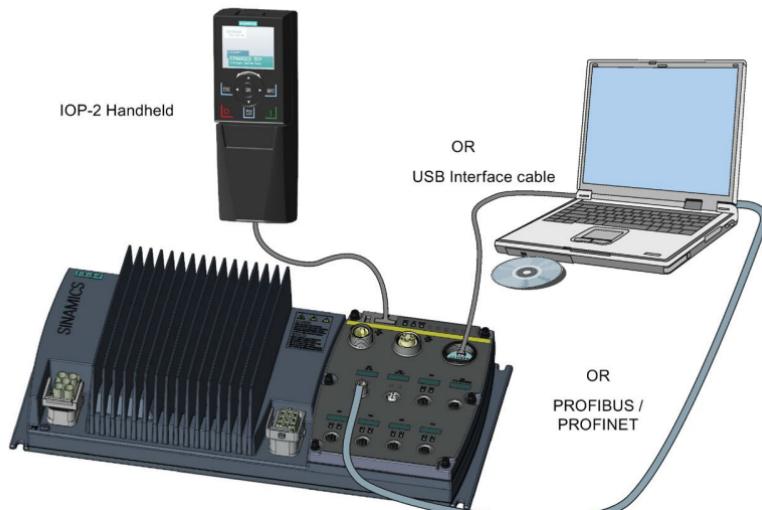
Tento frekvenční měnič je tvořen dvěma hlavními částmi - výkonová část a ovládací panel. Ovládací panel ovládá a monitoruje výkonovou část frekvenčního měniče pomocí několika kontrolních systémů na bází uzavřených smyček a díky tomu může kontrolovat bezpečný stav frekvenčního měniče a taky znemožnit ovládání, pokud by s měničem bylo něco v nepořádku. Také je schopný rekuperace energie z brždění 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 3.5. [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.

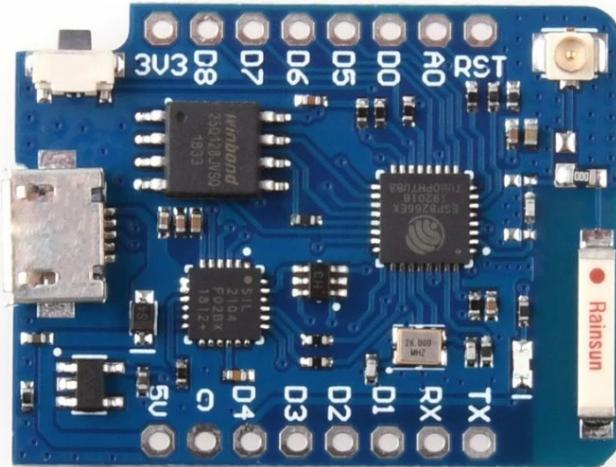
Tenho systém je ale navrhován pro industriální prostředí a je velmi důležité aby byla bezdrátová komunikace co nejspolehlivější. Proto je potřebné mít na vývojové desce k dispozici externí anténu, která významně zvýší dosah WiFi komunikace díky lepšímu umístění antény. Proto byla do systému vybrána vývojová 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]

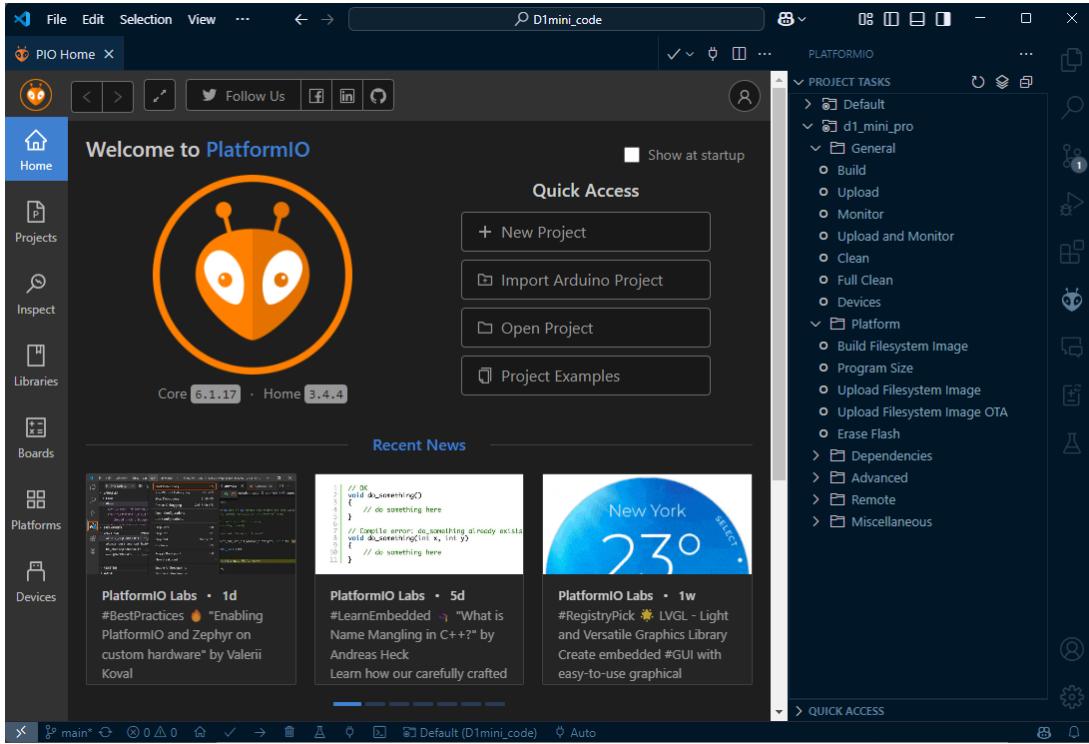
V této práci byl kód mikrokontroleru programován uvnitř programovacího prostředí PlatformIO (ukázané na obrázku 1.5) ve kterém byl Arduino framework pro ESP8266 využitý. Byla využitá i kompatibilita s existujícími Arduino knihovnami, jako v případě knihovny Ticker. To je knihovna, která v rámci Arduino 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

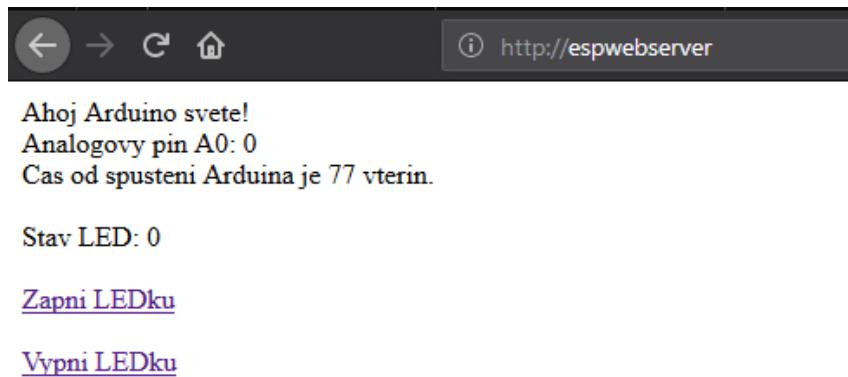


Obrázek 1.5: Ukázka rozhraní vývojového prostředí Platformio

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

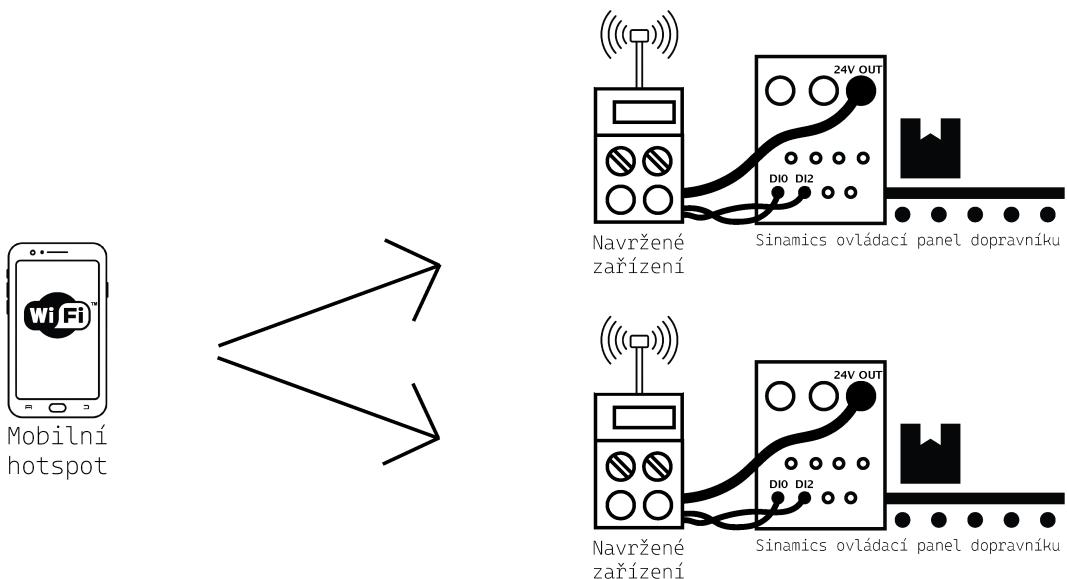
2.1 Princip funkce navrženého systému

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.



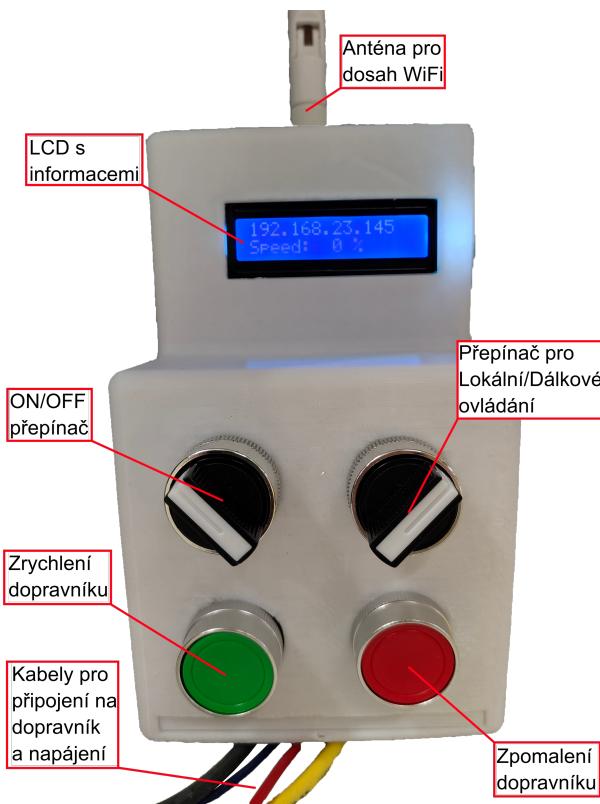
Obrázek 2.1: Schéma principu jak navržený systém funguje

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**

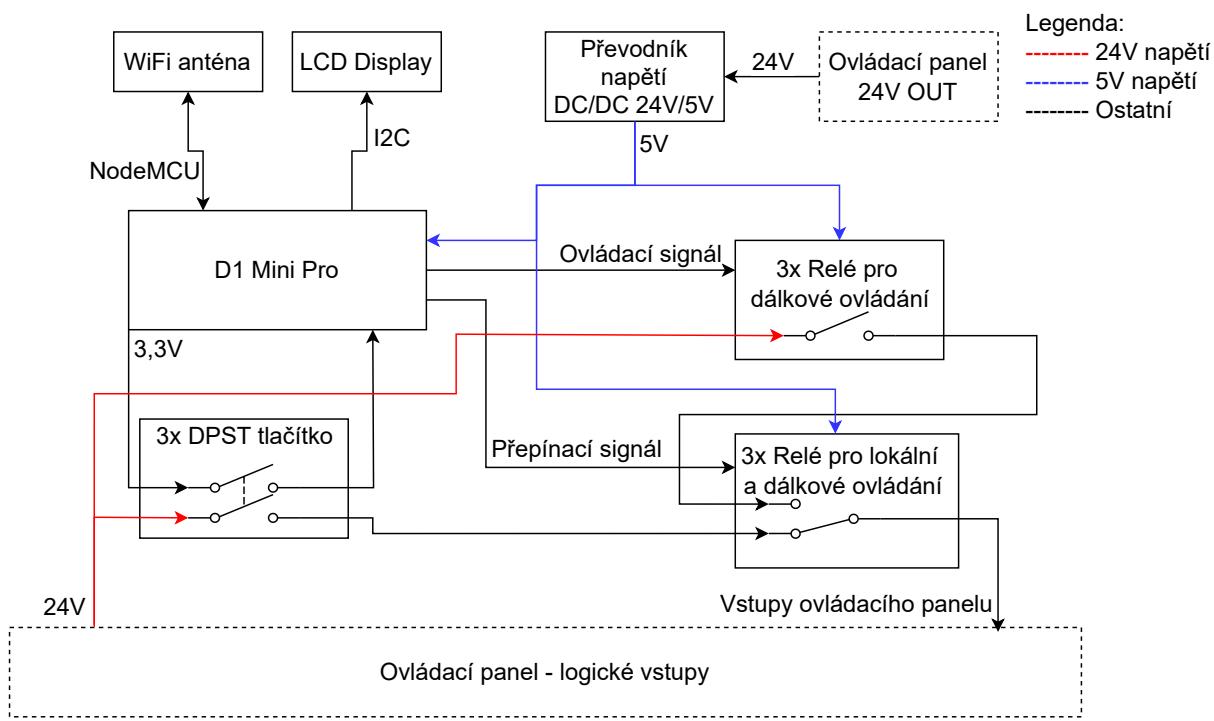
Systém by měl být navržen tak, aby bylo stále možné dopravníky ovládat i pokud by něco zamezovalo napájení vývojové desky.

- **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.

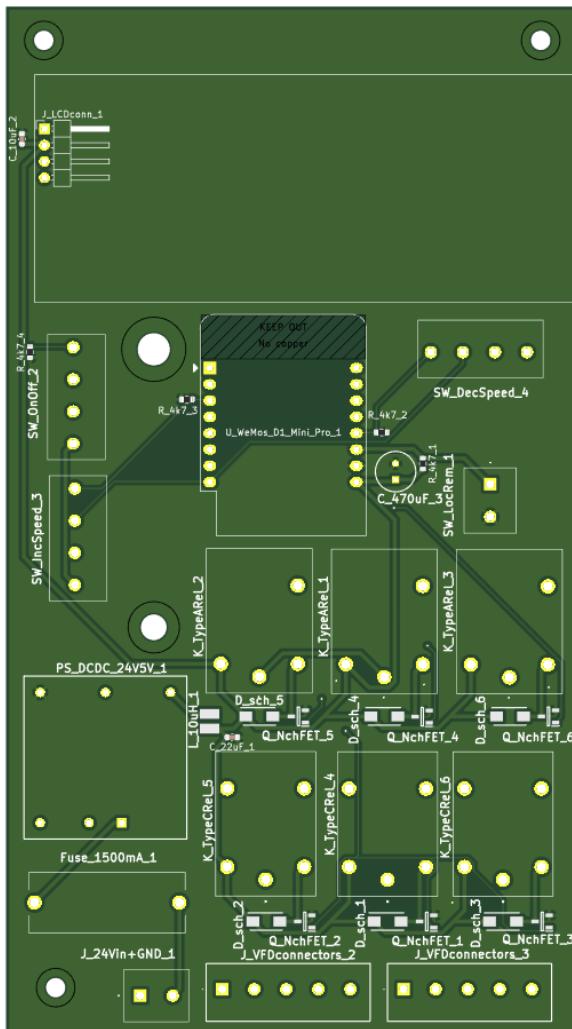


Při tvorbě desky se vycházelo z požadavků na systém. Nejdřív se vycházelo z toho, že desku musí být možné napájet z ovládacího panelu, který má výstupní port na kterém je 24V a který je schopný dodat maximálně 8A. Ze začátku se tedy počítalo s 24V napětím, pro které bylo potřeba zvolit dobrý převodník 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]

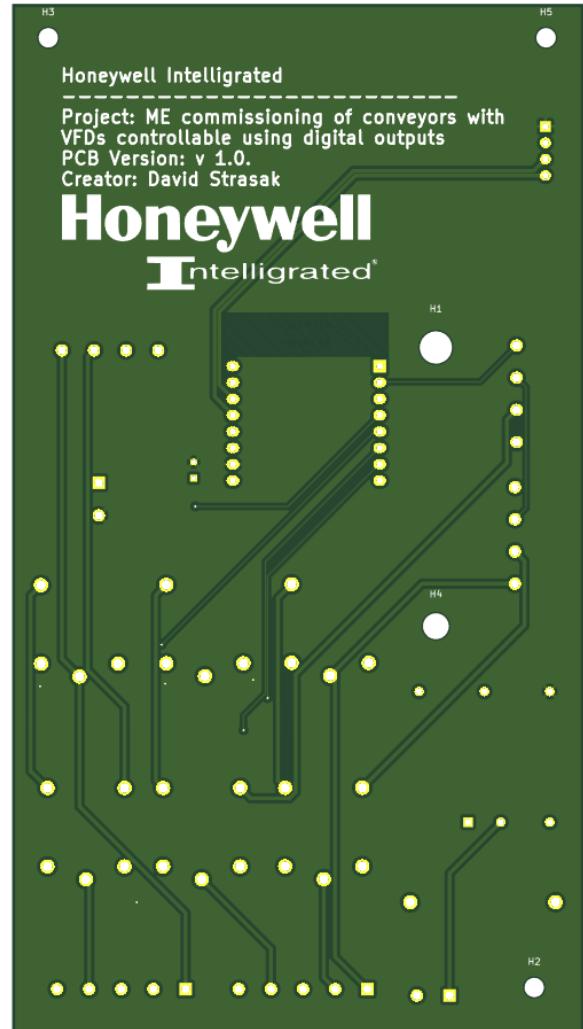
Dále se při návrhu vycházelo z myšlenky že systém musí mít i lokální i dálkové ovládání, s tím, že lokální ovládání musí být dostupné i bez napájení mikrokontroleru. Tohle bylo vyřešeno návrhem dvou přepínačů. *Relé pro lokální a dálkové ovládání* je přepínač, který je normálně v poloze lokálního ovládání (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

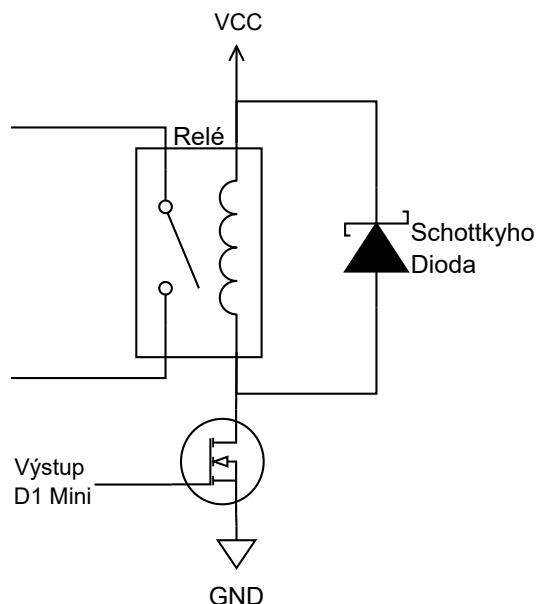
2.2.1 Ovládání relé

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

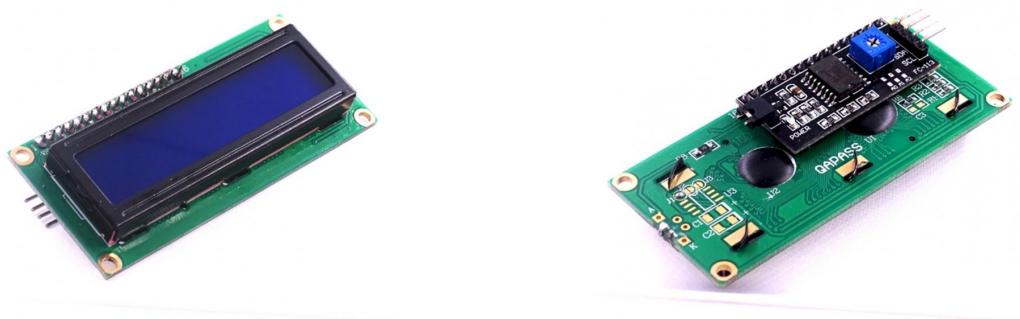
Aby bylo možné celý systém ovládat přes ESP8266 WebServer, je potřebné nějakým způsobem komunikovat s uživatelem IP adresu, kterou má vývojová deska připojená na hotspot mobilního zařízení. Tohle by bylo možné udělat například skrz nastavení multicast DNS na specifickou adresu a tu potom fyzicky napsat 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, umožňuje 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.

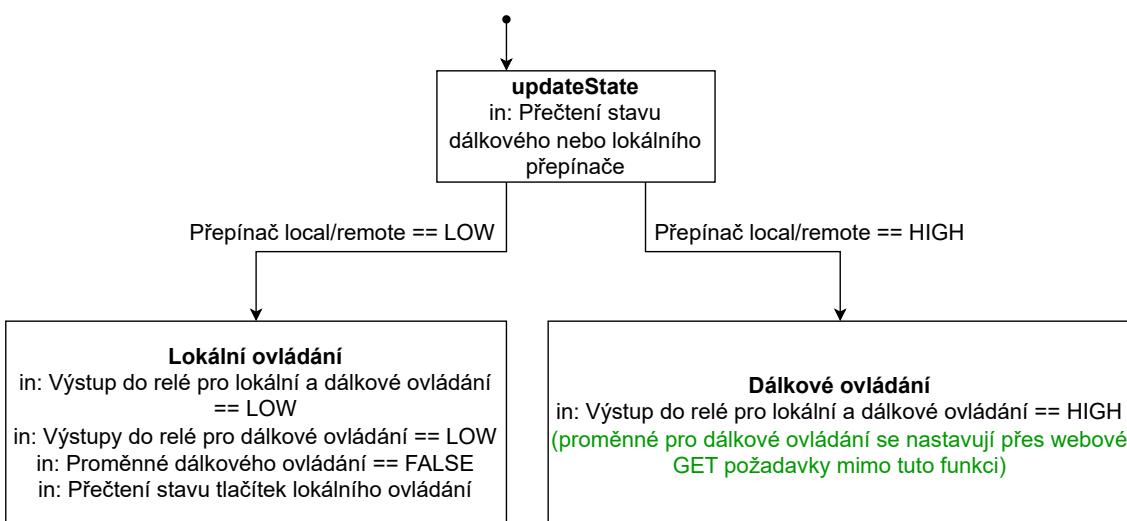
Hlavíčkový soubor dále obsahuje i několik prototypů privátních metod které urychlují psaní kódu při opakovaných úkonech a proměnných které jsou používány uvnitř metod (všechny proměnné používané uvnitř třídy jsou privátní). Specifické privátní metody které je potřeba poukázat jsou metody **mainRoute** 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.

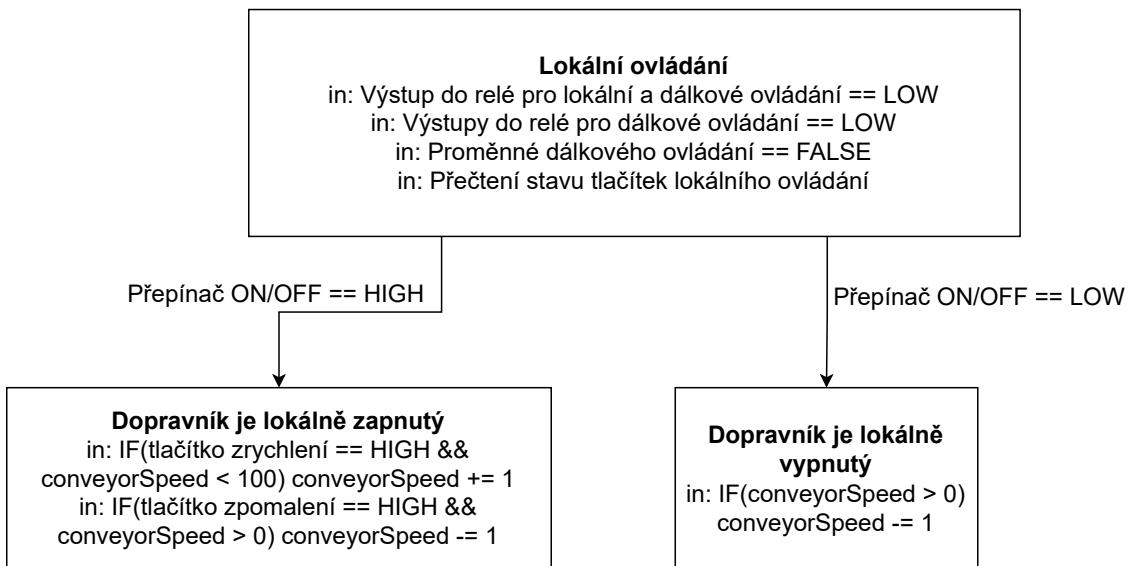
V tomhle systému je na tuto část kódu nahlíženo jako na stavový diagram Harelova typu. Pro přehlednost je zde stavový diagram rozdělený do tří různých částí (byl ale navrhován jako jeden celek):

- **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

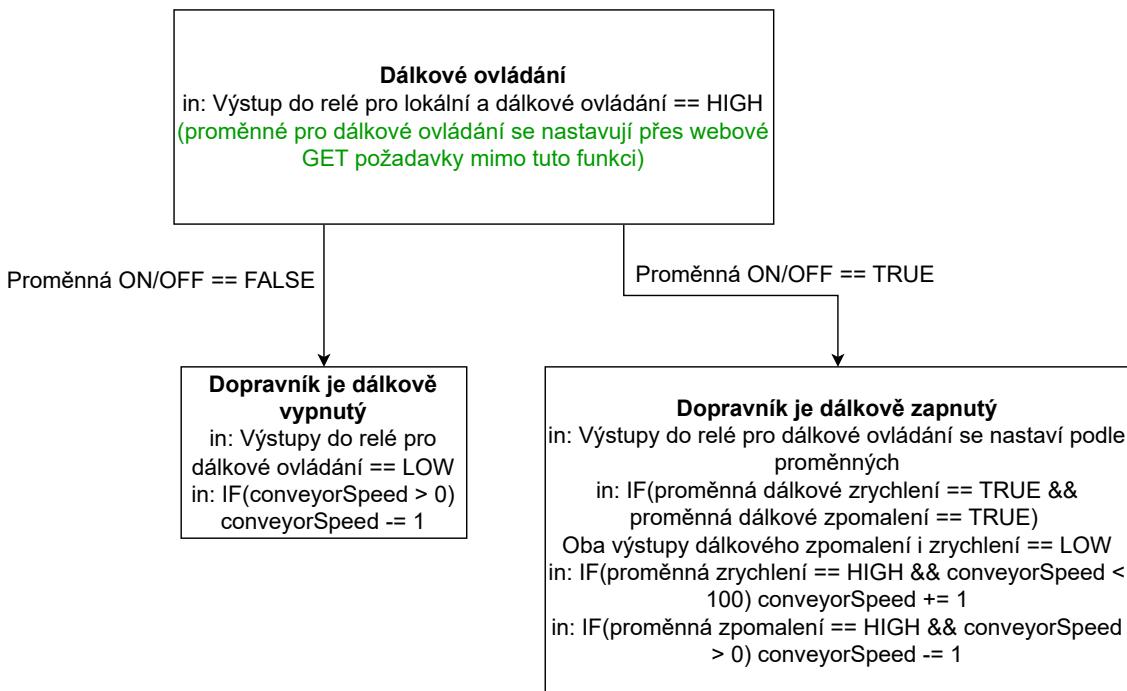
Ve stavu s lokálním ovládáním se nastaví výstupní pin vývojové desky ovládající tranzistor, který vypne relé, které přepíná mezi lokálním a dálkovým ovládáním. Následně se rovněž nastaví tranzistor ovládající dálkové ovládání, jelikož není potřeba toto relé mít zapnuté, když je tato větev obvodu deaktivována. 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, tlačítko zrychlení by bylo sepnuté po dobu těsně menší než 300ms a zrovna tato doba vyšla na dobu mezi volání funkce `updateState`. Toto je chyba která může nastat obecně u jakéhokoliv vzorkování. Pokud by se tak stalo i opakovaně, stejně by byla chyba v řádu jednotek procent, což je zanedbatelné.

Naopak častější volání funkce `updateState` než 300ms by vyžadovalo 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

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 zařízení, které bude fungovat jako dedikovaný hardwarový 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 s názvem **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. S takovým ovládáním by ale systém ztrácel další funkce, které mobilní aplikace nabízí. To je třeba zmíňovaná 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** (Více popsané v kapitole 2.4.5), které umí kterémukoliv uživateli ukázat 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, který je dostupný na lokální síti počítače, na kterém je node.js spuštěný.

Programování webové aplikace probíhalo tím způsobem, že se každá verze webové aplikace testovala 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 nározdíl 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 instalačního souboru je ale potřebné mít i stolní počítač značky Apple, kde je možné aplikaci konvertovat pomocí **Xcode**.

Proč webová aplikace?

Důvod, proč byla mobilní aplikace programována jako webová a ne přímo mobilní, je ten, že jejím hlavním účelem je posílat GET požadavky na IP adresu vývojové desky. Na tento úkol jsou moderní webové aplikace velmi dobře optimalizovány.

Každá větší webová aplikace dostupná na internetu má front-end část a v jejím rámci posílá webové požadavky na svůj server, kde je spuštěn back-end aplikace. Tyto požadavky souvisejí s dynamickým 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 rychlosť dopravníku.

Z těchto důvodů je mnohem snazší navrhnut nejdříve webovou aplikaci, implementovat komunikaci obou částí webového systému a následně webovou aplikaci konvertovat do mobilní. Pokud by byla mobilní aplikace programovaná od základu pomocí knihoven pro tvorbu mobilních aplikací, zmizel by sice snadno řešitelný problém konverze webové aplikace do mobilní, ale vznikl by problém implementace webové komunikace uvnitř mobilní aplikace.

Adresy webové aplikace

Webová aplikace má tři URL adresy:

- **Vstupní stránka (adresa "/")** – Tato stránka obsahuje hlavní část aplikace, která umožňuje ovládání dopravníků.
- **Setup (adresa "/setup")** – Tato stránka obsahuje návod, jak dopravník nastavit, aby s tímto zařízením správně fungoval.
- **Help (adresa "/help")** – Tato stránka obsahuje popis častých chyb, které mohou nastat při používání zařízení a při nastavování dopravníků, a rady k jejich vyřešení.

Tyto URL adresy zůstávají stejné i při převedení do mobilní aplikace. Je ovšem potřeba mít v aplikaci nějaký způsob, jak se pomocí tlačítek mezi těmito adresami navigovat, protože při převedení webové aplikace do mobilní chybí adresní řádek, kde by bylo možné tyto adresy zadat. Tuto navigaci v mobilní aplikaci zajišťuje hlavička s navigačním panelem, která obsahuje tyto odkazy.

2.4.2 Použité knihovny a technologie v aplikaci

Při programování webové aplikace bylo použito několik knihoven, které webovým developerům pomáhají při návrhu aplikací. Zde je krátký popis každé z těchto technologií a role, jakou zastaly při programování webové aplikace.

HTML, CSS a JS

HyperText Markup Language neboli HTML je jeden z prvních jazyků, které stály u zrodu internetu. Definuje strukturu webových stránek a obsahuje veškerý text. Dále je zde Cascading Style Sheets neboli CSS, které popisuje vzhled dokumentu napsaného v HTML. JavaScript neboli JS je odlehčený programovací jazyk s just-in-time kompliací, který je nejvíce známý tím, že dodává funkčnost webovým stránkám. Je to ale i skriptovací jazyk pro aplikace mimo webové prostředí, jako například Node.js nebo Adobe Acrobat. [20, 21, 22]

Tyto technologie jsou stále využívány na této webové stránce, ačkoli se v nich přímo neprogramovalo. Tyto programovací jazyky jsou základní stavební kameny prohlížeče, a proto je nutné každou aplikaci konvertovat do souborů těchto tří jazyků. Bez toho by nebylo možné zobrazit webovou stránku jak ve webové aplikaci, tak v mobilní aplikaci.

React

React je knihovna pro webový front-end založená na komponentech. Uvnitř těchto komponent je možné vkládat i HTML, což umožňuje spojit funkcionalitu s textem a strukturou.

React si zakládá na interaktivitě a reaktivnosti. U statických webových stránek, které jsou napsané přímo v HTML, CSS a JS, se jakákoli změna projeví tím, že se musí znova načíst celá stránka. Na druhou stranu React umí aktualizovat pouze ty části webové stránky, kde změna proběhla. Tomuto se říká reaktivnost webové stránky. [23]

TypeScript

TypeScript je nadstavba JavaScriptu, která řeší jeho nedostatky. JavaScript například nemá implementovanou silnou typovou kontrolu, a tak je možné jakékoli proměnné přiřadit jakoukoliv hodnotu. To může vytvářet chyby v kódu i přesto, že editor při psaní kódu žádnou chybu přímo neoznačil.

TypeScript je navíc vždy možné kompilovat do JavaScriptu, který je spustitelný v prohlížeči, a proto je velmi užitečný nástroj při programování složitějších aplikací. [24]

Tailwind

Tailwind umožňuje developerům webových aplikací definovat vzhled prvků přímo v HTML kódu, čímž odpadá potřeba samostatných souborů se styly, jako je běžné u CSS. [25]

DaisyUI

DaisyUI přidává speciální klíčová slova pro rozšíření knihovny Tailwind. Zatímco Tailwind poskytuje většinou specifická klíčová slova, která zpravidla přímo odpovídají jednotlivým vlastnostem CSS, DaisyUI funguje jinak. Tato knihovna umožňuje pod jedním klíčovým slovem sdružit více tailwind stylů. [26]

Díky tomu DaisyUI poskytuje styly pro celé komponenty, jako jsou například tlačítka nebo alert boxy. Také poskytuje ovládání motivu celé webové stránky, díky čemuž je s trohou konfigurace možné jednoduše měnit celkový vzhled stránky.

Lucide for React

Poslední použitou knihovnou je Lucide, knihovna ikon vytvořená pro React. V běžné webové stránce by vložení ikony, jako například koš pro odstranění dopravníku, vyžadovalo stažení souboru s příponou například .svg a jeho následnou úpravu a vložení do stránky.

S Lucide však stačí pouze použít předpřipravené React komponenty, které obsahují většinu často používaných ikon. [27]

2.4.3 Princip komunikace s WebServery

Jak bylo nastíněno v kapitole 2.4.1, způsob, kterým spolu komunikují mobilní aplikace a ESP8266 WebServer, spočívá ve využití webových GET požadavků.

GET požadavek je základní nástroj pro komunikaci mezi klienty a servery webové infrastruktury. Například při zadání URL adresy webové stránky do prohlížeče odešle

prohlížeč GET požadavek na server nacházející se na dané webové adrese. Server tento požadavek zaregistrouje a odpoví zasláním obsahu webové stránky (front-endu).

Přesně tímto způsobem komunikuje i mobilní aplikace s WebServerem běžícím na vývojové desce. Protože je vývojová deska připojena k hotspotu hostovanému mobilním zařízením s aplikací, může mobilní aplikace komunikovat s WebServerem jako s jakýmkoliv jiným serverem, a proto může podávat požadavky na různé adresy. Nastavení vývojové desky navíc umožňuje spustit různé bloky kódu po přijetí těchto GET požadavků.

V JavaScriptu se GET požadavky implementují tímto způsobem: [28]

```

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          const json = await response.json();
9          console.log(json);
10     } catch (error) {
11         console.error(error.message);
12     }
13 }
```

Ukázka kódu 2.3: Základní způsob posílání GET požadavků v JavaScriptu

Níže jsou popsány tři hlavní způsoby využití GET požadavků pro ovládání systému:

Jak dálkově spouštět dopravník

V mobilní aplikaci je možné mít tlačítko, na něž lze kliknout, čímž se odešle GET požadavek na adresu 192.168.0.144/conveyorOn². Po kliknutí na tlačítko vývojová deska zaregistrouje tento požadavek a provede kód definovaný ve třídě ConveyorController (v kapitole 2.3.1). Tento kód nastaví proměnnou, na jejímž základě v další iteraci metody updateState dojde k sepnutí relé na desce plošných spojů, čímž se na ovládací panel frekvenčního měniče nastaví vysoká logická hodnota pro spuštění dopravníku.

Jak získat data o rychlosti dopravníku

Další možností je periodické vysílání GET požadavku funkcí. GET požadavky se zasílají na 192.168.0.144/getData. Vývojová deska tento požadavek zaregistrouje a odešle zpět

²192.168.0.144 je náhodně přiřazená lokální IP adresa mikrokontroleru, která se s každou vývojovou deskou mění.

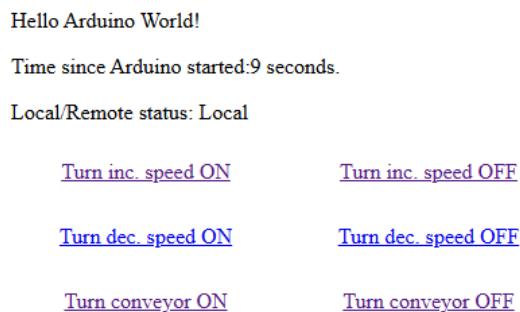
odpověď ve formátu JSON. Tento přijatý soubor obsahuje informace o rychlosti dopravníku (v procentech) a o stavu ovládání (lokální nebo dálkové). Data v tomto formátu lze ve webových aplikacích snadno konvertovat do TypeScript objektu, který lze zobrazit v HTML kódu mobilní aplikace.

Pokud by React nebyl reaktivní (jak bylo vysvětleno v kapitole 2.4.2), pak by se s každým přijetím dat musela celá webová stránka obnovit. S využitím funkce React knihovny lze však aktualizovat pouze rychlostní a ovládací data na stránce.

Jak ovládat vývojovou desku bez mobilní aplikace

Nakonec není ani nutné používat mobilní aplikaci, ale stačí použít webový prohlížeč v mobilním telefonu k ovládání desky. V tom případě lze do URL adresy zadat 192.168.0.144/ bez další cesty. WebServer zde znova zaregistrouje požadavek a tentokrát odešle zpět HTML kód webové stránky, která je nakonfigurována jako hlavní odpověď ve třídě `ConveyorController`.

Tato webová stránka umožňuje ovládat vývojovou desku pomocí tlačítek, jež jsou nakonfigurována k odesílání GET požadavků na příslušné adresy vývojové desky. Tato stránka ovšem není reaktivní, protože se jedná o statickou HTML stránku. Kvůli tomu se aktuální hodnoty rychlosti a stavu ovládání zobrazí pouze po jejím načtení. Na obrázku 2.11 je zobrazen vzhled této webové stránky.



Pin States:

```
locOnOffState: 0
locIncSpeedState: 0
locDecSpeedState: 0
remOnOffState: 0
remIncSpeedState: 0
remDecSpeedState: 0
remoteLocalState: 0
conveyerSpeed: 0
```

Obrázek 2.11: Ukázka stránky pro ovládání vývojové desky bez mobilní aplikace

2.4.4 Implementování GET požadavků s Capacitor

GET požadavky není kvůli konverzi webové aplikace do mobilní pomocí Capacitoru možné posílat běžným způsobem. Je tomu tak, protože v prostředí Android aplikace funkce `fetch` nefunguje stejně jako v prostředí webového prohlížeče. Tento problém je však již vyřešen

komunitou Capacitoru, která pro tento účel vyvinula balíček s názvem CapacitorHttp. Tento balíček zjednodušuje posílání GET požadavků v rámci Capacitor aplikací tím, že poskytuje vlastní metodu GET, která má mírně odlišnou syntaxi upravenou tak, aby fungovala v mobilním prostředí WebView (konverze do mobilní aplikace je popsána v kapitole 2.4.6).

GET požadavky se v aplikaci používají tímto způsobem:

```

1   // pred zacatkem komponentu:
2   import { CapacitorHttp } from "@capacitor/core";
3   import { HttpOptions } from "@capacitor/core/types/core-plugins";
4
5   // uvnitr komponentu se strankou aplikace:
6
7   // Posilani prikazu specifickemu dopravniku
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 // Nastaveni rychleho selhani pokud není ziskana odpoved
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     // Kdyz selze odpoved tak se ihned nastavi dopravnik jako nedostupny
30     setConveyors((prevConveyors) =>
31       prevConveyors.map((conv) =>
32         conv.ip === ip ? { ...conv, isOnline: false } : conv

```

```

33     )
34   );
35 }
```

Ukázka kódu 2.4: Rozšířený způsob posílání GET požadavků pro Capacitor mobilní aplikace

Nejdříve je nutné si importovat `HttpOptions` a `CapacitorHttp` z nainstalované Reactové Capacitor knihovny. Následně pokračuje definice komponentu, který obsahuje celou vstupní stránku aplikace a který je v rámci React aplikace zobrazován. Uvnitř stránky aplikace je definována funkce `sendCommand`.

Funkce `sendCommand` má jako vstupy IP adresu vývojové desky 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 aplikace se poté pomocí funkce `CapacitorHttp.get` pokusí odeslat požadavek. Pokud byl požadavek úspěšně doručen, aplikace se bude pokoušet parsovat odpověď jako JSON. To se pro většinu odpovědí nezdaří, jelikož aplikace v rámci některých odpovědí zasílá i HTML kód, aby bylo možné ji ovládat i přes webový prohlížeč (popsáno v kapitole 2.4.3). Pokud se ale nepodaří parsovat odpověď, nijak to neovlivňuje funkčnost aplikace a je tak přijatelné mít rádeček pro parsování odpovědi jako JSON ve try-catch bloku, který případnou chybu potlačí.

Pokud `sendCommand` nezíská odpověď do 3 sekund, kód aplikace pomocí funkce `fetchWithTimeout` signalizuje chybu, která se zobrazí v uživatelském rozhraní. Toto je obzvlášť důležitá funkce, protože se tato chyba uživateli v aplikaci ukáže, pokud během 3 sekund webový server neodpoví na požadavek. To se může stát hlavně pokud uživatel opustí dosah, ve kterém je webový server schopen komunikovat, a má tak jasnou vizuální indikaci, že by se pro ovládání dopravníku měl přiblížit. Tato implementace také vypíná možnost stisknout tlačítka, která ovládají dopravník, jelikož by na ně vývojová deska nereagovala.

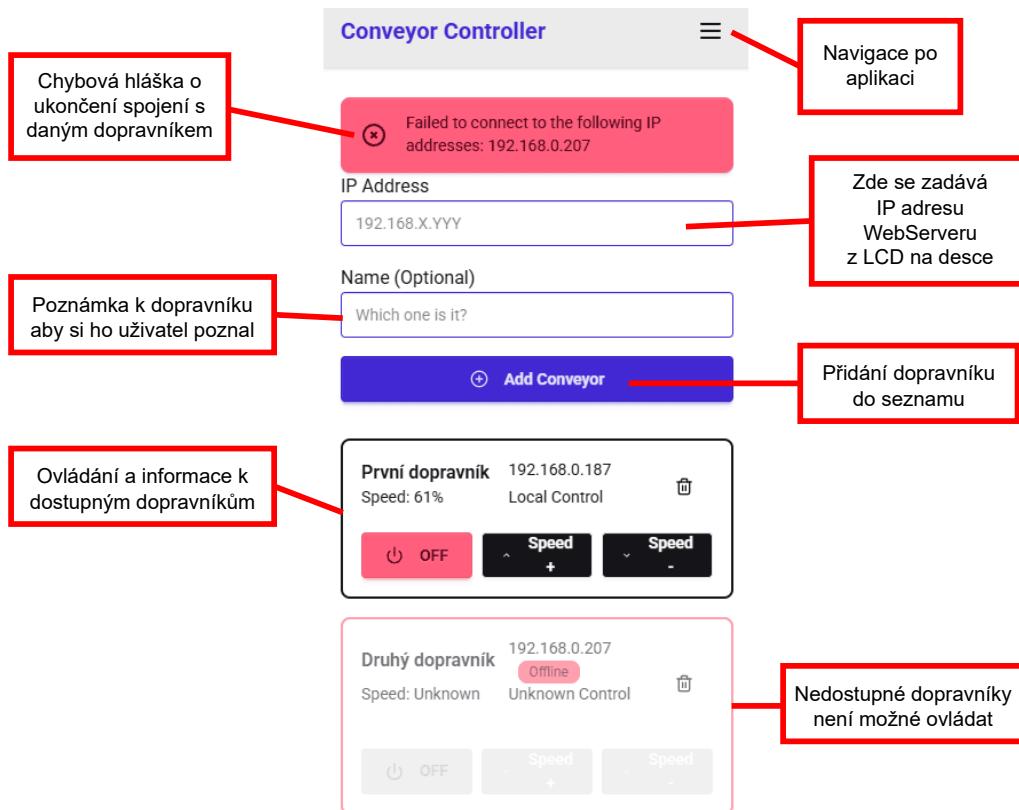
2.4.5 Design aplikace

Tato mobilní aplikace sice byla naprogramována jako webová, ale její design byl od začátku koncipován s ohledem na optimální vzhled při používání na mobilních zařízeních. Jelikož je v dnešní době kladen velký důraz na to, aby i webové stránky vypadaly dobrě na mobilních zařízeních, existují pro tento účel nástroje, jako jsou Chrome Developer Tools.

Chrome Developer Tools je soubor nástrojů pro webové vývojáře dostupný pro každého uživatele prohlížeče Google Chrome. V rámci těchto nástrojů je možné změnit vzhled webové stránky na jakékoli rozlišení za účelem testování vzhledu na tablettech a mobilních zařízeních různých velikostí. Chrome Developer Tools však kromě této funkce obsahuje i velké množství dalších užitečných nástrojů pro webové vývojáře, jako například konzole (kde se zobrazují chybové zprávy), nástroje pro analýzu rychlosti aplikace, náhled celého HTML kódu a mnoho dalších.

Všechny barvy aplikace jsou definovány pomocí klíčových slov v rámci správy motivů poskytované knihovnou DaisyUI (rozšíření, krátce představeného v kapitole 2.4.2). Tato klíčová slova jsou například `base`, `primary`, `secondary`, `accent` a další. Nastavení aplikace tímto způsobem dává možnost parametrizace barev, jelikož se určuje spíše účel barvy než její přímá hodnota. Díky tomuto nastavení je možné měnit motiv stránky na jeden z 30 přednastavených motivů od DaisyUI, nebo si nastavit vlastní barvy. Navíc je možné nastavit různé motivy pro světlé i temné rozhraní mobilního zařízení. V [29] lze vidět, jaká všechna nastavení DaisyUI poskytuje v rámci správy motivů.

V obrázku 2.12 je zobrazen vzhled hlavní stránky aplikace pro světlé rozhraní telefonu.



Obrázek 2.12: Popis designu hlavní stránky aplikace

Tento obrázek aplikace byl pořízen na počítači v rozhraní Chrome Developer Tools. Na hlavní stránce aplikace lze nahoře vidět hlavičku, která obsahuje tlačítko navigace. Po stisknutí tohoto tlačítka se zobrazí odkazy na další stránky obsažené v aplikaci, jak bylo popsáno v kapitole 2.4.1. Lze také vidět chybovou zprávu, která se však zobrazí pouze v případě, že se nepodařilo spojit s některým webovým serverem.

Uprostřed stránky je následně vidět část aplikace určenou k přidávání IP adres vývojových desek, čímž se v dolní části aplikace vytvoří rozhraní pro jejich ovládání. Zde je možné zadat IP adresu desky tak, jak je napsaná na LCD displeji fyzické desky připojené k dopravníku. Dále je zde poznámka, která je v aplikaci pouze pro identifikaci dopravníku a na funkci nemá žádný vliv. Po stisknutí tlačítka pro přidání dopravníku se přidají informace o dopravníku do matice objektů obsahující všechny dopravníky.

V dolní části aplikace je pak oblast obsahující tlačítka pro ovládání takto přidaných dopravníků. Jak bylo popsáno v kapitole 2.4.3, tato tlačítka jsou nastavena pro posílání GET požadavků na adresy vývojové desky. Na jakou adresu se požadavek odešle, záleží na tom, zda již bylo tlačítko stisknuto. Například pokud dopravník nebyl ještě spuštěn pomocí aplikace, tlačítka ON/OFF bude ve stavu OFF, bude mít červenou barvu a bude posílat GET požadavek na adresu "/conveyorOn". Po stisknutí se okamžitě změní jeho stav na ON, bude mít zelenou barvu a bude posílat GET požadavek na adresu "/conveyorOff". V aplikaci je pro tyto účely implementována lokální paměť a při zavření se ukládá, které dopravníky byly dostupné a jak byly ovládány.

2.4.6 Konvertování webové aplikace do mobilní aplikace

Aplikace byla konvertována v základní verzi programu **Android Studio** verze 2024.3.1 a pro konverzi nebylo potřeba instalovat žádné další balíčky.

Způsob konverze React aplikace do její mobilní podoby spočívá v zapouzdření webového obsahu do nativní komponenty. V případě zařízení s Androidem se k tomu využívá **WebView**, který zobrazuje webový obsah v simulovaném webovém prohlížeči, a díky tomu je možné některé funkce webové aplikace přímo integrovat do mobilní verze, zatímco některé funkce (jako třeba zaslání GET požadavků) je možné implementovat pomocí knihoven Capacitoru, jelikož v původní syntaxi ve WebView nefungují. [30]

Před samotným zapouzdřením aplikace je potřeba aplikaci připravit do tzv. "produkční verze". Pro vytvoření této verze stačí v moderních JavaScriptových knihovnách, jako je React, zkompilovat zdrojový kód. Výsledkem této konverze jsou statické soubory HTML, CSS a JS.

Aby bylo možné úspěšně komunikovat s webovými servery, je nutné splnit některé bezpečnostní aspekty v případech, kdy web komunikuje se serverem, specificky mechanismus **Cross-Origin Resource Sharing** neboli CORS. To je protokol, který pro servery z bezpečnostních důvodů omezuje, která zařízení mají právo na tento server zasílat požadavky, jako je GET požadavek. Mobilní aplikace je jiné zařízení než webový server a je na jiné lokální IP adresu, a tak je potřeba tento bezpečnostní aspekt vyřešit, jinak spolu nebudou moci tato dvě zařízení komunikovat. To bylo nakonec vyřešeno tak, že jsou na každé webové adrese webového serveru nastaveny CORS hlavičky v HTML kódu tak, aby přijímal jakékoli GET požadavky z jakékoli IP adresy. Toto rozhodnutí bylo provedeno, jelikož mobilní zařízení může mít jakoukoli IP adresu na lokální síti. Tímto způsobem je ale komunikace stále bezpečná, jelikož webový server je dostupný pouze lokálně, a tak by na něj mohl být útok proveden pouze v případě, že by byl útočník také připojen k hotspotu mobilního zařízení. [31]

Dalším důležitým aspektem specifickým pro platformu Android je výchozí konfigurace síťové bezpečnosti. U novějších verzí Androidu už operační systém standardně zakazuje komunikaci pomocí nešifrovaného protokolu HTTP a preferuje zabezpečený protokol HTTPS. Jelikož webový server v této implementaci využívá pro komunikaci nešifrovaný protokol HTTP, bylo nutné tuto restriku obejít. Povolení komunikace přes HTTP bylo provedeno pro všechny lokální domény modifikací konfiguračního souboru **AndroidManifest.xml** mobilní aplikace, který je generován při konverzi pomocí Capacitoru. Zavedení protokolu HTTPS na webovém serveru by v kontextu lokální sítě nepřineslo žádné bezpečnostní benefity. [32]

Výsledkem konverze aplikace je spustitelný soubor ve formátu **.apk**, který lze nainstalovat na mobilních zařízeních s Androidem. Konverzi je možné provést bez certifikace pro distribuci přes Google Play Store, ale v takovém případě se na zařízení zobrazuje jako "neověřená". V rámci testování ale aplikace prošla úspěšně standardní virovou kontrolou Google Play, což potvrdilo, že během procesu konverze v aplikaci nepřibyl žádný virus. Vzhledem k tomu, že je aplikace určená pro interní použití v rámci společnosti Honeywell a nebude veřejně distribuována, není zapotřebí provádět kroky nezbytné pro publikaci v Google Play Storu.

Postup konvertování webové aplikace

Pro konvertování je potřeba mít nainstalované Android Studio a Node.js. Dále musí být inicializována React aplikace a v ní nainstalovaný a nastavený Capacitor. Složka webové aplikace by měla být vzhledem k Android Studiu mimo složky s českou diakritikou a mimo složky synchronizované pomocí cloudových aplikací.

Specifický postup konvertování webové aplikace do mobilní je následující:

1. Začíná se v nejnovější verzi React webové aplikace.
2. Sestavení do "produkční verze webové aplikace" se provádí pomocí příkazu `npm run build`.
3. Dále je potřeba aplikaci synchronizovat s Capacitorem pomocí příkazu `npx cap sync`.
4. Nyní je potřeba otevřít složku jménem "android", která byla vytvořena, pomocí Android Studia. To je možné udělat rovnou z konzole pomocí příkazu `npx cap open android`.
5. V Android Studiu je možné aplikaci sestavit do "produkční verze mobilní aplikace" pomocí volby "Build".
6. Android Studio vygeneruje složku se soubory aplikace, ve které je obsažen soubor `main.apk`.

Na konci tohoto procesu je možné dostupný soubor `main.apk` poslat na zařízení s Androidem a tam nainstalovat. Pro konverzi není potřeba žádné další nastavování ani podpůrné balíčky.

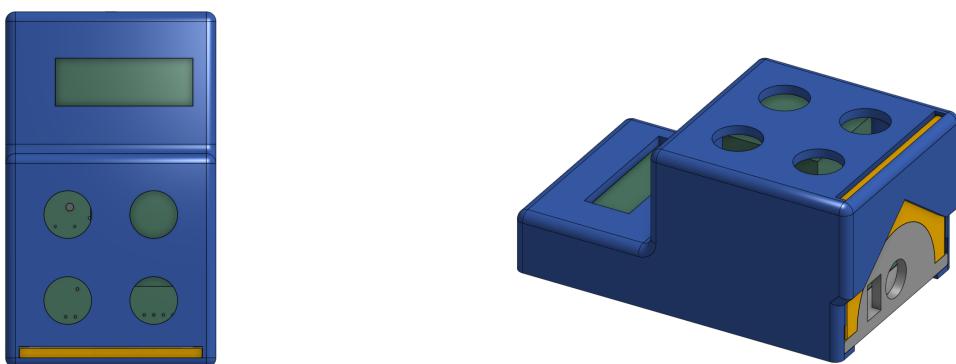
2.5 Vytvoření schránky pro desku

Myšlenka při návrhu schránky pro desku byla taková, aby bylo možné ji vytisknout v běžných podmírkách pomocí technologie 3D tisku - specificky byla zamýšlena pro tiskárnu Bambu Lab A1 Mini, která má tiskový objem 18x18x18 cm. Hlavním důvodem, proč byl 3D tisk zvolenou technologií, byl malý počet vytiskných schránek (předpokládá se 5 kusů) a nenáročné podmínky provozu. Navíc je 3D tisk pro schránky na osazené desky plošných spojů levné řešení, které dosahuje dostatečné kvality provedení. Nároky na schránku jsou základní - je důležité mít možnost ji rozdělat, pro možnost údržby desky, ale jinak nemá zvláštní nároky, jelikož bude systém využívaný pár týdnů v roce při kontrole kvality dopravníků.

Pro návrh schránky byl vybrán cloudový CAD software Onshape. Do tohoto prostředí byl importován 3D model desky plošných spojů ve formátu `.stl`, který byl vygenerován v prostředí KiCAD pomocí integrovaného 3D prohlížeče. Na základě tohoto referenčního modelu bylo přistoupeno k modelování schránky na míru. Koncepcně byla schránka navržena jako dvoudílná s cílem eliminovat použití šroubových spojů pro snadnou manipulaci se systémem a možnost rychlého prototypování a úprav. Zároveň byl ale kladen důraz na spolehlivé uzavření schránky během provozu, kdy není vyžadován přístup k vnitřním komponentám.

Toho bylo dosaženo použitím dvou komplementárních částí. Spodní část schránky je konstruována tak, aby bylo možné do ní zasunout osazenou desku plošných spojů, díky čemuž je deska dobře zafixována uvnitř schránky. V této části je navíc otvor pro kabely, které se připojují na ovládací panel frekvenčního měniče a kterými je deska napájena. Spodní část je během používání zasunuta do horní části schránky. Horní část je dostatečně velká, aby měla dostatek prostoru pro umístění tlačítek, LCD displeje a také obsahuje otvor pro našroubování externí WiFi antény dostatečně blízko vývojové desky. Posledním prvkem návrhu schránky je zajišťovací prvek (západka), která slouží pro fixaci spodní části v horní části a zabraňuje tak nechtěnému vysunutí. Tato západka je držena na místě pomocí tření s horní částí schránky.

Celou schránku lze vidět na obrázku 2.13



(a) Pohled shora na schránku

(b) Pohled z boku na schránku

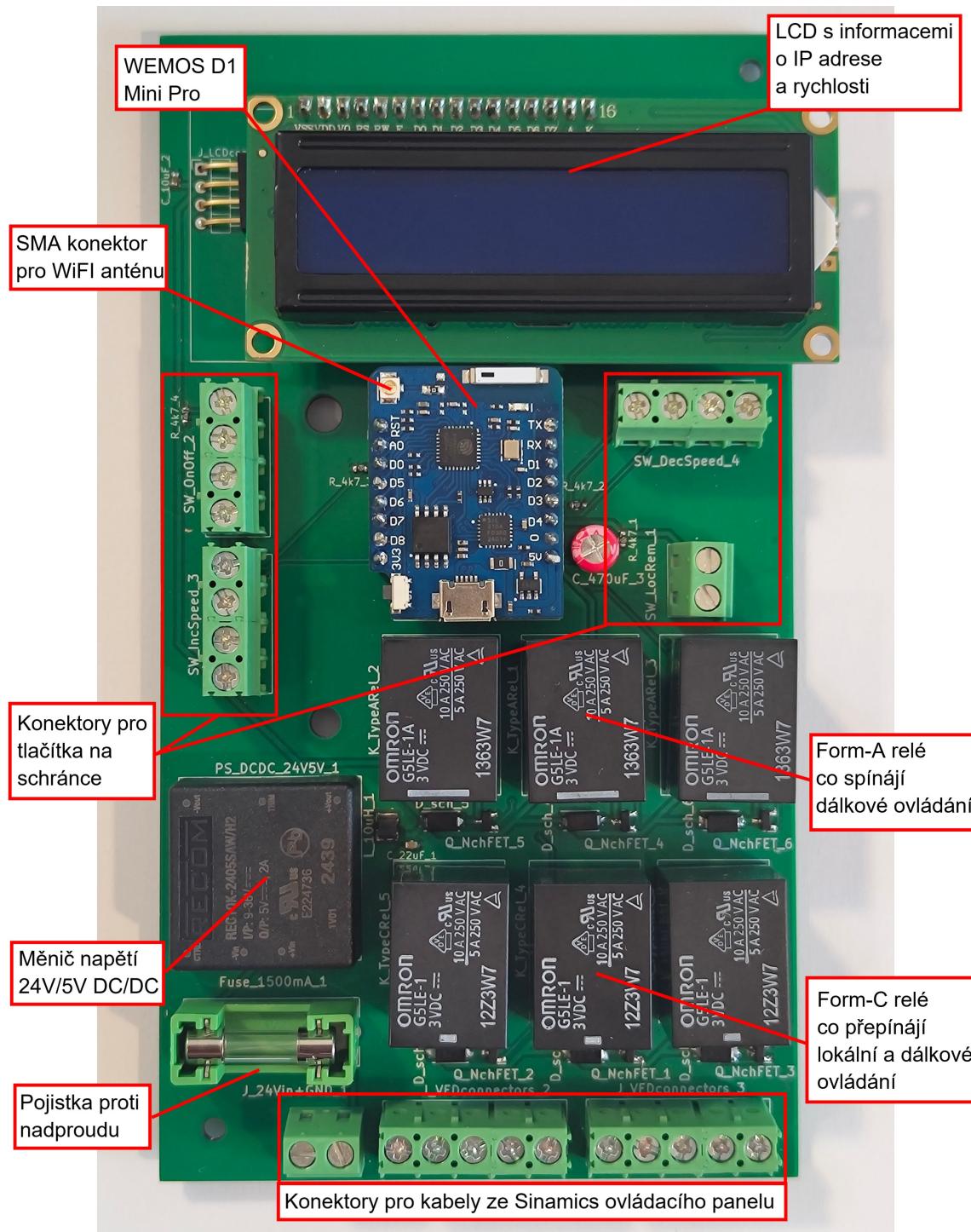
Obrázek 2.13: Model schránky pro desku plošných spojů

Po dokončení modelů v prostředí Onshape byly tiskové soubory exportovány do softwaru Bambu Studio a následně odeslány k tisku na zmíněnou tiskárnu. Pro tisk byl zvolen bílý PETG filament značky SUNLU. Tento materiál byl preferován pro svou vyšší mechanickou odolnost ve srovnání s PLA, přičemž jeho zpracování nevyžaduje náročné tiskové podmínky ani hardware. Během tisku byl tento filament vysoušen kvůli známým problémům s vlhkostí uvnitř PETG filamentu.

2.6 Kompletace řešení

Během kompletace systému byly nejdříve připájeny veškeré osazované součástky na desku plošných spojů. Následně došlo k připojení dvou komponent, které jsou navržené jako vyjmíatelné moduly – vývojová deska WEMOS D1 Mini Pro a LCD displej. Tyto součástky byly navrženy jako snadno vyjmíatelné primárně kvůli tomu, že to nejsou standardní zapouzdřené integrované obvody určené přímo k pájení. Jelikož jsou tyto součástky komplexnější, je dobré mít možnost je jednodušeji diagnostikovat a vyměnit, protože můžou vyžadovat složitější nastavení pro optimální funkčnost. Kromě toho je při prototypování vývojová deska často vyjmána kvůli nahrávání firmwaru a ladění kódu během vývoje, zatímco modul LCD displeje disponuje na spodní straně trimrem, který mění kontrast, a ten se typicky také nastavuje až po osazení modulu na desce.

Finální podobu desky plošných spojů lze vidět na obrázku 2.14.



Obrázek 2.14: Finální podoba desky plošných spojů

Po osazení komponent na DPS je dalším krokem připojení kabelů, které komunikují s ovládacím panelem frekvenčního měniče a napájí vývojovou desku. Tyto kably se připojují do šroubovacích svorkovnic na spodní straně desky plošných spojů. Následně byla tlačítka umístěna na schránku pro desku a k témuž tlačítkům byly připojeny kably do šroubovacích konektorů, které jsou umístěny uprostřed desky. Poté je možné upevnit desku ve schránce.

V této fázi je deska připravená pro připojení k dopravníkovému systému, jehož prin-

cip je zobrazen na začátku návrhové kapitoly na obrázku 2.1. Napájení desky je možné realizovat buď z ovládacího panelu frekvenčního měniče, což je doporučený způsob napájení kvůli jednoduchosti, vzhledem k tomu, že jsou veškeré ostatní kabely také připojené k frekvenčnímu měniči. Systém ale bude fungovat i v případě, že nebude napájen z frekvenčního měniče – desku je možné napojit na libovolný externí zdroj poskytující stejnosměrné napětí v rozsahu $9\text{--}48V$. Špičková hodnota proudu, kterou je možné v extrémním případě na DPS zaznamenat, je asi $1,3A$ při $5V$ napájení, takže na vstupu musí zdroj poskytovat při napětí $24V$ kolem $300mA$ proudu.

3 Ověření návrhu

Tato kapitola se věnuje ověření funkčnosti návrhu a praktické implementace navrženého systému pro ovládání dopravníkové linky. Cílem provedených testů bylo potvrdit správnou činnost jednotlivých komponent i celkové integrace systému do prostředí typické dopravníkové linky a otestovat chování v reálném provozu.

Způsob testování a prostředí

Veškeré testování bylo provedeno v Brněnské hale společnosti Honeywell (viz. obrázek 3.1). V té je rozmístěno několik různých plně funkčních dopravníkových systémů. Tato hala existuje primárně aby bylo možné ukázat potenciálním zákazníkům společnosti Honeywell jaké produkty firma nabízí, ale také velmi často slouží jako testovací prostředí pro inovace, jako je tento systém na ovládání dopravníků.

V době, kdy byla v hale testovaná funkčnost celého systému byla přibližně 80 metrů daleko v jiné části haly spuštěná jiná dopravníková linka, ale vzhledem k tomu, že vzdálenost dosahu WiFi signálu byla v rámci přijatelných mezí, lze předpokládat, že signál nebyl druhou linkou zarušený. Je ale potřebné zdůraznit, že charakteristiky šíření bezdrátového signálu a potenciální míra rušení v konkrétní lokalitě zákazníka se mohou významně lišit v závislosti na uspořádání dopravníků, přítomnosti překážek, hustotě instalovaných zařízení a dalších lokálních faktorech co by mohly rušit radiový signál.



Obrázek 3.1: Ukázka Brněnské haly pro testování dopravníků společnosti Honeywell [33]

Způsob, jakým byl systém prvně otestován je skrz konfiguraci ovládacího panelu frekvenčního měniče podle instrukcí specifikovaných na stránce "/setup" dostupné uvnitř mobilní aplikace. Nejdříve byla rozpojena výkonová část frekvenčního měniče a poté byl nastavený. Po dokončení softwarové konfigurace byla deska připojena na digitální vstupy do ovládacího panelu frekvenčního měniče včetně stejnosměrného 24V napájení z výstupního napájecího portu ovládacího panelu.

Po otestování správnosti zapojení a konfigurace pomocí lokálního zapojení byla provedena zkouška s vypnutou výkonovou částí frekvenčního měniče a následně i se zapnutou výkonovou částí frekvenčního měniče, kde se dopravník začal pohybovat, jak bylo předpokládáno. Při uvádění dopravníku do provozu se zapnutou výkonovou částí je potřebné zajistit, aby v systému nebyla načtená hodnota approximace rychlosti dopravníku jiná než 0. Zanedbání tohoto kroku vede k tomu, že bude systém ukazovat v mobilní aplikaci i na LCD displeji posunutou hodnotu rychlosti.

3.1 Ověření lokálního ovládání

Funkčnost lokálního ovládání byla ověřena tím způsobem, že se nejdříve otestovalo, že je možné ovládat dopravník s vypnutou deskou a poté se deska připojila a znova se testovalo jestli je možné dopravník ovládat s připojenou deskou, která je nastavená na lokální režim ovládání.

Při připojení systému na dopravník bez připojení napájecího kabelu bylo vidět, že LCD displej nefunguje a v mobilní aplikaci není dostupná předchozí IP adresu. Dopravník bylo stále možné ovládat, jelikož je deska plošných spojů provedena tím způsobem, že když není napájena, je ovládána lokálně.

Při připojení systému na dopravník s připojením napájecího kabelu se ihned spustil LCD displej a bylo vidět, že se vývojová deska snaží připojit na hotspot. Po připojení bylo možné vidět IP adresu vývojové desky i v mobilní aplikaci. Při nastavení přepínače na polohu ve kterém je deska lokálně ovládaná bylo stále možné ovládat dopravník pomocí tlačítka na schránce. Potvrdilo se tedy, že v tomto módu ovládání deska neposílá žádné napětí na sepnutí MOSFET transistorů co spínají relé a tak systém stále funguje plně analogově. Při tomto módu zapojení je ale dostupná informace o rychlosti dopravníku jak na LCD displeji tak i v mobilní aplikaci.

3.2 Ověření mobilní aplikace

Po úspěšném otestování analogového ovládání desky bylo přistoupeno k testování mobilní aplikace. Mobilní aplikace se už částečně otestovala při předchozím testu, kdy bylo vidět, že pokud je vývojová deska přidaná se správnou IP adresou do aplikace, lze vidět rychlosť a způsob ovládání dopravníku i pokud systém není přepnutý do dálkového ovládání.

Pro další testování byl systém přepnutý do stavu dálkového ovládání pomocí přepínače který je umístěný na schránce systému. Při přepnutí do stavu dálkového ovládání dopravník ihned začal zpomalovat, protože bylo přivedené napětí na MOSFET tranzistor, který ovládá relé pro přepínání mezi typy ovládání - rozpojil se tedy obvod, který přivádí vysokou digitální hodnotu na vstup do ovládacího panelu, který dopravník udržoval v zapnutém stavu. Následně bylo potvrzeno, že dopravník lze ovládat pomocí mobilní aplikace, která úspěšně posíala GET požadavky na WebServer a tak bylo vidět že vývojová deska přepíná relé, které na desce plošných spojů spíná digitální vstupy ovládacího

panelu.

Při přepnutí do dálkového stavu ovládání bylo také v mobilní aplikaci vidět, že se typ ovládání změnil na variantu "Remote".

3.3 Spolehlivost dálkové komunikace

Při zapojení systému s dálkovým ovládáním bylo možné otestovat zároveň i vzdálenost komunikace, které je možné se systémem dosáhnout. Pro otestování dosažitelné vzdálenosti bylo zvoleno jít směrem pryč od druhé spuštěné dopravníkové linky, aby bylo méně pravděpodobné, že se WiFi signál zaruší. V rámci testování byl také celý systém umístěn tak, aby byla zajištěna přímá viditelnost na anténu vycházející ze schránky. Použité mobilní zařízení bylo během testování Nothing Phone 1. verze. Tímto způsobem byl dosažen dosah WiFi komunikace kolem 35 metrů.

Tento dosah není tak velký, jak bylo předpokládáno, jelikož se předpokládal WiFi dosah až 90 metrů. Dosah systému je menší, jelikož je omezený mobilním zařízením – dosah hotspotů není u běžných Android zařízení parametr, který by se výrobci snažili optimalizovat na maximální hodnotu. Tento dosah je ale pro systém stále přijatelný. Dálkové ovládání je v systému navrženo například pro případy, kdy budou mechanické zkoušky prováděny na dopravních nainstalovaných pod stropem skladové haly. V těchto případech se výška dopravníků může pohybovat kolem 10 metrů, a tak by měl být dosah 35 metrů dostatečný i pro tyto případy.

Do maximální vzdálenosti nebyly zaznamenány žádné problémy s komunikací. Vývojová deska reagovala okamžitě. Tohle se dalo předpokládat, vzhledem k tomu, že vývojová deska odpovídá na WebServerové požadavky velmi často a jinak není zatížená dlouhými výpočty, které by výrazně zvyšovaly dobu prodlení mezi požadavkem a zpracováním.

3.4 Ověření zapojení více desek

V rámci tohoto testu byly nakonfigurovány a zapojeny dvě desky dle standardního postupu s tím, že každá deska byla zapojena k samostatnému dopravníku. Dopravníky byly zvoleny tak, aby od sebe byly frekvenční měniče vzdálené v rámci několika metrů a aby bylo možné z jednoho místa vidět antény obou schránek. Obě schránky byly pomocí přepínačů nastaveny na dálkový režim ovládání a IP adresy obou WebServerů byly přidány do mobilní aplikace.

Následně bylo nejdříve s vypnutou výkonovou částí frekvenčního měniče potvrzeno, že aplikace posílá správné GET požadavky na správné WebServery a tak je možné tímto způsobem ovládat oba dopravníky z jedné mobilní aplikace. Toto bylo potvrzeno ještě jednou se zapnutou výkonovou částí frekvenčního měniče s tím, že byly na dopravníky položeny testovací balíky, aby bylo možné vidět, kdy se dopravníky pohybují.

I v případě zapojení více desek komunikovaly WebServery promptně a bez jakýchkoli problémů s komunikací.

3.5 Posouzení z hlediska bezpečnosti

Navržený systém už ze své podstaty umožňuje dálkové ovládání dopravníkových linek, a to i na delší časové úseky – například při zahořovacích testech, kdy se ověruje, že je dopravník schopný nepřetržitého provozu. Takovýto způsob ovládání může představovat inherentní riziko v případě ztráty komunikačního spojení mezi mobilní aplikací a vývojovou deskou

v kritickém okamžiku, kdy by operátor nemohl dopravník bezprostředně zastavit.

Z tohoto důvodu byla zvažována implementace mechanismu periodické kontroly aktivního spojení (například periodický „ping“ dotaz), který by inicioval zastavení dopravníku, pokud by vývojová deska tento požadavek neobdržela ve stanoveném intervalu. Od implementace takového mechanismu však bylo upuštěno, jelikož by příliš negativně ovlivňoval používání aplikace uživateli. Během provádění těchto dynamických zkoušek uživatelé hledají chyby instalace dopravníkového systému a tyto chyby se běžně zapisují do dokumentace v jiných aplikacích. Dá se tedy předpokládat, že aplikace běžně nebude na mobilním zařízení stále aktivní.

Toto rozhodnutí je ovšem podpořeno tím, že navržený systém se spoléhá na již existující bezpečnostní prvky instalované na samotných dopravníkových linkách a přebírá je. Dopravníkové linky mají hned po instalaci standardní bezpečnostní prvky, jako jsou tlačítko nouzového zastavení (E-STOP) a bezpečnostní lanka po celé délce dopravníku.

V kontextu bezpečného provozu je rovněž důležité adekvátní proškolení uživatelů systému. Uživatel musí být seznámen s postupy bezpečného používání systému – hlavně s nutností deaktivace výkonové části frekvenčního měniče pomocí příslušného vypínače před jakoukoliv manipulací s ovládacím panelem. Uživatel musí být dále informován o správném postupu konfigurace systému a o způsobu řešení běžných provozních problémů. Tyto informace jsou obsaženy v mobilní aplikaci na stránkách Setup a Help.

Závěr (1 strana)

Hlavním souhrnným cílem této bakalářské práce byl **navrhnut systém pro dálkové ovládání dopravníků** pro společnost Honeywell – specificky pro použití při dynamických kontrolách instalace dopravníků. Primárním cílem bylo vytvořit jednoduché a bezpečné řešení, které je možné ovládat i z bezprostřední blízkosti, ale i prostřednictvím mobilního zařízení pomocí WiFi komunikace.

V úvodní části práce byly položeny teoretické základy nezbytné pro pochopení návrhu systému. Nejprve bylo prozkoumáno, jakým způsobem se řídí rychlosť dopravníků a jak frekvenční měniče fungují. Pozornost byla také věnována analýze specifického frekvenčního měniče **Sinamics G120D**, pro který byl tento návrh primárně určen. Důležitou součástí této analýzy byly možnosti nastavení ovládacího panelu, díky kterým celý systém komunikuje s frekvenčním měničem. V rámci rešerše byla také prozkoumána možnost integrace mikrokontroléru do návrhu, k čemuž byla zvolena vývojová deska **WEMOS D1 Mini Pro** s mikrokontrolérem **ESP8266EX**. Tato deska poskytuje dostatečné parametry s integrovanou WiFi komunikací a možností připojit externí WiFi anténu. Společně s touto deskou bylo prozkoumáno také, jakým způsobem je možné desku programovat prostřednictvím Arduino frameworku pro ESP8266.

Druhá kapitola se věnovala návrhu celého systému – nejprve z hlediska architektury a požadavků, a následně návrhu jednotlivých dílčích částí. Tři rovnocenné části systému jsou:

- **Uživatelská mobilní aplikace**, pomocí které je jednoduše možné ovládat a nastavit dopravník.
- **Firmware vývojové desky**, který pomocí ESP8266 WebServeru interpretuje instrukce z mobilní aplikace.
- **Elektronika**, která pomocí navržené desky plošných spojů převádí výstupy z vývojové desky na instrukce pro ovládací panel frekvenčního měniče.

Na začátku byl v rámci návrhové kapitoly popsán proces tvorby obvodu pro komunikaci s ovládacím panelem frekvenčního měniče, včetně některých zajímavých problémů, které při tvorbě nastaly. Následně byl hardware integrován do desky plošných spojů. V části zabývající se firmwarem vývojové desky byl popsán objektově orientovaný přístup při programování a jeho výhody. Následně se tato část zaměřila na stavový diagram logiky systému a approximaci rychlosti dopravníku. Po popisu firmwaru se tato kapitola věnovala sekci týkající se vývoje mobilní aplikace. Tam byla nejprve popsána architektura aplikace a následně byla do hloubky rozebrána komunikace aplikace a WebServeru. Pozornost byla také věnována designu mobilní aplikace. Na konci návrhové části bylo věnováno místo návrhu schránky pro desku plošných spojů a postupu kompletace hardwaru.

Závěrečná kapitola se zabývala ověřením navrženého systému. To bylo provedeno v brněnské hale společnosti Honeywell – bylo tedy možné systém otestovat v prostředí,

které je velmi podobné halám zákazníků, kde bude systém skutečně používán. Při testování bylo potvrzeno, že je systém možné ovládat jak z lokálního ovládání umístěného na schránce, tak i v režimu dálkového řízení pomocí mobilní aplikace. Testována byla také spolehlivost bezdrátové komunikace, přičemž byl naměřen uspokojivý dosah 35 metrů a bylo potvrzeno, že je možné ovládat více jednotek pomocí jedné mobilní aplikace. Zhodnocena byla také bezpečnost navrženého systému, kde bylo vyhodnoceno, že systém spoléhá na stávající, již nainstalované bezpečnostní prvky dopravníkových linek. Testy potvrdily správnou implementaci a funkčnost navrženého systému v podmírkách, které se blíží reálnému provozu.

Závěrem lze říci, že byly splněny všechny cíle stanovené pro tuto bakalářskou práci, jelikož se týkaly návrhu tří hlavních částí tvořících systém (Hardware, Firmware a Software) a následného otestování funkčnosti navrženého systému. Díky této bakalářské práci tedy vznikl systém, který může inženýrům ve společnosti Honeywell usnadnit dynamické testování dopravníků.

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] *HTML: HyperText Markup Language / MDN*. 2025. Dostupné také z: <https://developer.mozilla.org/en-US/docs/Web/HTML>.
- [21] *CSS: Cascading Style Sheets / MDN*. 2025. Dostupné také z: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [22] *JavaScript / MDN*. 2025. Dostupné také z: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [23] *React*. 2025. Dostupné také z: <https://react.dev/>.
- [24] *TypeScript: JavaScript With Syntax For Types*. 2025. Dostupné také z: <https://www.typescriptlang.org/>.

- [25] *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.* 2025. Dostupné také z: <https://v2.tailwindcss.com/>.
- [26] *daisyUI — Tailwind CSS Components (version 5 update is here).* 2025. Dostupné také z: <https://daisyui.com/>.
- [27] *Lucide React / Lucide.* 2025. Dostupné také z: <https://lucide.dev/guide/packages/lucide-react>.
- [28] *GET - HTTP / MDN.* 2025. Dostupné také z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods/GET>.
- [29] *daisyUI and Tailwind CSS theme generator — Tailwind CSS Components (version 5 update is here).* 2025. Dostupné také z: <https://daisyui.com/theme-generator>.
- [30] *Capacitor Blog - How Capacitor Works.* 2025. Dostupné také z: <https://capacitorjs.jp/blog/how-capacitor-works>.
- [31] *Cross-Origin Resource Sharing (CORS) - HTTP / MDN.* 2025. Dostupné také z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS>.
- [32] *Android – Cleartext HTTP Traffic Not Permitted / GeeksforGeeks.* 2025. Dostupné také z: <https://www.geeksforgeeks.org/android-cleartext-http-traffic-not-permitted/>.
- [33] 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	Ukázka stránky pro ovládání vývojové desky bez mobilní aplikace	43
2.12	Popis designu hlavní stránky aplikace	46
2.13	Model schránky pro desku plošných spojů	49
2.14	Finální podoba desky plošných spojů	50
3.1	Ukázka Brněnské haly pro testování dopravníků společnosti Honeywell [33]	52

Seznam tabulek

Seznam příloh