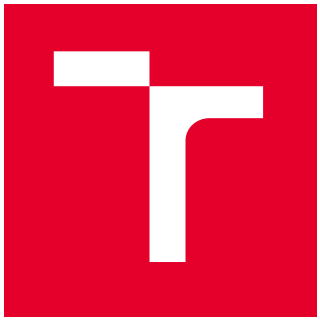


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

I4.0 ŘÍDICÍ SYSTÉM 3D TISKÁRNY PRO TISKOVOU FARMU

I4.0 3D PRINTER CONTROL SYSTEM FOR THE PRINTER FARM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Roman Štrobl**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Ondřej Baštán**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Roman Štrobl ***ID:*** 211186

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

I4.0 řídicí systém 3D tiskárny pro tiskovou farmu

POKYNY PRO VYPRACOVÁNÍ:

Práce se zabývá návrhem řídicího systému 3D tiskárny, umožňujícího její činnost v autonomním režimu.

1. Seznamte se s koncepty průmyslu 4.0
2. Definujte požadavky na řídicí systém
3. Vyberte vhodnou HW platformu pro systém
4. Navrhněte strukturu řídicího systému
5. Implementujte řídicí systém
6. Řešení otestujte.
7. Zhodnoťte dosažené výsledky.

DOPORUČENÁ LITERATURA:

An Industry 4.0 Testbed (Self-Acting Barman): Principles and Design (Kaczmarczyk, 2018)

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 23.5.2022

Vedoucí práce: Ing. Ondřej Baštán

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zabývá vytvořením autonomního řídicího systému, který bude komunikovat s 3D tiskárnou a MES systémem. Jako HW platforma pro řídicí systém byl zvolen jednodeskový počítač Raspberry Pi. Systém byl napsán v programovacím jazyku Python, který podporuje moderní programovací paradigmata. V práci je popsána struktura systému dle požadavků, které byly nadefinovány a programové řešení jednotlivých modulů. Závěr práce je věnován testování systému a zhodnocení dosažených výsledků.

KLÍČOVÁ SLOVA

Průmysl 4.0, 3D tisk, Řídicí systém, Python

ABSTRACT

The bachelor thesis deals with the creation of an autonomous control system that will communicate with a 3D printer and MES system. The Raspberry Pi single board computer was chosen as the HW platform for the control system. The system was written in the Python programming language, which supports modern programming paradigms. The work describes the structure of the system according to the requirements that were defined and the software solution of individual modules. The conclusion of the work is devoted to testing the system and evaluating the achieved results.

KEYWORDS

Industry 4.0, 3D print, Control system, Python

ŠTROBL, Roman. *I4.0 řídicí systém 3D tiskárny pro tiskovou farmu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2022, 56 s. Bakalářská práce. Vedoucí práce: Ing. Ondřej Baštán

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Roman Štrobl
VUT ID autora:	211186
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	I4.0 řídicí systém 3D tiskárny pro tiskovou farmu

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Ondřeji Baštánovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Žijeme v době, kdy průmysl zažívá svou čtvrtou revoluci, která bude mít dopad na celou lidskou společnost. Upustí se od repetitivní práce, kterou za nás budou dělat roboti a vytvoří se nové pracovní pozice, které budou využívat kreativního myšlení člověka.

Tato bakalářská práce se věnuje návrhu řídicího systému pro 3D tiskárny a jeho použití v tiskové farmě. Téma jsem si zvolil z důvodu, že myšlenka průmyslu 4.0 je velmi zajímavá a vidím v ní velký potenciál posunou vývoj lidské společnosti velkou rychlostí kupředu.

V práci se budu věnovat průmyslu 4.0, kde popíši její ideu a základní principy. Poté krátce představím 3D tisk, kde se pokusím popsat možnosti, které tato technologie nabízí.

V další části se budu zabývat návrhem řídicího systému, kde si nadefinuji požadavky, které od řídícího systému očekávám. Poté se zamyslím nad tím, jak bude řídící systém komunikovat s MES (Manufacturing Execution System) a posílat G-kód (programovací jazyk pro 3D tisk) do 3D tiskárny.

Poté budu vybírat HW platformu (Hardware) na které řídicí systém bude spuštěný. To zahrnuje i výběr vhodného programovacího jazyka, ve kterém řídicí systém bude napsán.

V další kapitole se budu zabývat návrhem struktury řídícího systému. ve které popíšu, jak systém bude fungovat a navrhnu si propojení jednotlivých sub-systému, aby jednotlivé moduly nebyly na sebe příliš závislé. Na tuto kapitolu naváži popisem implementace řídicího systému, kde vysvětlím jak jednotlivé části programu fungují.

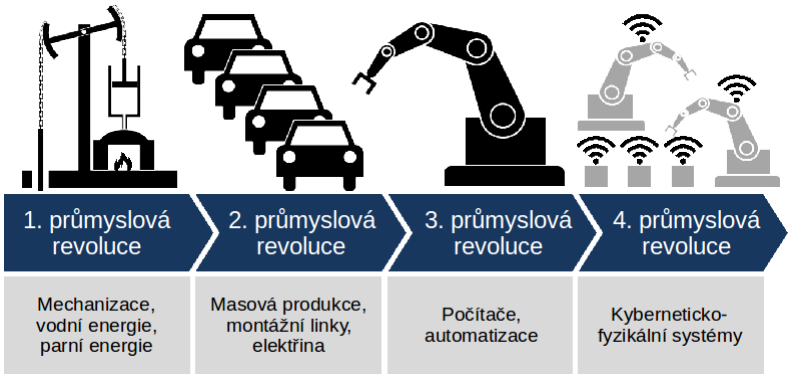
V poslední kapitole otestuji moje řešení řídicího systému a zhodnotím dosažené výsledky.

1 Průmysl 4.0

V této kapitole vysvětlím základní myšlenku průmyslu 4.0. Dále se budu zabývat 3D tiskem a síťovou komunikací v průmyslu.

1.1 Definice průmyslu 4.0

Průmysl 4.0 neboli čtvrtá průmyslová revoluce přináší nové možnosti, jak průmysl zdigitalizovat. Tato revoluce není pouze o robotizaci a automatizaci, ale o propojení virtuálního a skutečného světa. Konceptem tohoto průmyslu by měla být komunikace strojů a různých součástek pomocí senzorů, virtualizace, kdy každý fyzický objekt bude mít své digitální dvojče nebo řešení řízení pomocí cloudu. Průmysl 4.0 by měl odběratelům nabídnout možnost přizpůsobit si prvek podle svých potřeb, protože chytré továrny nabídnou možnosti modularity a rekonfigurace výrobní linky. Chytré továrny a jejich dodavatelé by měli být propojení pomocí průmyslového internetu, aby se vše mohlo zpracovávat v reálném čase a celkový čas výroby produktu se tím zkrátil na minimum.[1]



Obr. 1.1: Diagram znázorňující 4 industriální revoluce.[2]

1.2 Základní principy průmyslu 4.0

Průmysl 4.0 má šest důležitých principů, podle kterých by jsme se měli řídit, aby se idea této průmyslové revoluce naplnila v co největší míře.

Obr. 1.2: Základní principy průmyslu 4.0

1. Interoperabilita, neboli schopnost kyberfyzikálních systémů, osob a dalších entit, které jsou součástí inteligentní továrny, společně komunikovat pomocí dedikovaných sítí.
2. Virtualizace, která zaručí propojení fyzických objektů s jejími virtuálními modely. Na virtuálních modelech můžeme dělat simulace a zdokonalit výsledný fyzický objekt.
3. Decentralizace umožní jednotlivým sub-systémům být autonomní v rozhodování a řízení.
4. Zpracování v reálném čase je klíčovým předpokladem v průmyslu 4.0, protože systémy musí umět komunikovat, rozhodovat se a řídit v reálném světě.
5. Orientace na službu preference výpočetní filosofie nabízení a využívání standardních služeb. Pro integraci aplikace použijeme koncept SOA (Service Oriented Architectures).
6. Modularita umožní systémům autonomní rekonfiguraci na základě přítomné podmínky. [5] [3] [6]

1.2.1 Intereporetabilita

Schopnost kyberfyzikálních systémů, osob a dalších entit, které jsou součástí inteligentní továrny, společně komunikovat pomocí IoT (Internet of Things) a IoS (Internet of Services). Intereporetabilitu můžeme rozdělit podle formy a podoby. Dle formy na:

- Syntaktická intereporetabilita
- Sémantická intereporetabilita

Dle podoby na:

- Horizontální integrace
- Vertikální integrace [4]

Syntaktická intereporetabilita

Data, která jsou přenášena mezi zařízeními, musí mít nadefinovanou syntaxi a kódování. Toto pravidlo platí jak pro komunikační protokoly, které nazýváme higl-level protokoly (HTML, XML, SQL), tak také pro bitovou komunikaci. Tato syntaxe a kódováni nevyjadřuje význam dat, ale docílíme tím standardizace datových formátů a forem komunikace v celém systému.[4]

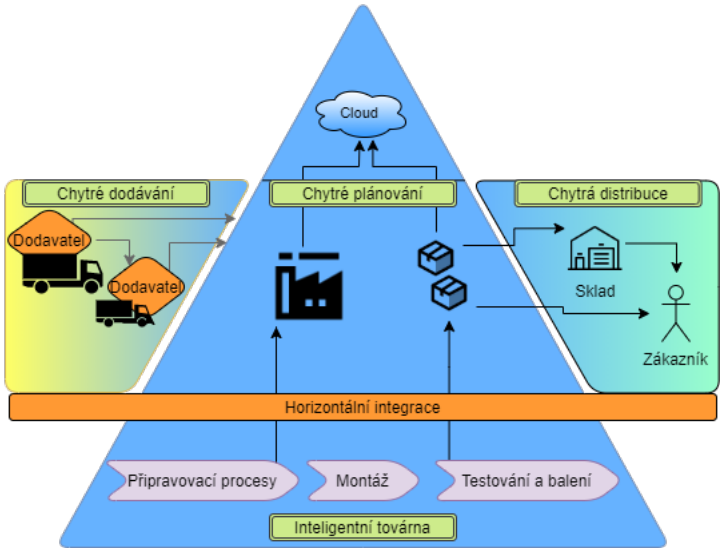
Sémantická intereporetabilita

Sémantická forma vyjadřuje význam (kontext) dat, protože dokáže přesně vyjádřit význam přenášených dat mezi různými systémy, tato komunikace umožní kombinovat datové prvky z různých slovníků a vyhledávat informace v systému. U sémantické formy se k datům přidají metadata, která mají informace o posílaných datech. Abychom docílili funkčnosti této komunikace, musí mít všechny strany, které komunikují, stejný informační model.[4]

Horizontální integrace

Horizontální integrace přesahuje interní operace výrobního podniku. Prostřednictvím nové generace globálních hodnototvorných sítí propojuje všechny články dodavatelsko-odběratelského hodnototvorného řetězce od dodavatelů přes výrobce až po distribuci koncovému zákazníkovi a následný servis, včetně integrace obchodních partnerů a zákazníků a nových obchodních a partnerských modelů napříč jednotlivými zeměmi a kontinenty."[4] Kvůli této integraci už firmy nebudou vyrábět nadbytek produktů, ale přesně tolik, kolik jich odběratel/zákazník bude potřebovat, také tu pro zákazníka bude vetší možnost si produkt upravit dle jeho požadavků, a protože všichni dodavatelé budou s montážním závodem propojeni průmyslovým internetem tzv. IIoT (Industrial Internet of Things), bude se požadavek zákazníka v reálném čase

zpracovávat u dodavatelů, a tím se zkrátí čas dodání produktu do montážního závodu a poté k zákazníkovi.



Obr. 1.3: Horizontální integrace

Vertikální integrace

Vertikální integrace je uspořádání podniku od nejnižší úrovně řízení, která funguje v reálném čase, přes úroveň, která plánuje výrobu, až po nejvyšší úroveň, ve které je ERP systém (Enterprise Resource Planning), který rozhoduje o výrobě v časovém rozpětí dnů až týdnů. V průmyslu 4.0 se využívá virtualizace nejnižší vrstvy, která vytvoří přímé spojení nejnižší úrovně s nejvyšší úrovní, a díky tomu dokáže ERP systém flexibilně plánovat výrobu a dokáže trasovat výrobu jednotlivých produktu a vytvořit tak každému produktu digitální rodný list, který bude mít informace o jeho výrobě. [7]

1.2.2 Virtualizace

Virtualizace je jeden ze základních prvků průmyslu 4.0. Nahrazuje fyzické prototypy virtuálními návrhy, na kterých lze provádět simulaci, a zcela tak odladit prototyp bez nutnosti ho vyrobit fyzicky. Virtuální prototypy můžou být jak výrobku, tak i výrobních prostředků a procesů. Pomocí virtuálně navrhnutého prototypu můžeme jeho uvedení do procesu udělat v rámci jednoto integrovaného procesu, na kterém se podílí jak výrobce samotný, tak i jeho dodavatelé. [4]

V rámci virtualizace můžeme uvažovat o použití tří možných způsobů zobrazení:

- Virtuální realita (VR)
- Rozšířená realita (AR)
- Light-guide system

Virtuální realita

Pomocí virtuálně navrhnutých prototypů můžeme vytvořit virtuální prostor, který bude vypadat například jak reálná výrobní linka, nebo také pomocí virtuálního prostoru můžeme navrhnou nejlepší rozpoložení výrobních linek.

Virtuální a fyzický objekt jsou spolu propojeny a vytvářejí tak dvojčata, z niž jedno je v reálném světe a druhé v digitálním, a díky tomu může virtuální model kopírovat pohyby fyzického objektu.

Pomocí virtuálních brýlích se můžeme bez omezení pohybovat po pracovišti a kontrolovat výrobní proces, dělat revize nebo simulace.[5]

Rozšířená realita

Rozšířená realita vylepšuje zobrazení v reálném čase. Pracuje se vstupy jako je video, obrázky, zvuk a data z GPS. Pro takové zpracování se používají algoritmy počítačového vidění, aby obraz byl co nejpřesnější. AR rozšiřuje reálný obraz o virtuální obrazy nebo rozšiřující informace o věcech nebo lidech. Pro AR existují dva způsoby jak ji použít: [5]

- Optical see-through - průhledné brýle, na které se bude promítat rozšířený obraz
- Video see-through) - tablet nebo telefon, který zachycuje obraz přes čočku a při zpracování se do výsledného obrazu přidá rozšířený obraz.

Light-guide system

Tento systém pomocí senzorů a světel navádí uživatele co má dělat.

1.2.3 Decentralizace

Jednotlivé sub-systémy budou mít možnost se autonomně rozhodovat a řídit. Tímto vznikne síť propojených sub-systémů, kde nebude žádný nadřazený systém, který by koordinoval funkčnost.

1.2.4 Orientace na služby

Systém se budeme snažit rozdělit na jednotlivé sub-systémy, které budou dělat specifickou úlohu. Tímto úkonem docílíme toho, aby systém byl robustější, a když se objeví chyba v jedné službě, tak to nepostihne ostatní, protože na sobě nejsou závislé. Tento koncept integrace se nazývá SOA. Služby v architektuře SOA mají dvě strany. Na jedné straně jsou to poskytovatelé služeb a na straně druhé jsou spotřebitelé, kteří na základě sledování poskytovatele dokáží dosáhnout vlastních cílů.[4]

1.3 3D Tisk

3D tisk je proces výroby trojrozměrných objektů, který funguje na principu vrstvení materiálu na sebe, dokud není celý objekt dokončen. Abychom mohli vytisknou objekt, musíme ho nejdříve vymodelovat a poté poslat soubor příkazů do 3D tiskárny, která objekt vytiskne. Příkazům, které jsou součástí souboru se říká G-kódy a M-kódy.

G-kód a M-kód jsou částečně standardizované jazyky. To znamená, že standardní příkazy pro posuv tiskové hlavy, nastavení pracovní roviny nebo nastavení teploty tiskové hlavy budou u všech výrobců stejné.

Výpis 1.1: Příklad začátku tisku

```
M107      ;vypnutí ventilátoru                                    1
M190 S50    ;nastavení teploty podložky                           2
M104 S217     ;nastavení teploty tiskové hlavy s čekáním          3
G28     ;domovská pozice                                          4
G1 X-10 Y50 Z20 F10000 ;pohyb na souřadnice                       5
M109 S217   ;nastavení teploty tiskové hlavy s čekáním            6
G21     ;nastavení jednotek na milimetry                          7
G90     ;nastavení absolutní pozice hlavy                         8
M82     ;nastavení absolutní pozice extruderu                     9
G92 E0    ;nastavit aktuální polohu extruderu                     10
G1 E-6.00000 F2400.00000 ;pohyb extruderu                         11
G92 E0                                                            12
G1 Z0.200 F24000.000 ;pohyb hlavy o určité rychlosti              13
G1 X103.427 Y103.697 F24000.000                                   14
```

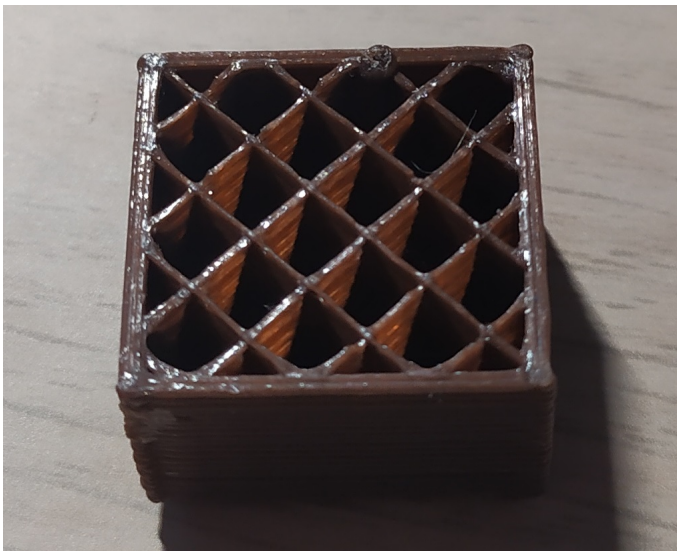
1.3.1 3D tisk v průmyslu 4.0

3D tisk se dříve používal hlavně pro vytváření prototypů produktu. Jakmile se produkt vyladil, vytvořila se například vstřikovací forma, díky které už 3D tisk nebyl potřeba.

V průmyslu 4.0 se bude ve větší míře používat 3D tisk přímo ve výrobě, protože přináší inovativní řešení například v automobilovém průmyslu. Když vytiskneme karoserii auta na 3D tiskárně, můžeme docílit stejné pevnosti a daleko nižší spotřeby materiálu a tím i menší hmotnosti. Toto dokážeme tak, že výplň karoserie nebude celistvá, ale vymyslí se jiná, která bude mít stejné vlastnosti. Když snížíme hmotnost karoserie, dokážeme například zvýšit možnou ujetou vzdálenost elektromobilu.

3D tisk také dovolí zákazníkovi větší možnosti úpravy produktu podle sebe. Pokud zůstaneme u automobilového průmyslu, tak při koupi auta budeme moci vybrat například tvar světel nebo jiné úpravy vzhledu, které udělají auto unikátní.

[8]



Obr. 1.4: Příklad jednoho možného řešení struktury výplně produktu při 3D tisku

1.4 Komunikace v průmyslu

V této kapitole popíši komunikace, které budou použity v bakalářské práci. Jsou to také často používané komunikace v průmyslu.

1.4.1 REST

REST neboli Representational state transfer je architektura rozhraní, která může být implementována různými způsoby. Požadavky na klienta přes RESTful API (Application Programming Interface) jsou nejčastěji poslány pomocí HTTP (Hypertext Transfer Protocol) protokolu v několika formátech: JSON (Javascript Object Notation), HTML, XLT, Python, PHP nebo prostý text. Nejpoužívanější formát je JSON, který je jednoduchý vytvořit a je dobře čitelný pro lidi, tak i pro stroje. Ve výpisu 1.2 je příklad JSON formátu, který budu používat jako konfigurační soubor.[9]

Výpis 1.2: Příklad JSON formátu

```
{                                                    1
    "MQTT": {                                        2
        "IP_address": "127.0.0.1",                   3
        "port": 1883,                                4
        "auto_connect": false                        5
    }                                                6
}                                                    7
```

V bakalářské práci budu využívat návrhového vzoru klient-server, který vynucuje oddělení zájmů. To zaručuje, že klient a server se mohou vyvíjet nezávisle na sobě. Zatím co se klient a server nezávisle na sobě vyvíjí, tak je důležité hlídat, aby se rozhraní mezi klientem s serverem neporušilo.[10]

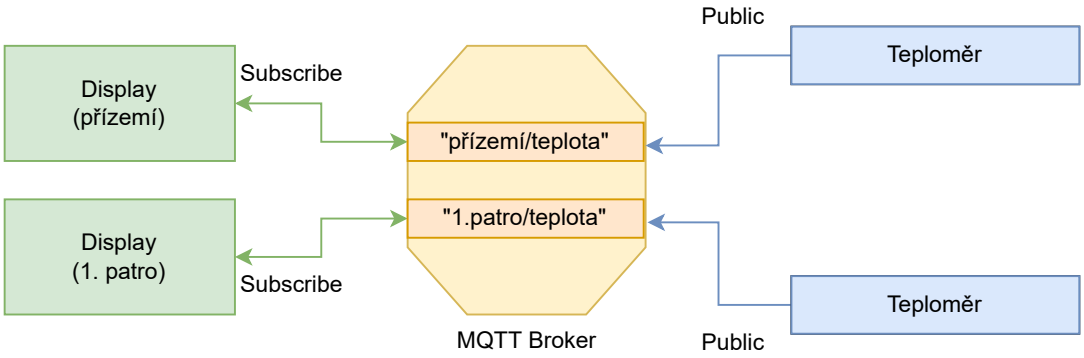
Jak už bylo zmíněno, tak požadavky RESTful API jsou dodávány pomocí HTTP protokolu. V tomto případe nám vznikají možnosti, jak s koncovým bodem můžeme komunikovat, HTTP protokol nabízí 4 metody:

- GET - získání dat
- POST - vytvoření/odeslání dat
- PUT - úprava dat
- DELETE - smazání

1.4.2 MQTT

Protokol MQTT (MQ Telemetry Transport) je v IoT velmi rozšířený komunikační protokol, který nabízí velmi jednoduchou implementaci do aplikací. Tento protokol funguje na komunikačním modelu public/subscribe. U této implementace to znamená, že máme v síti broker neboli server, který distribuuje zprávy všem připojeným zařízením podle odebíraných témat. Klienti přes tento protokol mohou zprávy odesílat i přijímat, ale vždy tato komunikace běží přes broker. Jednotlivé zprávy patří do určitých témat (topic). Témata fungují hierarchicky a jsou oddělená lomítky. takže když budeme mít dům se dvěma patry, tak hierarchie témat může být následující

"přízemí/obývací-pokoj/teploměr". Témata v MQTT jsou řetězce UTF-8, takže je možné používat diakritiku.[11][12]



Obr. 1.5: Příklad modelu MQTT komunikace

Pro přihlášení odběru má klient několik možností. První možnost je odebírat pouze jedno téma. Odebírané téma bude vypadat následovně: "přízemí/teplota". Další možností je používat u témat speciální znaky a to jsou '#' a '+'. Tyto znaky nám pomůžou v odebírání více témat naráz. Když použijeme při odebírání tématu tento řetězec "+/teplota", tak budeme odebírat teploty ze všech pater, a nebo můžeme použít řetězec "přízemí/#", který nám poskytne odběr všech témat na tomto patře.[11]

MQTT má také QOS (Quality of Service), které je rozděleno na tři úrovně. Toto nastavení nám zaručí, jak bude zpráva doručena. Jednotlivé úrovně fungují takto:

- úroveň 0 - Pošli zprávu nanejvýš jednou. To znamená, že zpráva bude odeslána a dál klient neřeší, jestli zpráva byla přijata, nebo ne.
- úroveň 1 - Pošli zprávu alespoň jednou. Tato zpráva bude poslána brokeru, který to pošle odběratelům určeného tématu. Pokud odběratelé zprávu přijmou, tak odešlou brokeru informaci o přijetí zprávy. Broker si proto smaže zprávu z paměti a pošle klientovi, který zprávu poslal, že všichni zprávu přijali. Klient si to zaznamená a zprávu vymaže z paměti.
- úroveň 2 - Pošli zprávu přesně jednou. Tato zpráva je zpracovaná stejně jak na úrovní 1 s tím rozdílem, že broker pošle publisherovi, že zprávu přijal. Po přijetí této zprávy publisher zprávu smaže a potvrdí to brokeru.[11]

2 Požadavky na systém

Abych mohl začít s návrhem systému, tak si musím nadefinovat požadavky, které od řídícího systému očekávám. Požadavky pro tento řídící systém budou následující:

- Komunikace a řízení 3D tiskárny
- Komunikace s MES
- Autonomní řízení
- Grafické uživatelské rozhraní
- Log
- Robustnost a jednoduchost

Komunikace a řízení 3D tiskárny

Řídicí systém bude připojen k 3D tiskárně pomocí sériové komunikace USB (Universal Serial Bus). Řídicí systém bude v reálném čase monitorovat zásobník 3D tiskárny, jestli na sériové lince nejsou k dispozici data k přečtení. Další funkce řídicího systému bude posílání příkazů do 3D tiskárny, který funguje v režimu jednoduché handshake komunikace, kdy systém pošle tiskárně příkaz a čeká na odpověď od 3D tiskárny, aby mohl poslat další.

Komunikace s MES

Komunikace s MES systémem je další důležitý požadavek, který řídicí systém musí mít implementovaný. Tato komunikace bude fungovat přes Ethernet. Teplota a pozice tiskárny budou distribuované přes MQTT protokol a pro požadavky na práci vytvořím REST API.

Autonomní řízení

Systém se bude samostatně rozhodovat a řídit. Příkladem autonomního řízení v řídicím systému bude rozhodování, zda je 3D tiskárna zaneprázdněná nebo bude moci tisknout. Když bude mít možnost tisknout, tak se poptá MES, jestli má ve frontě k dispozici práci k vytisknutí.

Grafické uživatelské rozhraní (GUI)

Grafické uživatelské rozhraní bude vytvořeno pro dotykový displej, tak aby bylo přehledné a dobře se na dotykovém displeji ovládalo. V uživatelském rozhraní bude možnost nastavit komunikaci s 3D tiskárnou, to znamená výběr portu, na kterém bude tiskárna připojena a nastavení vhodné modulační rychlosti (baud rate). Pomocí GUI bude možné pohybovat s krokovými motory tiskárny, nastavovat teplotu tiskové

hlavy, podložky a komory, nebo vybrat soubor pro tisk a pomocí dalšího ovládání bude možné tisk zapnout, pozastavit nebo vypnout.

Log

Log který bude zaznamenávat teploty během tisku a zároveň je bude posílat do MES.

Robustnost a jednoduchost

Systém by měl být robustní, aby se při jakékoliv chybě nezhroutil a dokázal se s chybou vyrovnat a opravit ji. Robustnost je v menší míře závislá na jednoduchosti systému. Když nebude systém moc složitý, bude menší pravděpodobnost, že se naskytne v softwaru chyba.

3 HW platforma

V této kapitole zdůvodním, proč jsem se rozhodl pro vybranou HW platformu a popíši výběr programovacího jazyku pro vývoj řídicího systému.

3.1 Výběr HW

Výběr HW byl vcelku jednoduchý, i když na trhu existuje spoustu jednodeskových počítačů, tak pro mě byla nejlepší volba malý jednodeskový počítač Raspberry Pi. Tuto HW platformu jsem si vybral z důvodu nízké ceny a dobrého výkonu, které tato platforma nabízí.

Řídicí systém budu vyvíjet na modelu Raspberry Pi 4 Model B - 8GB RAM, který je nejvýkonnějším jednodeskovým počítačem, jaký firma Raspberry Pi Trading vyvinula, ale řídící systém by měl bez problému fungovat i na méně výkonných modelech. Na jednodeskovém počítači bude operační systém Raspbian, který funguje na unixovém jádře. Tento systém jsem si zvolil, protože je na něm jednoduchá implementace programů, které jsou napsané v Pythonu a systém je přímo vytvořen pro Raspberry Pi.

3.2 Výběr programovacího jazyka

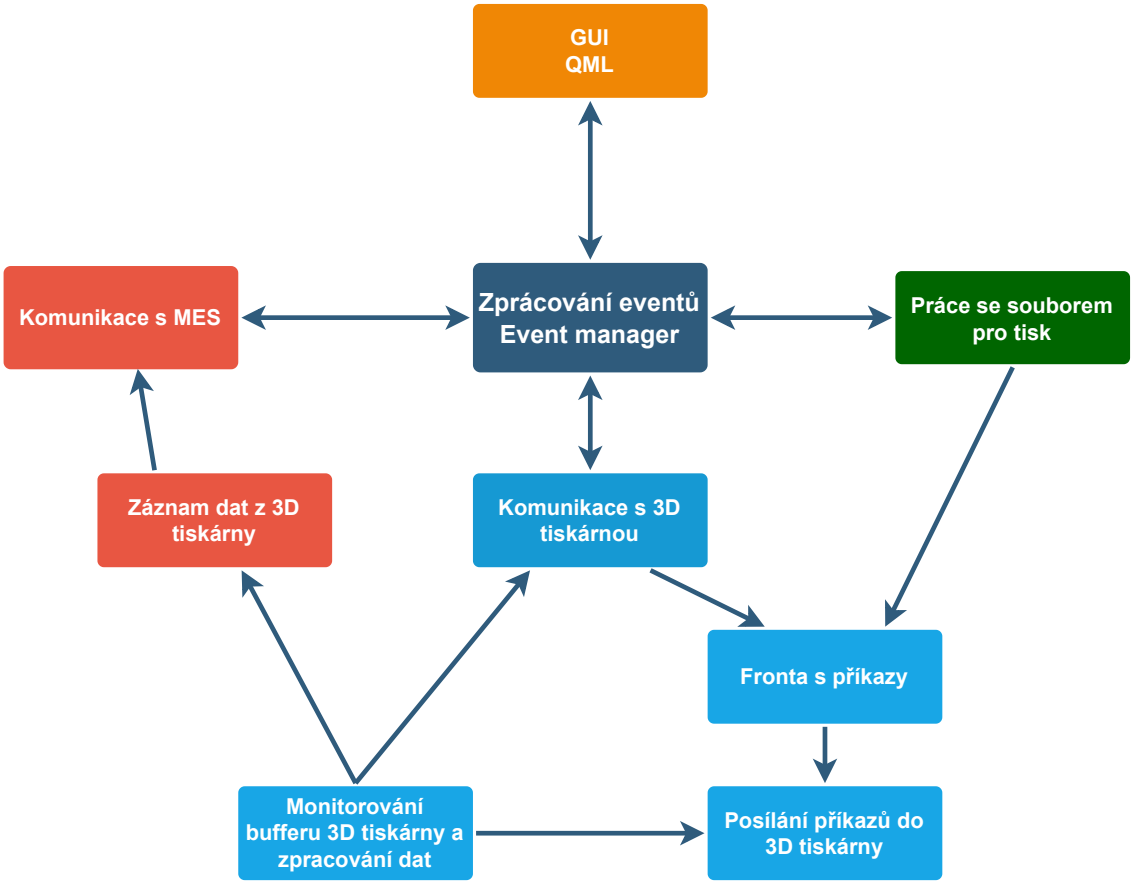
Jako první jsem si vybral tři programovací jazyky, ve kterých umím částečně programovat:

- C - nízkoúrovňový, kompilovaný jazyk
- C++ - rozšíření jazyka C, které nabízí objektově orientované a generické programování
- Python - vysokoúrovňový, skriptovací jazyk, který nabízí programovací styly jako je objektově orientované programování, procedurální a generické programování

Abych vybral nejlepšího kandidáta, udělal jsem si rešerši dostupných knihoven pro jednotlivé jazyky. Zjistil jsem, že pro všechny programovací jazyky jsou vytvořené knihovny, které budu potřebovat pro vytvoření řídícího systému, ale zároveň jsem si při tomto průzkumu uvědomil, že by bylo dobré systém vyvíjet pomocí objektově orientovaného programovaní, který programovací jazyk C nenabízí. Ze zbylých dvou programovacích jazyků jsem si vybral Python, protože je to moderní jazyk a rád bych se v něm naučil vyvíjet komplexní aplikace. Python je dostupný na všech běžných platformách jako je Unix, MS Windows, macOS a Android a je součástí základní instalace většiny linuxových systémů, takže implementace řídicího systému na jakémkoli operačním systému bude opravdu jednoduchá.

4 Struktura systému

Při programování řídicího systému budu používat objektově orientované programování. Vytvořil jsem návrh struktury systému a rozdělil jednotlivé funkce systému do sub-systémů, které spolu budou komunikovat podle následujícího diagramu.



Obr. 4.1: Návrh struktury řídicího systému pro tiskovou farmu.

Systém jsem rozdělil podle požadavků, které jsou popsány v kapitole 2. Nejprve jsem začal komunikací řídicího systému s 3D tiskárnou. Komunikaci jsem rozdělil na dvě části. První část se zabývá monitorováním zásobníku 3D tiskárny a druhá část posíláním příkazů do 3D tiskárny. Monitorování bude běžet v samostatném vláknu a bude číst zásobník tiskárny. Pokud se na zásobníku objeví data, tak je po přečtení zpracuje a pošle přes event manažer všem ostatním modulům.

Posílání příkazů do tiskárny bude také běžet v samostatném vláknu. Příkazy k odeslání jsou uloženy ve FIFO (First In, First Out) frontě. Pro správné fungování bude toto vlákno spolupracovat s monitorováním. 3D tiskárna po každém přijatém příkazu pošle nazpět zprávu "ok". Pokud tato zpráva dojde, může systém poslat další příkaz. Jinak čeká než odpověď dorazí z tiskárny. Fronta poskytující příkazy bude

prioritní, což v této aplikaci bude znamenat, že příkazy, které přidal uživatel v GUI mají větší váhu a zpracují se dřív než příkazy, které jsou přidané ze souboru. Tato funkcionalita je přidaná z toho důvodu, aby uživatel během tisku mohl popřípadě měnit teplotu extruderu a nebo pohnout s tiskovou hlavou, kdyby to bylo zapotřebí.

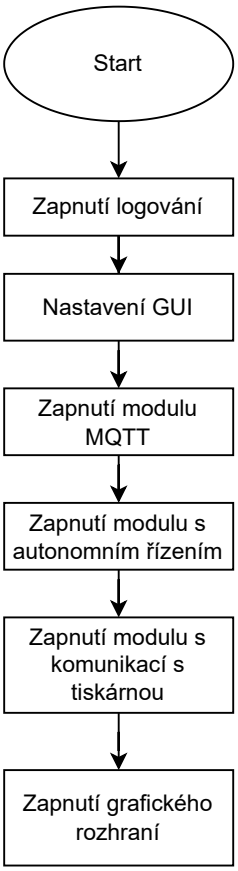
Další modul bude zpracovávat soubory pro tisk. Tento modul otevře soubor pro tisk a začne postupně přidávat všechny příkazy do tiskové fronty a zároveň bude muset smazat případné komentáře v G-kódu.

Jeden z nejdůležitějších modulů bude grafické uživatelské rozhraní, které bude poskytovat uživateli možnost nastavovat sériové připojení tiskárny. Bude zde možnost nastavit komunikaci pomocí MQTT a REST api. V neposlední řadě zde bude moci uživatel ovládat nejdůležitější věci na tiskárně (pohyb, teplota, tisk). Tento modul bude dostávat všechny informace přes event manažer a nebude přímo závislý na ostatních modulech.

Jako poslední modul na obrázku najdeme komunikaci s MES systémem, která bude posílat teplotu tiskárny a pozici přes protokol MQTT. Dále bude poptávat MES systém, jestli pro tiskárnu není k dispozici práce k výtisku.

5 Implementace řídicího systému

V této části řeším programové řešení řídicího systému. Každá podkapitola bude řešit jeden nebo více sub-systémů, které spolu souvisí. V následujícím diagramu je popsán postup spuštění systému.



Obr. 5.1: Ukázka startu systému ve funkci Main

5.1 Komunikace s 3D tiskárnou

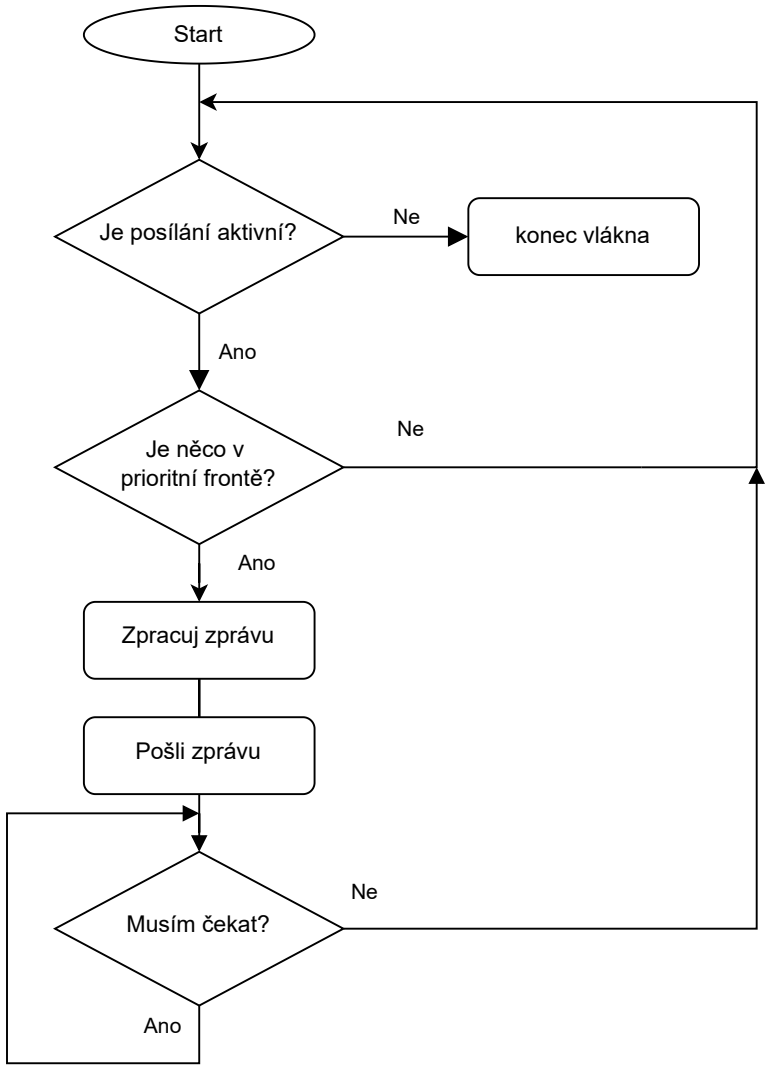
Pro naprogramování komunikace s 3D tiskárnou jsem zvolil knihovnu pySerial, která nabízí vytvoření třídy pro sériovou komunikaci. Pro řízení a komunikaci jsem vytvořil třídu *Printer*, která využívá třídu *Serial* ze zmíněné knihovny. Pro připojení 3D tiskárny vytvořím objekt *comm* třídy *Serial*, který nastavím na vhodný port, baudrate, časový limit zápisu a časový limit čtení. Toto nastavení se dělá při inicializaci objektu typu *Printer*. Při inicializaci spolupracuje objekt s modulem *Settings*, odkud získává výchozí hodnoty pro sériovou komunikaci. Pro navázání komunikace jsem vytvořil metodu connect(), která se pokusí otevřít sériovou komunikaci, a při

úspěšném otevření se zapnou dvě samostatná vlákna. První vlákno se stará o monitoring zásobníku tiskárny a druhé posílá příkazy do tiskárny. Po zapnutí těchto vláken pošle systém do tiskárny příkaz M115, který poskytne informace o softwaru tiskárny. Poté systém uvědomí všechny moduly, které tuto informaci potřebují, že systém úspěšně navázal komunikaci s 3D tiskárnou. Když se nepodaří s tiskárnou navázat komunikaci, tak se chyba vytiskne do konzole a dále se pošle tato chyba event manažerovi.

Výpis 5.1: Metoda pro navázání komunikace s 3D tiskárnou

```
    def connect(self) -> None:                                          1
        try:                                                            2
            self._comm.open()                                           3
            time.sleep(1)                                               4
        except Exception as ex:                                         5
            post_event("Serial_ERROR", ex)                              6
            print(ex)                                                   7
        self._start_threads()                                           8
        time.sleep(2)                                                   9
        self._command_to_send.put("M110")                               10
        self._command_to_send.put("M115")                               11
        post_event("printer_connection", "CONNECTED")                   12
```

Když je komunikace navázána, je možné přidávat příkazy do fronty *_command_to_send*. Pokud je příkaz přidán do fronty, tak ho odbaví vlákno, ve kterém je spuštěný *_sender*. Toto vlákno pracuje ve while cyklu, který je podmíněný otevřenou sériovou komunikací. Jakmile se komunikace uzavře, tak se vlákno dostane do neaktivního stavu a je ho potřeba opětovně zapnout pomocí metody *connect*.



Obr. 5.2: Ukázka vývojového diagramu pro vlákno, které posílá příkazy.

Jakmile se program dostane do while cyklu, tak se při vykonávání snaží zachytit výjimky v programu. V tomto případě zachycuje výjimku *queue.Empty*, která indikuje, že fronta na příkazy je prázdná. V tu chvíli program ukončí tuto iteraci použitím příkazu *continue*. Jakákoli jiná výjimka bude neočekávaná chyba v sériové komunikaci. Ta se zpracuje tak, že se sériová komunikace uzavře a systém dá vědět event manažerovi, že se naskytla chyba v sériové komunikaci. Pokud se nenaskytne žádná výjimka během provádění kódu, tak program dostane příkaz z fronty, následně ho zpracuje, pošle do tiskárny a čeká na odpověď, aby mohl poslat další příkaz. Toto čekání kooperuje s vláknem, kde je spuštěný monitoring zásobníku a bude popsán níže v této kapitole. Zpracování zprávy probíhá v objektu *G_Command_with_line*.

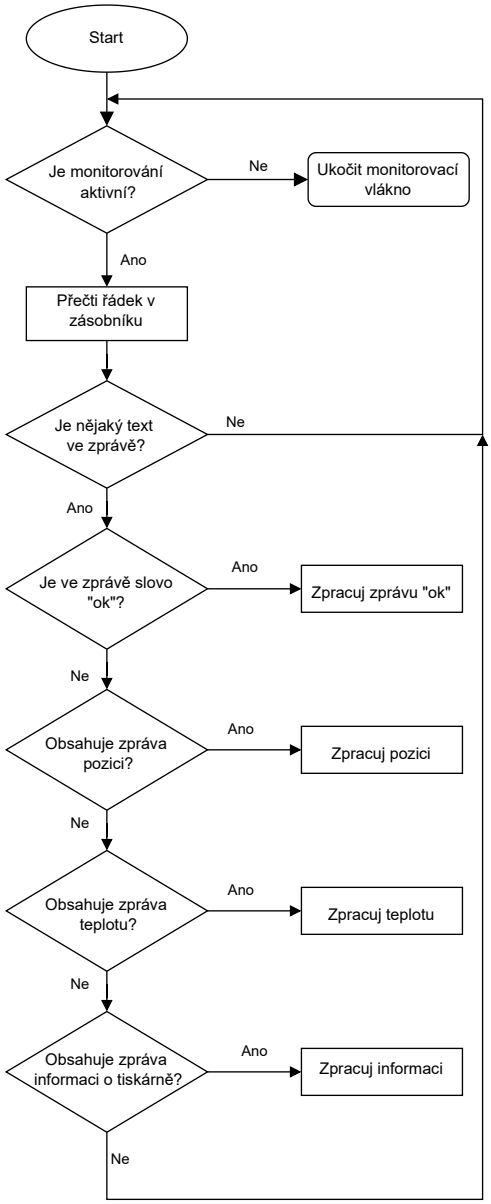
Výpis 5.2: Metoda pro zpracování Gkódu

```
@dataclass
class G_Command_with_line():
    command: str
    n_lines: int
    def process(self) -> bytes:
        command = "N" + str(self.n_lines) + "␣" + self.command
        checksum = string_checksum(command)
        command += "*" + str(checksum) + "\n"
        return command.encode()
    def __str__(self):
        return "N" + str(self.n_lines) + "␣" + self.command

def string_checksum(string: str) -> int:
    checksum = 0
    for c in bytearray(string.encode()):
        checksum ^= c
    return checksum
```

Tato datová třída má dva argumenty, a to příkaz a číslo příkazu. Číslo příkazu program hlídá ve while cyklu, kdy po každém poslaném příkazu se hodnota proměnné *n_line* inkrementuje o jedničku. Pro zpracování příkazu je potřeba použít metodu *process*. Tato metoda přidá na začátek zprávy číslo příkazu ve tvaru "N<číslo> "a na konec přidá kontrolní součet celé zprávy. Pokud by při komunikaci nastala chyba a zpráva by neodpovídala kontrolnímu součtu, tiskárna si zažádá o znovu poslání zprávy, která vypadá takto: "Resend: <číslo>".

Na monitorování zásobníku 3D tiskárny používá systém vlákno, ve kterém je spuštěná metoda *_monitoring*. Toto vlákno opět pracuje ve while cyklu, který je podmíněný otevřenou komunikací s tiskárnou.



Obr. 5.3: Ukázka vývojového diagramu pro vlákno, které monitoruje zásobník tiskárny.

Pro čtení zpráv ze zásobníku tiskárny se používá metoda *_reader*. Tato metoda přečte jeden řádek zásobníku, dekóduje ho pomocí UTF-8 (Unicode Transformation Format) a vrátí dekódovaný řetězec znaků. Pokud se při čtení naskytne chyba, vypíše se tato chyba do terminálu, uzavře se sériová komunikace a návratová hodnota se nastaví na None.

Výpis 5.3: Metoda pro čtení zásobníku tiskárny

```
def _reader(self):
  if self._comm is None:
    return None
    try:
      string = self._comm.readline()
    except Exception as ex:
      post_event("Serial_ERROR", ex)
      print(f"Naskytla se chyba {ex}")
      self.disconnect()
      return None
    string = string.decode('utf-8')
    return string
```

Po získání zprávy začne její zpracování. Pokud zpráva bude začínat "ok", tak to indikuje, že předchozí poslaný příkaz byl přijat a je možné poslat další. Proto aby mohl být poslaný další příkaz je potřeba ukončit čekání vlákna, které se stará o posílání příkazů. Toho docílí systém pomocí objektu *Event* z knihovny *threading*, který obě vlákna využívají. Metoda *event.set()* pošle všem čekajícím vláknům, že mohou pokračovat.

Pokud zpráva začíná na "X:"a nebo se vyskytují jiné písmena z os ve zprávě, tak začne zpracování polohy tiskárny.

Výpis 5.4: Zpracování zprávy s polohou

```
#tvar zprávy:
#X:-6.00 Y:4.00 Z:0.00 E:0.00 Count X:-240 Y:160 Z:0
if message.startswith("X:") or " Y:" in message or/
    " Z:" in message or " E:" in message:
  string = message.split("Count")
  data = {}
  for match in re.finditer(regex_position, string[0]):
    data[match["axis"]] = match["value"]
  post_event("position_update", data)
  continue
```

Zprávu s polohou je nejprve nutné rozdělit slovem "Count", protože druhá část není důležitá. Po rozdělení zprávy systém použije regulární výrazy, aby našel všechny osy ve zprávě a mohl je následovně zpracovat. Výsledek bude uložený v typu *dictionary*, kde klíč bude název osy a hodnota bude pozice, kde se hlava tiskárny nachází. Tyto data jsou distribuována pomocí event manažeru dalším modulům.

Jednou z dalších možných zpráv je indikace teplot tiskárny. Tato zpráva začíná

"T:"nebo "T0:". Pokud tato zpráva dojde a bude odpovídat podmínce, tak dojde k jejímu zpracování.

Výpis 5.5: Zpracování zprávy s teplotou

```
# zpráva: T:201 /202 B:117 /120 C:49.3 /50
if "␣T:" in message or "␣T0:" in message or /
    "␣B:" in message or "␣C:" in message:
  result = {}
  for match in re.finditer(regex_temp, message):
    values = match.groupdict()
    tool = values["tool"]
    try:
      actual = float(match.group(3))
      target = None
      if match.group(4):
        target = float(match.group(4))
        result[tool] = (actual, target)
    except ValueError:
      print("Vyskytla␣se␣chyba")
  if self._extruder_count == 1:
    #Zpracování teploty, když je pouze jeden extruder
  else:
    if "T" in result:
      self._temp[0], self._targetTemp[0] = result["T"]
    for i in range(self._extruder_count-1):
      if f"T{i}" in result:
        self._temp[i+1], self._targetTemp[i+1] = result["T"]
  if "B" in result:
    self._bedTemp, self._targetBedTemp = result["B"]
  if "C" in result:
    self._chamberTemp, self._targetChamberTemp = result["C"]
  data = {"time":datetime.datetime.now().ctime(),
      "tools":[self._temp, self._targetTemp],
      "bed":[self._bedTemp, self._targetBedTemp],
      "chamber":[self._chamberTemp, self._targetChamberTemp]
      }
  post_event("temperature_update", data)
  continue
```

Pro vyhledání jednotlivých teplot systém opět využije regulární výrazy, které rozdělí teploty do jednotlivých skupin. Po rozdělení se všechna data zpracují a uloží do proměnných. Poté se vytvoří proměnná *data* typu *dictionary*, kde se uloží všechna

posbíraná data a se přidá k nim čas, kdy byla přijata. Tato proměnná je potom poslána ostatním modulům pomocí event manažera.

Poslední důležitá metoda ve třídě *Printer* je *print_from_file_buffered* pomocí které se načte g-kód ze souboru a přidá se do fronty pro příkazy.

Výpis 5.6: Metoda pro načtění G-kódu ze souboru

```
def print_from_file_buffered(self, file: str):
  self._is_loading = True
  try:
    f = open(file)
    print(f'soubor␣{file}␣se␣podařilo␣otevřít')
  except Exception as ex:
    print(ex)
    self._is_loading = False
    return
  self._command_to_send.put("start")
  self.add_script_to_queue("start")
  for line in f:
    f_line = line.strip()  # vymazání zbytečných mezer
    f_line = decommenter(f_line)
    if f_line.isspace() is False and len(f_line) > 0:
      self._command_to_send.put(f_line)
      #print(f"příkaz {f_line} byl přidát do fronty")
  f.close()
  self.add_script_to_queue("end")
  self._command_to_send.put("done")
  self._is_loading = False
```

Této metodě je potřeba předat argument *file*, neboli cestu k souboru s G-kódem. V metodě se nejprve zkusí otevřít soubor. Nastane-li chyba při otevírání souboru, tak se tato chyba vypíše do terminálu a ukončí se bez návratové hodnoty. Jakmile se soubor otevře, tak se do fronty s příkazy přidá řetězec znaků "start", aby systém zaznamenal začátek tisku. Jako další se do fronty přidá startovací skript, pokud je nějaký nastavený. Tento skript si může nastavit uživatel libovolně v GUI. Jakmile je skript přidán, tak program začne procházet soubor s G-kódem a postupně přidá všechny příkazy do fronty. Po dokončení této operace se do fronty přidá ukončovací skript, který si opět uživatel může vytvořit podle sebe. Pro zaznamenání dokončení se na konec fronty přidá řetězec "done".

5.2 Modul pro nastavení

Tento modul slouží pro ukládání nastavení do souboru, který bude mít formát JSON. Modul funguje tak, že první objekt, který zavolá funkci *GetSettingsManager*, vytvoří třídu *Settings* jako globální proměnnou. Každý další modul, který tuto funkci zavolá, pak dostane vytvořený objekt.

Výpis 5.7: Funkce pro získání objektu Settings

```
_instance = None
def GetSettingsManager() -> Settings:
  global _instance
  if _instance is not None:
    return _instance
  else:
    _instance = Settings()
    return _instance
```

Při inicializaci této třídy se hledá soubor s nastavením. Pokud třída tento soubor nenajde, tak se soubor vytvoří s výchozím nastavením, které je uložené v proměnné *default_setting*. Toto nastavení se také uloží do proměnné *setting* uvnitř třídy. Nastavení obsahuje data pro následující moduly:

- Serial
- Printer
- MQTT
- MES
- GUI
- System

Při jakékoli změně nastavení je potřeba v proměnné *setting* změnit data a následně zavolat metodu *update*, která tyto data uloží do souboru. Dále je možnost stáhnout konfiguraci z URL adresy. Pro stažení nastavení je vytvořena metoda *updateSettingFromUrl*, kde jako argument předáme URL adresu obsahující nastavení ve formátu JSON.

5.3 Modul pro skripty

Tento modul se stará o získávání a úpravu skriptů pro 3D tisk. Modul funguje na podobném principu, jako modul *Settings*, protože pro získání skript manažeru je potřeba použít funkci *GetScriptsManager*. Takže první modul, který zavolá tuto funkci, vytvoří objekt *Scripts* a při dalším zavoláním této funkce bude vrácen už vytvořený objekt.

Výpis 5.8: Získání skript manažera

```
_instance_script = None
def GetScriptsManager() -> Scripts:
  global _instance_script
  if _instance_script is not None:
    return _instance_script
  else:
    _instance_script = Scripts()
    return _instance_script
```

Při inicializaci objektu *Scripts* se zjišťuje, jestli existují soubory pro skripty. Jestliže soubory neexistují, tak se vytvoří ve složce "utils/scripts"v projektu. Tato třída má tři metody. První je *get_list_of_scripts*, která vrátí pole s názvy skriptů. Další metoda je *get_script*, kde je potřeba předat argument s názvem skriptu. V první řade funkce najde předaný název v poli skriptů a vytvoří cestu ke správnému souboru. Poté soubor program přečte a do návratové hodnoty předá celý řetězec znaků ze souboru. Pokud nastane jakákoliv chyba, tak se vytiskne do terminálu a vrátí se prázdný řetězec znaků.

Výpis 5.9: Získání skriptu

```
def get_script(self, name: str) -> str:
  g_code = ""
  list_index = 10
  for index, file_name in enumerate(self.script_list):
    split = file_name.split(".")
    if name == split[0]:
      list_index = index
    if list_index < len(self.script_list):
      try:
        file = open(os.path.join(SCRIPT_DIR,
                                 self.script_list[list_index]), "r")
        g_code = file.read()
        file.close()
      except OSError as ex:
        if ex.errno != errno.EEXIST:
          print(ex)
          return ""
        return g_code
      else:
        print("Chyba, název souboru nebyl nalezen.")
        return ""
```

Poslední metoda je *update_script*, kde opět předáváme argument s názvem skriptu a dále předáváme řetězec znaků, který se má do skriptu uložit. Začátek metody je identický jak při získávání skriptu. Jakmile je soubor nalezen, tak se otevře a celý skript je uložen do souboru.

5.4 Event manažer

O komunikaci mezi moduly se stará jednoduchý event manažer, který funguje na modelu public/subscribe. Nejprve je nutné se zaregistrovat k eventu. Registrace se dělá pomocí funkce *subscribe*. Tato funkce má dva argumenty. První argument je název eventu a druhý argument je funkce, která se při zavolání tohoto eventu vykoná.

Výpis 5.10: Event manažer

```
subscribers = dict()

def subscribe(event: str, fn):
  if event not in subscribers:
    subscribers[event] = []
  subscribers[event].append(fn)

def fire_event(event: str, data=None):
  if event not in subscribers:
    return
  for fn in subscribers[event]:
    if data is None:
      fn()
    else:
      fn(data)
```

Když je potřeba vyvolat událost je pro to vytvořená funkce *fire_event*. Tato funkce má opět dva argumenty. První argument je název události. Druhý argument jsou data, která se mají poslat. Pokaždé není potřeba předávat data, takže je možné tuto položku nechat prázdnou. Při zavolání této funkce se snaží program najít všechny funkce, které jsou zaregistrované k dané události a postupně každou zavolat s předaným argumentem. Protože je předáván pouze jeden argument ve funkci *fire_event*, tak doporučuji pro předávání dat využít datový typ *dictionary*.

5.5 MQTT modul

Modul pro posílání zpráv pomocí MQTT protokolu využívá knihovnu Paho od společnosti Eclipse. Tato knihovna poskytuje třídu *client*, která systému umožní připojit se k brokeru, publikovat zprávy, přihlásit se k odběru určitého tématu a přijímat publikované zprávy.

Pro využití v systému jsem vytvořil třídu *MQTT*, která při inicializaci spolupracuje s modulem *Settings*, pomocí ho které jsou získána počáteční nastavení klienta. Poté se přihlásí základní metody k event manažeru, aby bylo možné ovládat modul z grafického rozhraní pomocí metod *change_settings* a *connection*. Dále má třída přihlášený odběr na teplotu tiskárny a polohu tiskárny. Když přijde jedna ze zpráv, tak se ihned pošle na téma "printer/<název_tiskárny>/<temperature/position>".

Výpis 5.11: MQTT modul

```
class MQTT(object):                                                    1
  def __init__(self):                                                 2
    self.settings = GetSettingsManager()                              3
    self.IP_address = self.settings.setting["MQTT"]["IP.."]           4
    ... základní nastavení                                            5
    subscribe("temperature_update", self.send_temperature)            6
    subscribe("position_update", self.send_position)                  7
    subscribe("MQTT_connection", self.connection)                     8
    subscribe("MQTT_settings", self.change_settings)                  9
                                                                      10
  def send_temperature(self, data):                                   11
    #ukázka posílání zpráv s teplotou                                 12
    if self.client.is_connected():                                    13
      self.client.publish/                                            14
      (f"printer/{self.printer_name}/temperature", str(data))         15
                                                                      16
  def change_settings(self, data: dict):                              17
    #ukázka změny nastavení                                           18
    if data.get("ip_address") is not None:                            19
      self.IP_address = data["ip_address"]                            20
      self.settings.setting["MQTT"]["IP_address"] = self.IP..         21
      self.settings.update()                                          22
    if data.get("port") is not None:                                  23
      self.port = int(data["port"])                                   24
      self.settings.setting["MQTT"]["port"] = self.port               25
      self.settings.update()                                          26
```

V případě metody *change_settings* je potřeba předat data v datovém typu *dictionary*, aby bylo možné předat všechna důležitá data v jedné proměnné.

Při připojení nebo odpojení klienta od brokera se pošle zpráva ostatním modulům a informuje je o stavu tiskárny pomocí datového typu *bool*, kdy *true* indikuje připojený stav a *false* indikuje odpojený stav.

5.6 Automatický systém

Jedním s požadavků na systém byla komunikace s MES systémem a autonomní řízení. První část komunikace tvoří MQTT a druhou část komunikace tvoří stavový automat, který posílá požadavky na MES. Stavový automat bude zadávat požadavek na tisk a postará se o automatické odebrání výtisku z podložky.

Stavový automat má dohromady 4 stavy. První stav je nečinnost, kdy bude automatický systém vypnutý a čeká na požadavek zapnutí. V tomto stavu využívám třídu *event* z knihovny *thread*, která zajistí, že v tomto stavu setrvá dokud nedostane signál o pokračování.

Výpis 5.12: Stavový automat stav: nečinnost

```
def Idle_state(self):
  print("Idle_state")
  if self._status:
    if self._comm_status:
      self._state = States.REQUEST
      return
  self._next_status.wait() #čekání na signál
  self._next_status.clear()
```

Druhý stav automatu je poptávka. V tomto stavu systém posílá požadavek na MES systém, jestli nemá pro tiskárnu práci. Tento požadavek je posílán každých 5 sekund.

Výpis 5.13: Stavový automat stav: poptávka

```
def Request_state(self):
  print("Request_state")
  file = self._getFileFromQueue(self._MES_url)
  if file == "":
    time.sleep(5)
    return
  fire_event("printer_start_print", file)
  print(f"Tisk␣souboru␣{file}")
  self._state = States.PRINTING
```

MES by měl mít vytvořenou REST API, aby automatický systém na určené URL adrese dostal po požadavku typu GET odpověď, kde bude URL adresa pro stažení souboru na tisk a ID práce. Pro URL jsou dvě možnosti jak jej poslat. Pokud se soubor nachází na stejném serveru, tak stačí poslat URL adresu ve tvaru: "localhost/<cesta k souboru". Pokud se soubor nenachází na stejném serveru, tak je potřeba poslat celou URL adresu. Jakmile systém soubor stáhne, tak ho uloží jako "job.gcode". Poté pošle event manažeru, aby dal modul *Printer* tisknout stažený soubor. Jakmile začne tisk, pošleme pomocí metody POST následující zprávu ve formátu JSON na stejnou URL adresu, ze které jsme získali první odpověď.

Výpis 5.14: Zpráva pro metodu POST

```
{
  "id":  self._job_id,
  "status": "printing",
  "printer": self.printer_name,
  "MQTT": f"/printer/{self.printer_name}"
}
```

Po této zprávě se stavový automat dostane do stavu tisknutí, ve kterém čeká, dokud tisk není dokončen. Jakmile systém dostane zprávu, že tisk je hotový, tak stavový automat pošle MES systému zprávu metodou POST, ve stejném tvaru jako je ve výpisu 5.15, jen s tím rozdílem, že v položce *status* bude "done". Poté smaže soubor s G-kódem a stavový automat se dostane do stavu "odstranění".

Výpis 5.15: Stavový automat stav: tisk

```
def Printing_state(self):
  print("printing_state")
  self._next_status.wait()
  self._state = States.REMOVAL
  self._next_status.clear()
  requests.post(self._MES_url,
                json={"id": self._job_id,
                "status": "done",
                "printer": self.printer_name})
  if os.path.exists("job.gcode"):
    os.remove("job.gcode")
    print("The␣file␣has␣been␣deleted␣successfully")
  else:
    print("The␣file␣does␣not␣exist!")
```

Stav odstranění řeší odejmutí výrobku z tiskové plochy. Pro odejmutí jsou zde dva módy a to manuální a automatický. Pokud je systém nastaven na manuální

mód, tak se v grafickém rozhraní ukáže výzva na odejmutí vytisknutého objektu z tiskové plochy. Jakmile na obrazovce odklikneme tlačítko "OK", tak se automat dostane do stavu nečinnosti. Pokud je nastaven systém na automatické odebrání, tak pošle požadavek přes event manažera, aby byl použit skript pro automatické odstranění výrobku z tiskové plochy. Jakmile se skript dokončí, tak systém dostane signál, že může přejít do počátečního stavu.

Výpis 5.16: Stavový automat stav: odstranění

```
def Removal_state(self):
  if self._removal_mode == "manual":
    fire_event("GUI_removal_dialog", None)
  elif self._removal_mode == "auto":
    fire_event("printer_auto_removal", None)
  print("Removal_state")
  self._next_status.wait()
  self._state = States.IDLE
  self._next_status.clear()
```

5.7 GUI

Grafické uživatelské rozhraní jsem řešil pomocí QML neboli Qt Modeling Language. Tento modelovací jazyk byl vytvořen společností QT a na jeho implementaci v Pythonu je potřeba knihovna PySide 2. Pro vytvoření grafického rozhraní jsem využil program QT Design studio, které funguje jako vývojové prostředí pro snadnější vytvoření pěkného GUI.

Grafické rozhraní jsem rozdělil na 6 obrazovek a přizpůsobil pro dotykovou obrazovku s rozlišením 1024x600. Jako první obrazovku jsem vytvořil hlavní menu. V hlavním menu lze otevřít sériovou komunikaci s 3D tiskárnou. Když je komunikace otevřená, tak se otevřou další 3 možnosti v menu.

První možnost je kontrolní panel 3D tiskárny. V kontrolním panelu je možné ovládat jednotlivé krokové motory, rychlost posuvu extruderu, otáčky ventilátoru a v neposlední řade je tu možnost poslat jakákoliv příkaz, který napíšete.

Druhá možnost je kontrolní panel teploty 3D tiskárny. V tomto panelu se zobrazí teploty extruderů, teplota vyhřívané podložky a jako poslední teplota komory.

Poslední možnost, která se nám otevře je panel tisku. Na tomto panelu lze vybrat soubor pomocí souborového dialogu. Poté lze zapnout tisk, pozastavit tisk nebo ukončit tisk. Na tomto panelu lze posunout obrazovku doprava, kde najdeme editor skriptů.

Na hlavní obrazovce najdeme dále panel pro nastavení. V tomto panelu lze nastavit sériovou komunikaci. Je možné nastavit port, baudrate a automatické připojování při startu systému. Jako další je zde nastavení intervalu hlášení pozice a teploty. Jako poslední věc zde najdeme nastavení pro maximální teplotu extruderů, vyhřívané podložky a komory. Toto nastavení se promítá pouze do grafického rozhraní, kde podle nastavené hodnoty půjde v panelu teplot nastavit maximální hodnota jednotlivých položek. V panelu nastavení lze také posunou obrazovku doprava. Zde najdeme nastavení MQTT komunikace, URL adresy pro MES a nastavení automatického systému.

Posledním tlačítkem v hlavním menu vypneme celý program. Když na toto tlačítko klikneme, tak se objeví dialog, jestli opravdu chceme vypnout program.

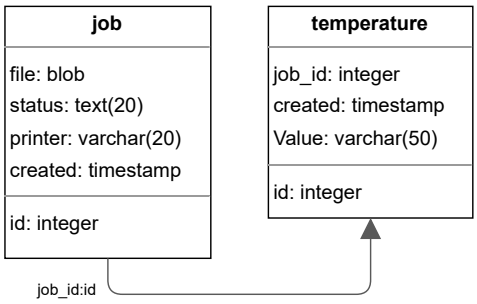
Ukázku všech panelů lze nalézt v příloze A.

6 Testování řešení

V následující kapitole popíši postup při testování mého řešení řídicího systému. Nejprve popíšu vytvoření testovacího programu pro autonomní režim a poté samotné testování systému, kde se budu zabývat testováním tisku a následně autonomního režimu.

6.1 Simulace MES

Pro otestování autonomního režimu naprogramovaného řídicího systému jsem vytvořil simulaci MES systému. Tento systém je napsán opět v programovacím jazyku Python. Protože MES je webová aplikace, tak jsem pro jeho vytvoření použil webový framework Flask a jako databázi jsem použil SQLite. Schéma databáze vypadá následovně.



Obr. 6.1: Schéma tabulek v SQLite použité v simulaci MES

MES má pouze jednu HTML stránku na které lze nahrát soubor pro tisk. Poté co se soubor nahraje a přidá do databáze, tak je možné poslat požadavek s řídicího systému 3D tiskárny. Komunikace je napsaná jako RESTful API a posílá odpovědi v JSONu. Tato komunikace byla popsána v kapitole 5.6.

Výpis 6.1: MES URL cesta pro upload

```
@app.route('/')
  def upload_files():
    if request.method == 'POST':
      #soubor se přidá do databáze
      #vrátí opět HTML stránku pro upload
      return render_template('up.html')
    if request.method == 'GET':
      return render_template('up.html')
```

Výpis 6.2: MES URL cesta pro zakázku a config

```
@app.route("/printer/config")
  def config():
    return default_setting
@app.route("/printer/queue")
  def queue():
    if request.method == 'POST':
      #přijme zprávu od řídicího systému a updatuje databázi.
    if request.method == 'GET':
      #vrátí JSON odpověď, kde se nalézá ID zakázky a
      #URL pro stažení souboru
@app.route("/printer/job/<int:id>/download")
  def download_file(id: int):
    #vrátí soubor (dojde ke stažení souboru klientem)
    return send_from_directory(soubor)
```

6.2 Testování systému

6.2.1 Tisk

Jako první jsem otestoval tisk pomocí mého systému. Při prvních testování se tisk někdy zasekl a systém přestal posílat příkazy. Tento problém byl způsoben tím, že se v systému objevoval souběh (anglicky race condition). Tato chyba se objevovala mezi vlákny na monitorování zásobníku tiskárny a odesílatelem příkazů, kdy než se stihla zpracovat celá iterace poslání příkazu, tak monitorovací vlákno už zachytilo odpověď "ok"od tiskárny. Ve výpisu 6.3 na řádku 14 je část kódu, ke kterému když program nestihl dojít než se odbavila odpověď od tiskárny, tak se program zasekl.

Výpis 6.3: Problémová část kódu

```
if not self._pause:
  try:
    command = self._command_to_send.get()
  except queue.Empty:
    continue
  try:
    command_to_send = G_Command_with_line(command, n_line)
    self._comm.write(command_to_send.process())
    self._comm.flush()
    n_line += 1
  finally:
```

```
    self._command_to_send.task_done()
with self.condition:
  self.condition.wait()
```

Tento problém jsem vyřešil tím, že místo objektu *Condition* z knihovny *threading* jsem použil objekt *Event* ze stejné knihovny. To mi dovolilo lépe ovládat čekání na odpověď, protože tento objekt dovoluje vypnout čekání pomocí funkce *event.set()* a znovu zapnout pomocí funkce *event.clear()*. Takže v tuto chvíli, když dojde od monitorovacího vlákna signál, že může být poslán další příkaz, tak začne další iterace v odesílacím vlákně. Jako první program v této iteraci zpracuje příkaz, ale než ho odešle po sériové komunikaci, tak si opět zapne čekání na funkci *event.wait()* pomocí funkce *event.clear()*. Právě touto posloupností funkcí docílím toho, že už nebude nastávat souběh v programu.

Výpis 6.4: Opravená část kódu

```
try:
  command = self._command_to_send.get()
  command_to_send = G_Command_with_line(command, n_line)
  self.event.clear()
  self._comm.write(command_to_send.process())
  self._comm.flush()
  n_line += 1
  except queue.Empty:
    continue
  except Exception as ex:
    fire_event("Serial_ERROR", ex)
    print(f"Naskytla␣se␣chyba␣{ex}")
    self.disconnect()
  finally:
    self._command_to_send.task_done()
self.event.wait()
```

6.2.2 Automatický systém

Jakmile jsem měl otestovaný tisk, tak jsem mohl otestoval automatický systém pomocí naprogramované simulace MES systému. Pro testování jsem si vybral model kostky. V grafickém rozhraní systému jsem zapnul automatický systém a odstranění vytisknutého objektu jsem nechal na manuálním režimu, protože má tiskárna nedisponuje touto funkcí.

Systém funguje dle očekávání, po připojení k tiskárně se stavový automat přepnul do režimu poptávky a začal posílat na MES požadavky pro práci. Jakmile dostal

odpověď s URL adresou a identifikačním číslem práce, tak si systém soubor stáhl a začal tisknout. Ještě před tiskem dostal MES systém odpověď od řídicího systému, že práce s touto ID je ve stavu tisknutí.

Po vytisknutí objektu došla zpráva, že práce je hotová a stavový automat přešel do režimu odstranění. V grafickém rozhraní se objevila výzva na odstranění vytisknutého objektu, takže jsem objekt odstranil z tiskové plochy a na displeji tuto výzvu potvrdil. Systém hned po potvrzení požádal o další práci, kterou zase začal provádět.

6.3 Zhodnocení výsledku

Z testování řídicího systému jsem zjistil, že systém dokáže fungovat v autonomním režimu bez jakéhokoli zasekávání programu. Systém si dokáže zažádat o práci a při tom posílá pomocí MQTT protokolu zprávy s teplotou a polohou tiskárny. Právě tyto informace by mohli být využity pro virtuální dvojče pro 3D tiskárnu.

Při dalším vývoji bych chtěl zlepšit sériovou komunikaci, kdy bych chtěl zkusit pracovat s velikostí zásobníku 3D tiskárny aniž bych zpomaloval posílání příkazu čekáním na odpověď "ok". Touto komunikací lze zlepšit kvalitu tisku při rychlejším tisknutí. V momentální verzi programu nemůže nastat situace, kdy by mi tiskárna poslal žádost, ať pošlu znovu příkaz, protože vždy čekám na odpověď tiskárny.

Závěr

Zadáním bakalářské práce bylo navrhnout řídicí systém pro 3D tiskárnu, která umožní její činnost v autonomním režimu. V úvodu práce jsem se seznámil s koncepty průmyslu 4.0, abych měl představu, jak bych měl řídicí systém navrhnout.

Abych mohl začít vyvíjet řídicí systém, tak bylo zapotřebí nadefinovat požadavky, které od řídícího systému očekávám. Hlavní požadavkem na systém bylo autonomní řízení. Dále bylo zapotřebí, aby systém dokázal komunikovat s MES systémem a 3D tiskárnou, kterou systém řídí. Protože je nezbytné, aby uživatel mohl systém také ovládat, tak jsem se rozhodl o vytvoření GUI pro dotykovou obrazovku.

Při výběru HW platformy jsem se rozhodl pro jednodeskový počítač Raspberry Pi, který je cenově dostupný a jeho výkon je dostačující běh programu. Na Raspberry Pi byl operační systém Raspbian. Jako programovací jazyk jsem zvolil Python, protože nabízí moderní paradigmata pro programování a je dostupný na všech operačních systémech.

Systém jsem si rozdělil na jednotlivé moduly a graficky vytvořil strukturu systému. Vytvořil jsem si tak představu, jak jednotlivé bloky spolu budou komunikovat.

V další částí jsem řešil programové řešení řídicího systému, kde jsem popsal všechny částí programu jak fungují. Nejvíce jsem se zabýval sériovou komunikací s 3D tiskárnou a autonomním systémem, který funguje na modelu stavového automatu.

Závěr práce jsem věnoval testování systému a zhodnocení dosažených výsledků. Systém pro 3D tiskárny je funkční a požadavek na autonomní rozhodování systému byl splněn.

Literatura

[1] *Krejčí, J. AND Ambler, M. Průmysl 4.0: národní, firemní a akademické přístupy. Současná Evropa, 2017, vol. 2017, iss. 2, p. 46-62.* [online]. [cit. 21. 12. 2020]. Dostupné z URL: <https://sev.vse.cz/artkey/sev-201702-0003_prumysl-4-0-narodni-firemni-a-akademicke-pristupy.php>.

[2] *Christoph, R Industry 4.0*[foto] [online]. [cit. 21. 12. 2020]. Dostupné z URL: <https://commons.wikimedia.org/wiki/File:Industry_4.0_(cs).png#/media/File:Industry_4.0_(cs).png>.

[3] *KACZMARCZYK, V.; BAŠTÁN, O.; BRADÁČ, Z.; ARM, J., 2018: An Industry 4.0 Testbed (Self-Acting Barman): Principles and Design. , p. 163 - 6* [online]. [cit. 22. 12. 2020]. Dostupné z URL: <https://www.sciencedirect.com/science/article/pii/S2405896318309108?via%3Dihub>.

[4] VUT v Brně: *Pásek,J. Digitální transformace průmyslu* [online]. [cit. 23. 12. 2020]. Dostupné z URL: <https://www.vutbr.cz/www_base/priloha_fs.php?dpid=183201&skupina=dokument_priloha>.

[5] VUT v Brně: *Kaczmarczyk,V. Prezentace Průmysl 4.0* [online]. [cit. 23. 12. 2020]. Dostupné z URL: <https://www.vutbr.cz/www_base/priloha_fs.php?dpid=185233&skupina=dokument_priloha>.

[6] *Ilanković, Nikola AND Živanić, Dragan AND Zelić, Atila AND Miklós, Gubán AND Szabo, Laszlo. (2019). Basic principles of Industry 4.0 as the foundation for smart factories and digital supply networks.* [online]. [cit. 25. 12. 2020]. Dostupné z URL: <https://www.researchgate.net/publication/337032679_Basic_principles_of_Industry_40_as_the_foundation_for_smart_factories_and_digital_supply_networks>.

[7] *Nová infrastruktura inteligentního průmyslu: Smart Industry a ERP* [online]. [cit. 26. 12. 2020]. Dostupné z URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.

[8] *Bulletin Průmyslu 4.0* [online]. 07/2018 [cit. 27. 12. 2020]. Dostupné z URL: <https://www.ncp40.cz/files/bulletin-prumyslu4-2018-07.pdf>.

[9] *What is a REST API?* [online]. 3/2020 [cit. 15. 4. 2022]. Dostupné z URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.

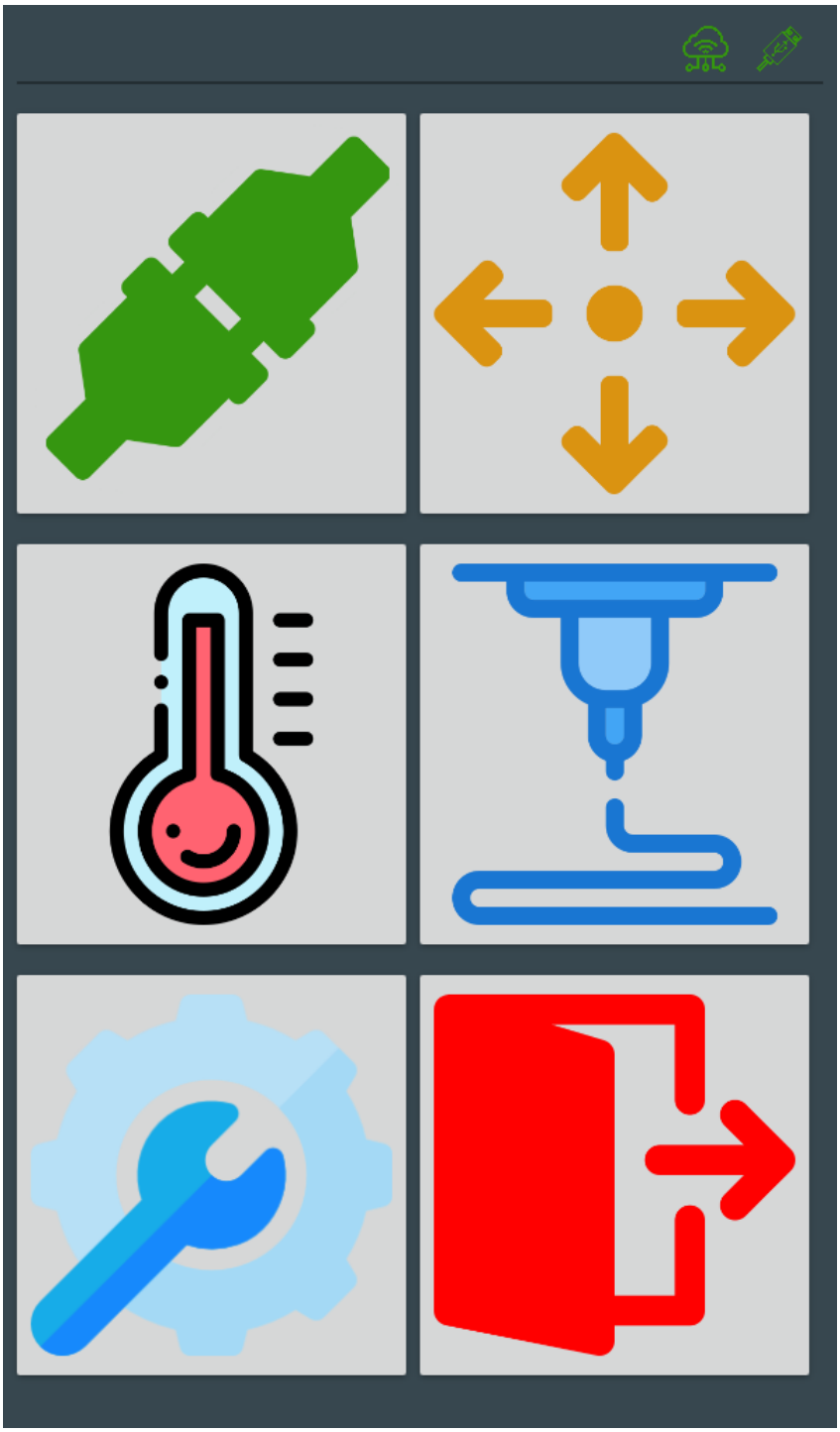
[10] *Lokesh Gupta. What is REST* [online]. 04/2022 [cit. 15. 4. 2022]. Dostupné z URL: <https://restfulapi.net/>.

[11] *Martin Malý, Protokol MQTT: komunikační standard pro IoT* [online]. 06/2016 [cit. 16. 4. 2022]. Dostupné z URL: <https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/>.

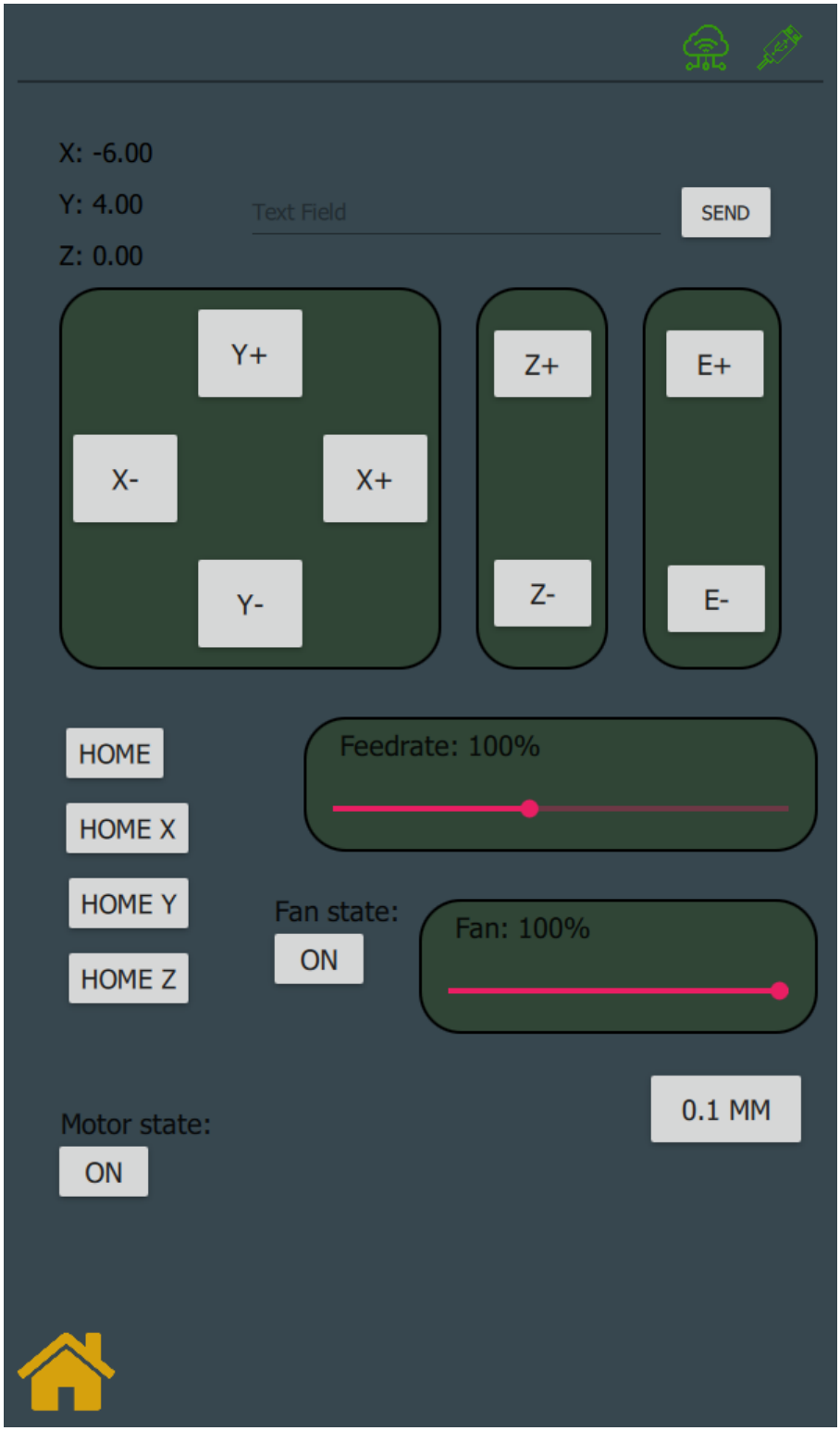
[12] *Corinne Bernstein, Kate Brush, Alexander S. Gillis, MQTT (MQ Telemetry Transport)* [online]. 01/2021 [cit. 16. 4. 2022]. Dostupné z URL: <https://www.techtarget.com/iotagenda/definition/MQTT-MQ-Telemetry-Transport>.

A Ukázka obrazovek v GUI

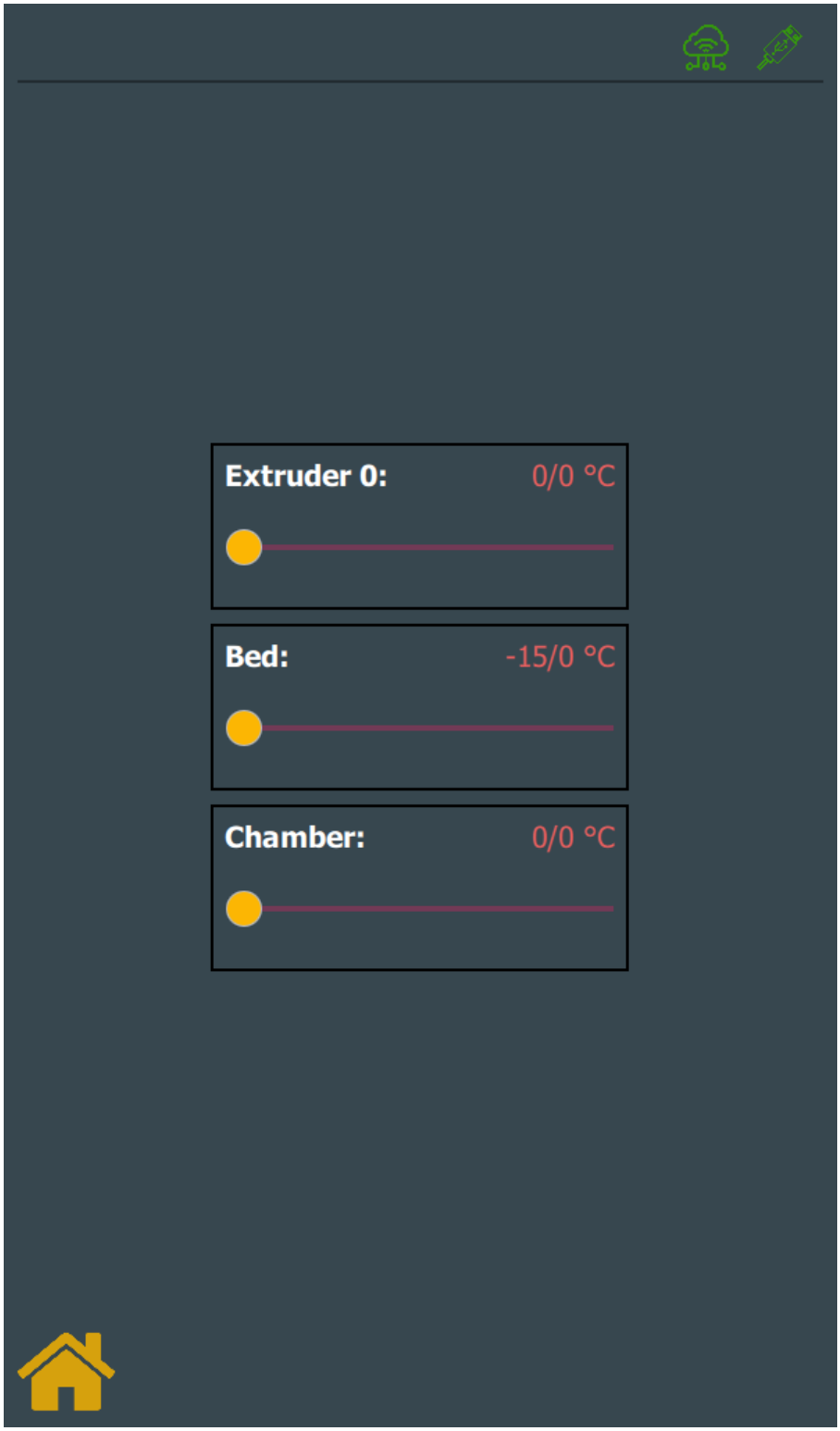
V této příloze se nachází obrázky jednotlivých panelů grafického rozhraní.



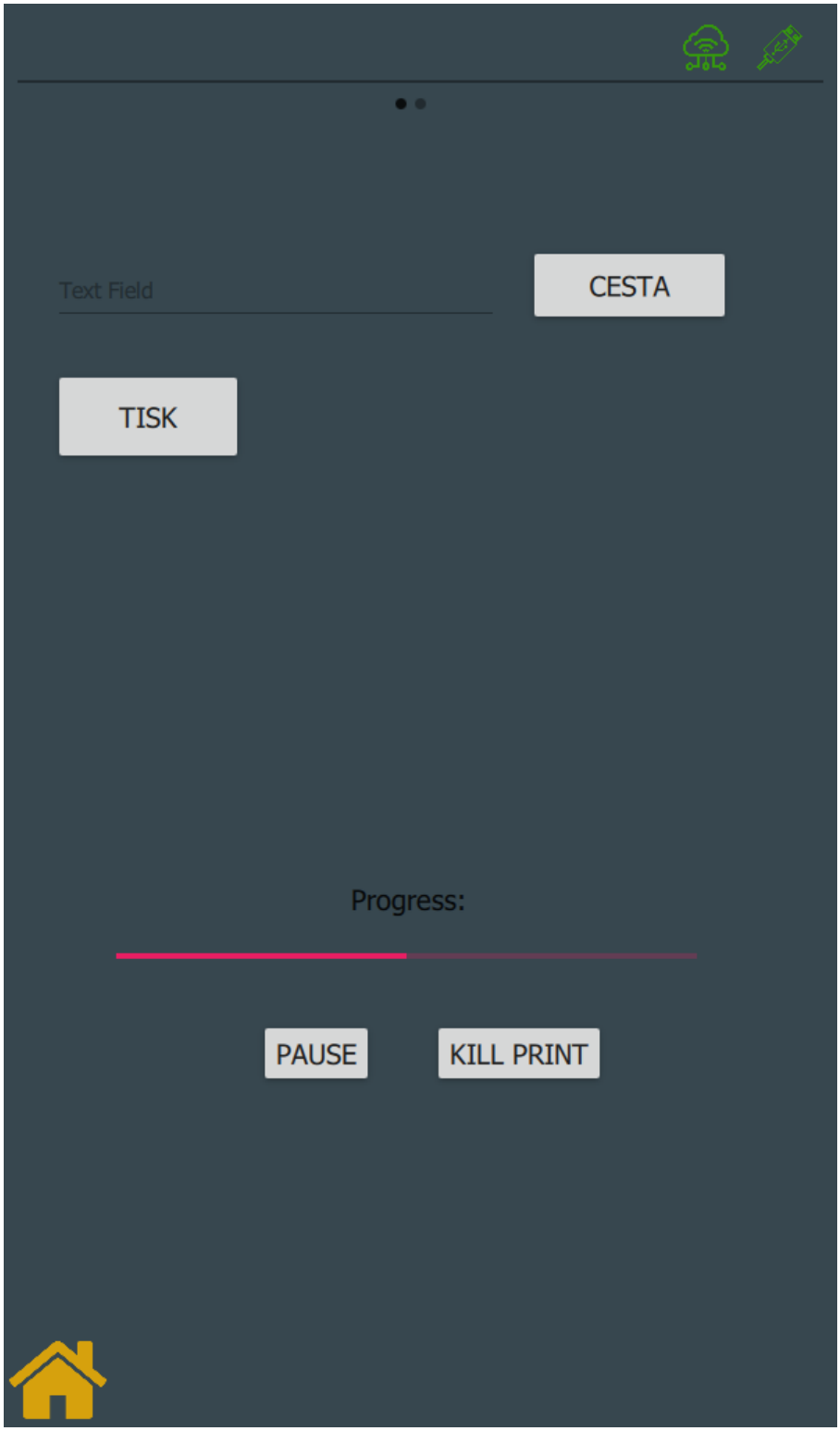
Obr. A.1: Ukázka obrazovky hlavního menu



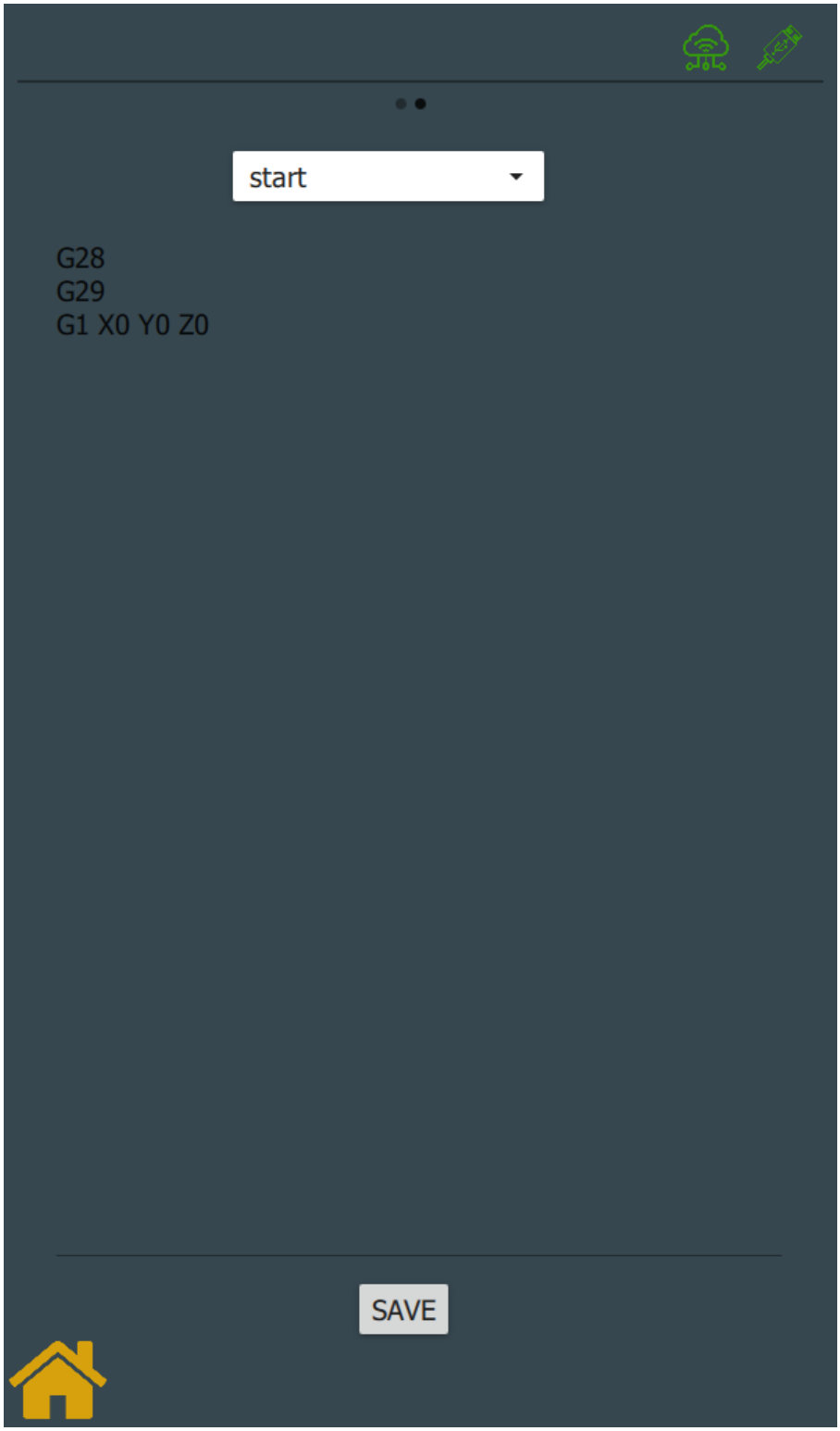
Obr. A.2: Ukázka obrazovky kontrolního panelu



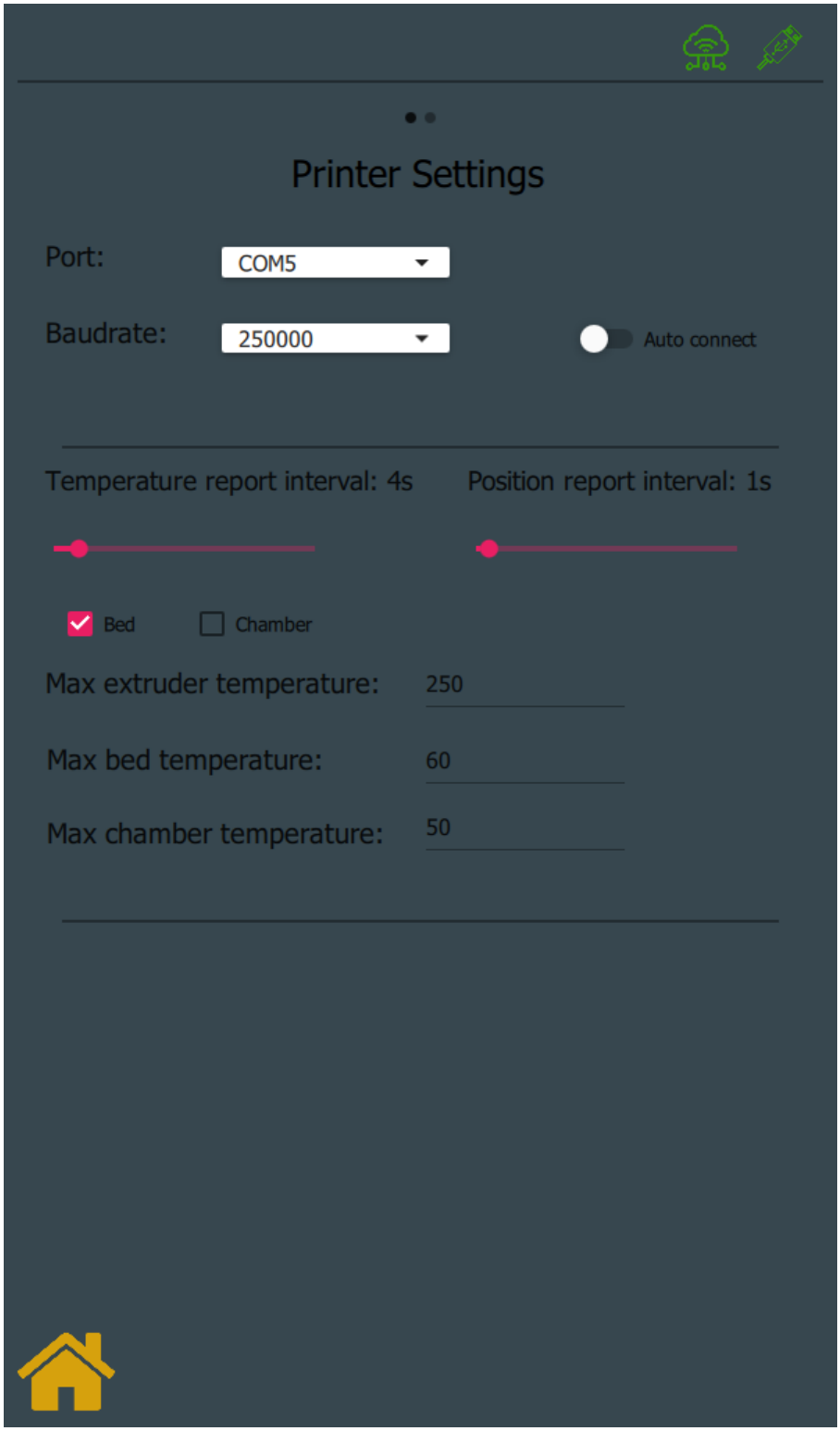
Obr. A.3: Ukázka panelu s teplotou



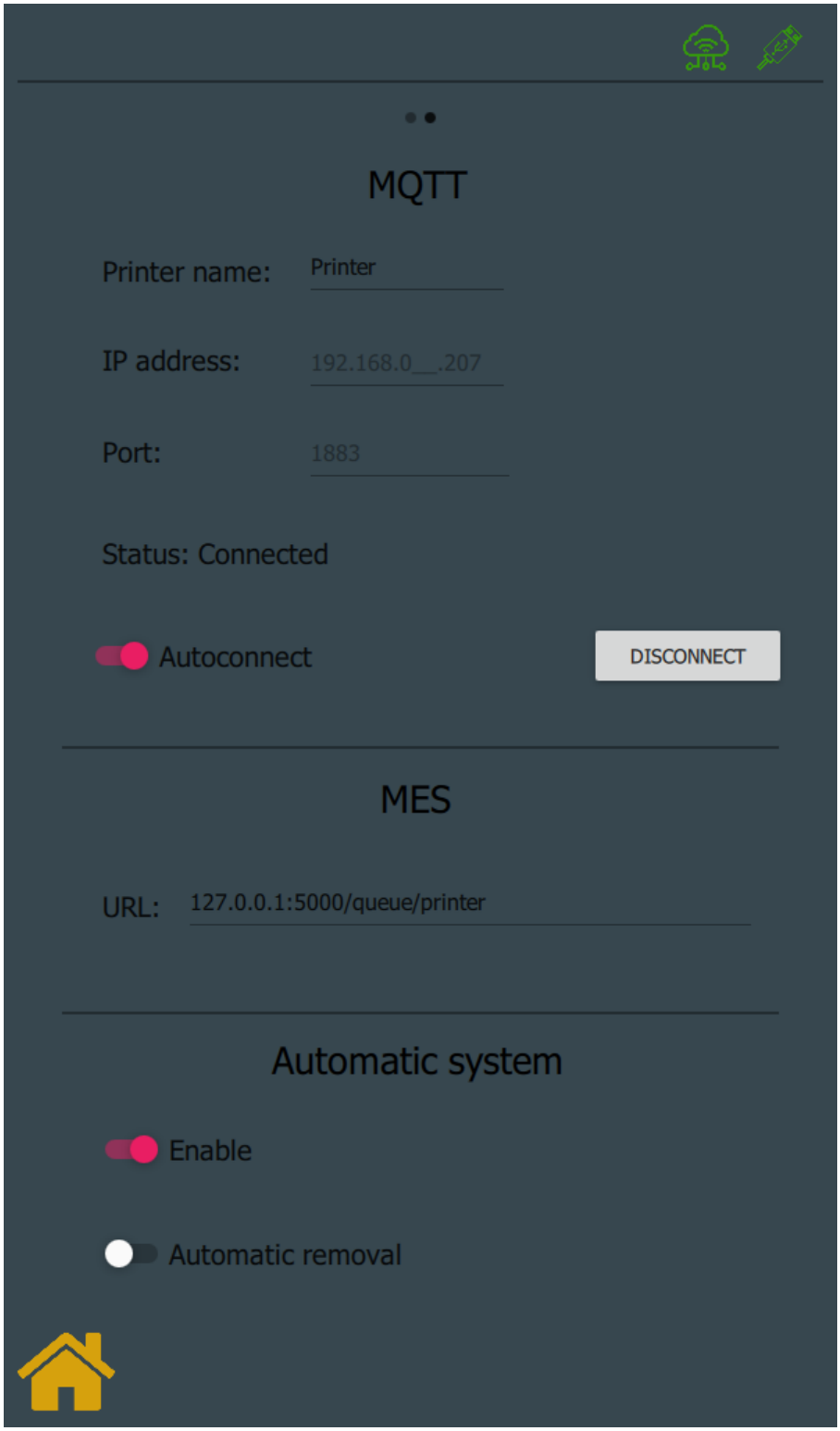
Obr. A.4: Ukázka panelu pro tisk



Obr. A.5: Ukázka panelu pro úpravu skriptu



Obr. A.6: Ukázka obrazovky panelu s nastavením



Obr. A.7: Ukázka obrazovky panelu s nastavením

B Obsah přiloženého CD

- `/` ..... Kořenový adresář.
  - `Bakalarska_prace` ..... Textová část bakalářské práce.
  - `Programove_reseni`
    - `ridici_system` ..... Řídicí systém pro 3D tiskárnu.
    - `simulace_MES` ..... Simulace MES systému pro testování.
    - `readme.txt` ..... Soubor s návodem pro zapnutí obou systému.