VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV VÝKONOVÉ ELEKTROTECHNIKY A ELEKTRONIKY
DEPARTMENT OF POWER ELECTRICAL AND ELECTRONIC ENGINEERING

TVORBA GUI PRO OPTIMALIZACI ELEKTRICKÝCH STROJŮ
CREATION OF GUI FOR OPTIMIZATION OF ELECTRICAL MACHINES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Tereza Bártková**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Ladislav Knebl**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Silnoproudá elektrotechnika a elektroenergetika**

Ústav výkonové elektrotechniky a elektroniky

Studentka: Tereza Bártková ***ID:*** 203499

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Tvorba GUI pro optimalizaci elektrických strojů

POKYNY PRO VYPRACOVÁNÍ:

1. Proveďte návrh grafického prostředí ve zvoleném jazyce.
2. Vybraný typ motoru pomocí navrženého algoritmu s grafickým prostředím optimalizujte
3. Porovnejte původní a optimalizovaný motor dle kritérií zvolených při optimalizaci.

DOPORUČENÁ LITERATURA:

[1] J. Pyrhonen, J. Jokinen, V. Hrabovcova "Design of Rotating Electrical Machines" John Wiley & Sons, Ltd, 2008. 512 s. ISBN 978-0-470-69516-6(H/B)

[2] A. E. Fitzgerald, Ch. Kingsley, S. Umans "Electric Machinery", McGrow-Hill Companies Inc., 2003. 688 s. ISBN 0-07-112193-5

[3] Cigánek, L., Bauer, M.: Elektrické stroje a přístroje

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 27.5.2021

Vedoucí práce: Ing. Ladislav Knebl

doc. Ing. Petr Toman, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce sleduje vznik grafického uživatelského prostředí od výběru nástrojů po realizaci. Smyslem bylo udělat zadaný program přívětivější k používání vytvořením reprezentativní komunikační vrstvy mezi programem a uživatelem a ukázat jeho funkce. Nejprve byl proveden rozbor různých programovacích jazyků a jejich nástrojů pro tvorbu grafických rozhraní. Z těchto byl vybrán jazyk Python, protože zadaný program je napsán v Pythonu, navíc vývoj aplikací v něm probíhá patrně nejefektivněji. Dále jsou rozebírány vlastnosti tohoto jazyka a PyQt, což je nástroj, který je možné využít k samotné tvorbě grafického rozhraní. Třetí kapitola je věnována problematice elektrických strojů a jejich optimalizaci. Jsou zde představeny některé algoritmy v praxi používané pro zlepšení charakteristik elektromotorů. V následující kapitole je nastíněn zadaný typ motoru a popsán konkrétní stroj, pro který je výsledný program určen. Poté je podrobně popsán layout aplikace realizovaný v Qt Designeru. Je vysvětlena nová struktura programu obohacená o grafické prostředí a popsán chod programu. Nakonec je vysvětleno jak program použít, je provedena optimalizace zadaného stroje a jsou interpretovány výsledky.

KLÍČOVÁ SLOVA

GUI, Qt Designer, Python, PMSM, optimalizace, SOMA

ABSTRACT

This work follows the development process of graphical user interface (GUI) spanning from choosing the right tools for implementation to the implementation itself. The purpose of the work was making the optimization program user-friendly by creating a communication layer between the optimization algorithm and the user and demonstrating its functions. First of all, the study of various programming languages and their toolkits regarding GUI programming was undertaken. Python was chosen because the original program is written in it, also software development in Python is probably the quickest one. Furthermore there are analyzed characteristics of this language and PyQt, which is a tool meant for GUI creation. The third chapter introduces very thoroughly the issues of electrical machines and their optimization. Introduction of these issues is followed by brief description of two optimization methods being used in electrical machines optimization. Following chapter introduces the actual machine under the optimization study, on which the final program was tested. Afterwards follows detailed description of application layout created in Qt Designer. New program structure enriched with graphical user interface is described and its functions are shown. Finally there is a guideline on how to use this program and the optimization is executed accordingly. Also, the results of the optimization are analyzed.

KEYWORDS

GUI, Qt Designer, Python, PMSM, optimization, SOMA

BÁRTKOVÁ, Tereza. *Tvorba GUI pro optimalizaci elektrických strojů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav výkonové elektrotechniky a elektroniky, 2021, 51 s. Bakalářská práce. Vedoucí práce: Ing. Ladislav Knebl

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Tereza Bártková
VUT ID autora:	203499
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Tvorba GUI pro optimalizaci elektrických strojů

Prohlašuji, že svou závěrečnou práci jsem vypracovala 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 autorka uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma 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 autorky*

*Autor podepisuje pouze v tištěné verzi.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Elektromotory jsou dnes pro své vlastnosti využívány natolik, že je lze najít téměř všude, kde je civilizace. Mezi jejich výhody patří například cena, absence emisí nebo snadná údržba. Pro šíři možností jejich použití je věnována značná pozornost jejich výzkumu a vývoji, který vede ke zvyšování jejich účinnosti a snižování výrobních nákladů. V tomto kontextu vznikla dnes již rozšířená skupina synchronních motorů s permanentními magnety, která je jedním z předmětů této práce. Prostředkem zlepšování vlastností těchto elektrických strojů je optimalizace, která může využívat různých přístupů. V této práci je použit samo-organizující migrující algoritmus, který byl vypozorován u divoce žijících smeček.

Pro účely optimalizace synchronního motoru s permanentními magnety pomocí samo-organizujícího migrujícího algoritmu existuje program zahrnující v sobě odpovídající funkcionalitu, avšak postrádající její snadné zpřístupnění pomocí grafického rozhraní. Na prezentační vrstvě aplikace přitom závisí i následná využitelnost programu s ohledem na její výsledný vzhled a přívětivost k uživateli.

Smyslem této práce je tedy vytvořit pro výše zmíněný program takové grafické prostředí, které bude pro uživatele přehledné a učiní tak program jako celek snadno použitelným. Cílem nového grafického prostředí je snadné zadávání parametrů pro optimalizaci a umožnění pohodlné interpretace výsledků. Jeho funkčnost bude možné demonstrovat na optimalizaci zadaného elektromotoru, jejíž výstupem by měl být model o lepších parametrech než model výchozí.

V práci je použito velkého množství pojmů z různých technických oblastí, zvláště z programování. Kvůli lepší čitelnosti práce a pochopení smyslu jsou právě pojmy související s programováním (názvy zdrojových souborů, tříd, metod aj.) zvýrazněny použitím strojového písma.

1 Programovací jazyky v tvorbě grafického uživatelského rozhraní

Smyslem této kapitoly je představit některé populární programovací jazyky a zvážit jejich vhodnost pro účel splnění praktické části této práce. Před samotným rozborem bude provedeno vysvětlení několika pojmů, které jsou podstatné pro pochopení problematiky tvoření GUI a výběru frameworku:

GUI – zkratka značí spojení „grafické uživatelské rozhraní“. Je nástupcem rozhraní příkazové řádky a textového uživatelského prostředí, které byly nahrazeny grafickým rozhraním pro jeho přívětivost k uživatelům. Pro grafické uživatelské rozhraní je typické, že interakce mezi uživatelem a zařízením probíhá pomocí tzv. widgetů. Jsou to grafické ovládací prvky jako ikony, tlačítka, posuvníky aj., se kterými uživatel komunikuje myší a klávesnicí. GUI tvoří jakousi srozumitelnější vrstvu mezi programem a uživatelem, díky které je program intuitivnější co se týče jeho ovládání a používání.

Programovací jazyk – standardizovaný nástroj pro komunikaci s počítačem. Každý jazyk má vlastní soubor klíčových slov (takových, která mají zvláštní funkcionalitu) a gramatických pravidel pro instruování počítače k vykonání určitého úkolu. Škála vhodných aplikací se různí v závislosti na daném jazyku.

Framework – podpůrná struktura programu. Jedná se o znovupoužitelný kód poskytující určitou funkcionalitu, která může být využita k vytvoření aplikace. Frameworky vznikají za účelem urychlení a zjednodušení psaní různých druhů programů, například webových aplikací, grafických uživatelských rozhraní, her aj. Na rozdíl od knihoven ovlivňují řídicí strukturu program.

Binding – tzv. *wrapper* umožňující využití frameworku či knihovny i v jiném jazyce, než v jejich nativním. Aby framework mohl být použit v nějakém jiném programovacím jazyce, než pro který je určen, je nutno použít binding daného jazyka.

Pro tvorbu GUI existuje řada nástrojů (frameworků, knihoven) lišících se od sebe jednoduchostí použití, designem, portabilitou a dalšími charakteristikami, které budou dále rozebírány. Frameworky mohou být použitelné napříč různými programovacími jazyky, tzn. je-li například framework napsaný nativně v C++, mohou k němu existovat bindingy, díky kterým je možné jej použít i v jiných jazycích. Níže jsou stručně popsány programovací jazyky, které jsou podle [1] nejpoužívanější. Spolu s nimi jsou charakterizovány frameworky napsané v těchto jazycích.

1.1 C/C++

C je již dlouho a široce používaný programovací jazyk. Jako jediný ze zde rozebraných jazyků je kompilovaný (jeho zdrojový kód je přeložen do binárního), díky čemuž je chod programů rychlý. Tuto vlastnost lze využít při renderování grafiky, čehož se využívá v případě C++ při programování her. Tento jazyk je podle [2] nejpoužívanějším jazykem ve vývoji operačních a embedded systémů. Je oblíbenou volbou pro programování elektroniky, robotů a jiného hardwaru a to právě proto, že je kód kompilován do binárního jazyka, který je hardwaru blízký. O využití operační paměti (alokaci a uvolnění) rozhoduje programátor při psaní kódu. Výhoda této skutečnosti tkví v možnosti maximální optimalizace kódu. Nevýhodou je, že takto často a snadno vznikají chyby. Pro tvoření GUI je tomto jazyce k dispozici množství frameworků, které jsou napsány přímo v C nebo C++. Níže jsou detailněji rozebrány frameworky Qt, wxWidgets, GTK a Tk.

C a C++ jsou dva různé (byť úzce spjaté) jazyky. C je procedurální programovací jazyk (popisující řešení daného problému posloupností algoritmů). C++ je upravený jazyk C podporující navíc třídy a objekty. Ty představují základní prvky objektově orientovaného programování, které umožňují dědičnost atributů mezi třídami a polymorfismus, díky kterému podtřídy odpovídají na stejnou metodu různě.

1.1.1 Qt

Mezi výhody použití Qt frameworku patří jeho multiplatformnost, dobrá dokumentace a moderní design výsledných aplikací. Pro Qt existují bindingy pro jiné jazyky, takže je možné se vyhnout programování v C++ a využít přednosti jiných jazyků. Dá se použít například v kombinaci s Pythonem pomocí bindingu PyQt. Pro Javu existuje binding Qt Jambi, který měl původně komerční licenci, ale od roku 2010 je spravován open-source komunitou. Podle [4] existují bindingy na množství jazyků, v některých případech však v raných fázích vývoje nebo jsou neaktuální. Samotný Qt je k dispozici zdarma pro nekomerční účely. Maximální uživatelské přívětivosti slouží nástroj Qt Designer, ve kterém mohou být tvořeny GUI velmi intuitivním WYSIWYG způsobem. Obojí Qt i Qt Designer jsou kvalitně zdokumentovány. Qt je podporován nejčastěji zastoupenými platformami (UNIX, Windows, macOS).

1.1.2 WxWidgets

WxWidgets je podobně jako Qt rozsáhlý projekt s širokou funkcionalitou, ale narozdíl od Qt je to free-ware. Podle [5] wxWidgets nemá oficiální technickou podporu, problémy je možné řešit pouze emailem nebo na fóru. Je dostupný zdarma i pro

komerční použití z oficiální stránky wxWidgets, kde se rovněž nachází bohatá dokumentace. Použití je možné na Windows, Linux, macOS a dalších platformách. Je o něco novějším frameworkem než Qt a podle [6] se snaží být i lepší. Pro wxWidgets je typická snaha o naprostou přirozenost na jakékoli platformě, kterou podporuje. V souladu s [6] správci garantují doživotní podporu i pro starší platformy. Umožňuje také kompilovat aplikace napsané pro jiný systém, než na kterém kompilace probíhá. Podporuje vícejazyčné aplikace – umožňuje snadný překlad. WxWidgets se používá spolu s běžným IDE, častá kombinace je s Visual Studiem či CodeBlocks. WxWidgets je vhodný spíše pro vývoj desktop aplikací, podpora ovládání hlasem nebo dotekem chybí. Pro tento framework je dostupný GUI designer wxGlade.

1.1.3 GTK

GTK je framework napsaný v jazyce C (což neplatí pro Qt a wxWidgets, které jsou v C++). Je produktem komunity The GNOME Project, která významně přispívá k rozvoji Unixových systémů a podle [6] byl původně vytvořen pro jejich potřeby. Nejlépe tedy funguje na systémech Linux. Je multiplatformní a volně šiřitelný. Oficiální dokumentace je spíše stručná. Aktuální verze plně podporuje škálu používaných jazyků, mnoho jich však podporuje pouze částečně nebo vůbec. I když se GTK dá použít na různých platformách, ne vždy vypadají aplikace vytvořené na jiné platformě než Linux přirozeně. Stejně tak kompilace na jiných systémech bývá problémová. Pro GTK je dostupný designer Glade.

1.1.4 Tk

Framework Tk byl původně zamýšlen jako rozšíření jazyka Tcl. Stejně jako GTK je napsán v jazyce C. Je open-source a multiplatformní. Tk je rozšiřitelný framework, rošíření na něj jsou volně dostupná. Podle [7] byl populární především v devadesátých letech, protože v porovnání s dobovými alternativami (např. Motif) stálo programování v něm mnohem méně úsilí. První verze vyšla roku 1991, framework je však stále aktuální a používaný především v kombinaci s Pythonem, ve kterém je zabudovaný binding Tkinter. Dokumentace na oficiálních stránkách je přehledná a obsáhlá, rozdělená na návod na Tcl a Tk.

1.2 Java

Jazyk Java byl navržen s důrazem na spolehlivost a pro svoji bezpečnost a podporu je častou volbou firem jako jejich interní programovací jazyk [8]. Vznikl v roce 1995. Je objektově orientovaný, ale s jednodušší syntaxí než C++. Je to interpretovaný

jazyk - kód napsaný v Javě je nejprve přeložen do byte kódu a poté interpretován virtuálním strojem Javy (JVM). Programy v Javě je možné rozběhnout na jakékoli platformě, na které se nachází vhodný JVM. Správa paměti je prováděna automaticky „garbage collectorem“ starajícím se o uvolňování paměti, kterou již program nevyužívá. To má za následek usnadnění programování. K napsání a rozběhu programu v Javě je zapotřebí Java Runtime Environment, které obsahuje JVM. Javu je možné využívat zdarma pro osobní potřebu, některé její verze jsou však pro komerční užití zpoplatněné.

1.2.1 Swing

Swing je standardní knihovnou pro tvorbu GUI v Javě. Je proto dobře zdokumentovaný a na internetu je dostupná řada tutoriálů. V rámci IDE Eclipse i NetBeans je dostupný designer, který však často generuje ne moc dobře čitelný kód [6]. Swing je dostupný od roku 1997 a podporované ovládací prvky poskytované touto knihovnou jsou tomu odpovídající, byť ze strany Oracle (stojí za vývojem Javy) jsou patrné snahy o přizpůsobení Swingu k použití pro mobilní aplikace. Stejně jako pro Javu tak pro Swing platí, že programy napsané v něm je možné spouštět na různých platformách bez významných zásahů do kódu a při zachování téměř identického vzhledu. Vývoj aplikace psané v jiném jazyce než Java s použitím Swingu je možné, pokud má tento jazyk v Javě implementaci. Mezi takovéto jazyky patří například Python, jehož implementace se nazývá Jython.

1.3 Python

Python jakožto uživatelsky přívětivý programovací jazyk je často doporučován jako první jazyk pro začínající programátory. Je tomu tak například díky automatickému rozpoznávání datových typů nebo automatické alokaci paměti, o kterou se Python postará bez zvláštního úsilí programátora. Tyto vlastnosti jsou typické pro interpretované jazyky, které jsou překládány pokaždé po spuštění programu, což ovšem nevyhnutelně vede k významnému zpomalení. Další výhodou je pravidelnost se kterou vycházejí nové verze Pythonu (k vydání nové verze dochází přibližně každý rok [9]). Disponuje bohatou standardní knihovnou zahrnující například jednoduchou kalkulačku nebo Tkinter. Pro vytvoření grafického uživatelského prostředí k programu v tomto jazyce je možné si vybrat buď nativní Kivy anebo Python bindingy na jiné frameworky.

1.3.1 Kivy

Kivy je poměrně nový a aktuální framework, jeho první verze byla k dispozici od roku 2011. Je to komunitní projekt udržovaný profesionálními softwarovými developery, ale je možné jej, podle [10], používat zadarmo. Podporuje technologii současného dotyku vice bodů na povrchu obrazovky (multi-touch) a je vhodný pro vývoj mobilních aplikací. Lze jej použít i pro desktop aplikace na Windows, macOS i Linux. Na oficiálních stránkách Kivy je dostupný rozsáhlý průvodce pro začínající vývojaře včetně tutoriálů s příklady programů, které se v Kivy dají vytvořit. Designovat uživatelské prostředí lze buď v Pythonu anebo pokud je aplikace složitější, doporučuje se použít KV language. Použití Kivy ve spojení s jinými jazyky není běžné a jejich bindingy nejsou dostupné. Podle [6] se Kivy hodí spíše pouze pro návrh samotného GUI a neimplementuje funkce, které je možné použít z jiného modulu. Pro Kivy není dostupný designer.

2 Tvorba GUI v Pythonu

Python se pro účel programování praktické části této práce jeví jako vhodná volba pro dostupnost množství užitečných nástrojů včetně designeru a pro poněkud nižší náročnost jeho použití. Hlavním důvodem však je jeho použití při napsání programu, ke kterému má být v rámci této práce vytvořeno grafické prostředí. V následující kapitole jsou popsány některé jeho vlastnosti. Dále je popsán způsob programování GUI v Pythonu pomocí zvolených nástrojů.

2.1 Vlastnosti Pythonu

Vlastnosti tohoto jazyka plynou z velké části z účelu, za jakým byl vytvořen. Python je podle [11] původně nástupce jazyka ABC, který cílil na programátory-neprofesionály. Samotný ABC byl určen k výuce programování. Zvláštností jazyka, která významně přispívá k dobré čitelnosti kódu, je nutnost odsazování skupin příkazů patřících k sobě (tělo cyklu, funkce) na stejnou úroveň. Dalším prostředkem zpřehlednění kódu je minimalizace používání interpunkčních znamének, rovněž u Pythonu odpadá nutnost učit se nové významy speciálních symbolů. Důraz na čitelnost kódu vyvěrá z úmyslu jeho recyklace. Z důvodu přívětivosti k uživateli je tedy oblíbený i mezi profesionály, jejichž hlavní náplní práce není programování [12]. Python je kvůli nutnosti interpretace kódu při každém spuštění programu poměrně pomalý. Šetření času nastává spíše při samotném psaní kódu.

2.1.1 Skriptování

Typicky se Python používá pro psaní skriptů umožňující spolupráci jednotlivých komponent rozsáhlého softwaru. Podle [12] lze Python kódem snadno rozšířit kód napsaný v Javě či C/C++. Ačkoli Python neumí volat C funkce přímo, existují nástroje (např. SWIG) umožňující blízkou spolupráci kódů napsaných v těchto jazycích. Tato vlastnost je výhodná zejména při psaní programů vyžadujících hluboké znalosti jak v oblasti uplatnění programu, tak samotného programování. Například je-li vyvíjena aplikace pro potřeby výzkumu v nějaké oblasti jiné než programování, C/C++ programátoři mohou napsat potřebné efektivní numerické algoritmy, zatímco vědci v daném oboru si mohou sami napsat program v Pythonu, který tyto algoritmy využívá, přičemž programátoři nemusejí rozumět vědě za tímto programem a vědci nemusejí ovládat nízkoúrovňové programování.

2.1.2 Režimy

Interaktivní režim Pythonu je dostupný od instalace Python interpretu na zařízení. Probíhá v prostředí příkazové řádky. Po zavolání příkazu „python“ se vypíše verze Pythonu, seznam generických příkazů typu „help“, „license“,... a tři znaky „větší než“ v řadě (>>>, platí pro Windows), naznačující že se nacházíme v interaktivním režimu Pythonu, který nyní očekává příkazy určené pro jazyk Python. Příkazy určené příkazové řádce zde nefungují. Po napsání matematického výrazu Python vrátí jeho hodnotu (funguje jako kalkulačka). Řetězec znaků je nutné psát do závorek, jinak mu interpret nerozumí. Interaktivní režim je ukončen zavoláním „quit()“ a uživatel je poté zpět v prostředí příkazové řádky.

V programovacím režimu je kód zapisován jako text a ukládán s příponou „.py“. Program je spuštěn zavoláním interpretu spolu s jeho názvem napsáním příkazu do příkazové řádky. Pokud interpret zaznamená chybu ve spuštěném programu, vypíše v okně příkazové řádky chybovou hlášku spolu s číslem řádku, kde chybu lokalizoval. Druhou možností je použít IDE, které má v sobě funkce potřebné k psaní a spouštění kódu již zahrnuty.

2.1.3 PyPI

Vzhledem k tomu, že má Python širokou podporu v mnoha oborech, může být jeho funkcionalita rozšířena o nejrůznější moduly. Tyto projekty jsou zpravidla spravovány open-source komunitou. Moduly jsou přidávány zavoláním příkazu „pip“, což je prostředek k hospodaření s balíčky v Pythonu. Repozitář rozšiřujících modulů a software se nazývá Python Package Index, zkráceně PyPI. V něm je obsaženo množství balíčků zahrnujících celou škálu oborů. Například pro vědecké výpočty je vhodný modul `NumPy`, umožňující třeba vytváření n-dimenzionálních polí nebo Fourierovu transformaci. Pomocí modulu `pywin32` je možné ovládat aplikace ve Windows, což lze využít ve skriptech pro automatizaci úloh. Kromě těchto existuje široká škála dalších modulů vytvořených pro nejrůznější účely.

2.1.4 OOP paradigma

Python je objektově orientovaný, tzn. využívající objekty a třídy. Kód napsaný v objektově orientovaném jazyku by měl být znovupoužitelný v jiných projektech. Níže jsou vysvětleny pojmy související s objektovou orientací:

Objekt – reprezentace určité věci (např. nějaké osoby, předmětu), která má specifické vlastnosti, tzv. atributy. Tyto vlastnosti jsou proměnné závislé na konkrétním objektu. Každý objekt má také své funkce (nazývané metody), které jsou na data popisující objekt aplikovány.

Třída – šablona, podle které mohou být dodefinováním proměnných vytvořeny objekty. Zahrnuje pouze atributy a metody obecně, dosazením proměnných vznikne objekt. Instance třídy je jedno její použití (jeden objekt).

2.2 Proces programování GUI

Grafické uživatelské prostředí je možné vytvořit dvěma způsoby – buď psaním kódu s použitím knihoven pro tvorbu GUI, nebo pomocí designeru.

Při psaní GUI je možné použít stejné prostředí jako pro psaní programů bez grafického rozhraní, pouze je nutné si do něj obstarat potřebné knihovny. Pokud nejsou spolu s interpretem nebo kompilátorem jazyka dostupné ve výchozím nastavení, je nutné je doinstalovat. Na začátku kódu je příslušným příkazem knihovna vložena, od té chvíle jsou přístupné její metody a tzv. „widgety“, což jsou vizuální interaktivní prvky. Je tedy možné vytvořit např. klikací tlačítko zavoláním patřičného konstruktoru, který widget instancuje, umístit jej do layoutu aplikace zvolenou metodou (do mřížky, na přesně zvolené místo,...) a dát mu funkcionalitu přiřazením funkce, napsané v programu. Umisťování elementů do layoutu lze realizovat v absolutním nebo relativním systému [13]. V absolutním systému je elementu přiřazeno na pixel přesné místo, kam patří. Nevýhodou je, že se grafické rozhraní takto napsané špatně škáluje. V relativním systému je elementu vzdálenost od krajů obrazovky zadávána jako poměrná část vzhledem k celku. Také je možné element napevno připoutat do rohu, k nějakému okraji obrazovky či na střed.

Práce s designerem je v porovnání s „doslovným“ programováním GUI intuitivnější. V počáteční fázi vývoje aplikace v designeru není nutné se učit nové příkazy a názvy týkající se tvorby GUI, je v něm tvořena čistě jen její vizuální stránka a to metodou „drag and drop“. Požadovaná okna, tlačítka a ikony jsou jednoduše vybrány z lišty a umisťovány do layoutu. Tvorba grafického rozhraní v designeru je vlastně realizována v prostředí, které samo disponuje grafickým rozhraním. Po nastavení layoutu požadovaného vzhledu je možné vygenerovat jeho kód a pak už pracovat s ním pro doprogramování specifických funkcionalit aplikace.

2.3 PyQt

Nejrozšířenějšími nástroji pro tvorbu GUI v Pythonu jsou PyQt, Tkinter, wxPython, Kivy a PySide [14]. Který z nich zvolit je z velké části volbou osobní preference. PyQt byl zvolen z důvodu dostupnosti designeru a množství materiálů a návodů vyskytujících se na internetu. Knihovna widgetů v Qt je skutečně obsáhlá a svým vzhledem přizpůsobitelná vybrané platformě.

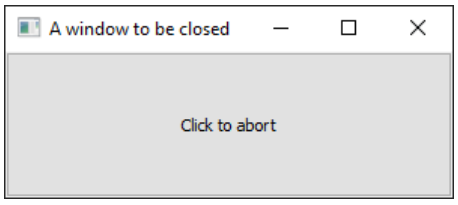
V PyQt jsou dostupné dva typy objektů – vizuální a ne-vizuální. Nevizuální objekty jsou např. uzly (třída `QNode`) ze kterých je možné skládat stromové struktury. Je pro ně typické, že nemají žádnou vizuální reprezentaci, ale mají své vlastnosti a metody. Vizuální objekty, takzvané widgety, jsou pod modulem `QtWidgets`. Mohou přijímat události z ovládacích prvků (myš, klávesnice) a jsou reprezentovány vizuálně na obrazovce. Hlavní okno je widget, který není vsazený do jemu nadřazeného widgetu, přitom je samo nadřazeným (mother) widgetem do něj vsazených widgetů. Může mít svůj rám a název. Jak nastavit jeho parametry spolu s příkladem jak do něj vložit widget je ukázáno v kódu níže.

2.3.1 Příklad jednoduché aplikace

Nejjednodušší aplikaci s grafickým rozhraním lze napsat na několik řádků. Jako příklad je zde uvedena aplikace s upraveným názvem okna a obsahující pouze tlačítko, které po kliknutí toto okno zavře. Kód vypadá následovně:

```
1  from PyQt5 import QtWidgets
2
3  app = QtWidgets.QApplication([])
4
5  button = QtWidgets.QPushButton('Click to abort')
6  button.setWindowTitle('A window to be closed')
7  button.clicked.connect(app.quit)
8  button.show()
9
10 app.exec()
```

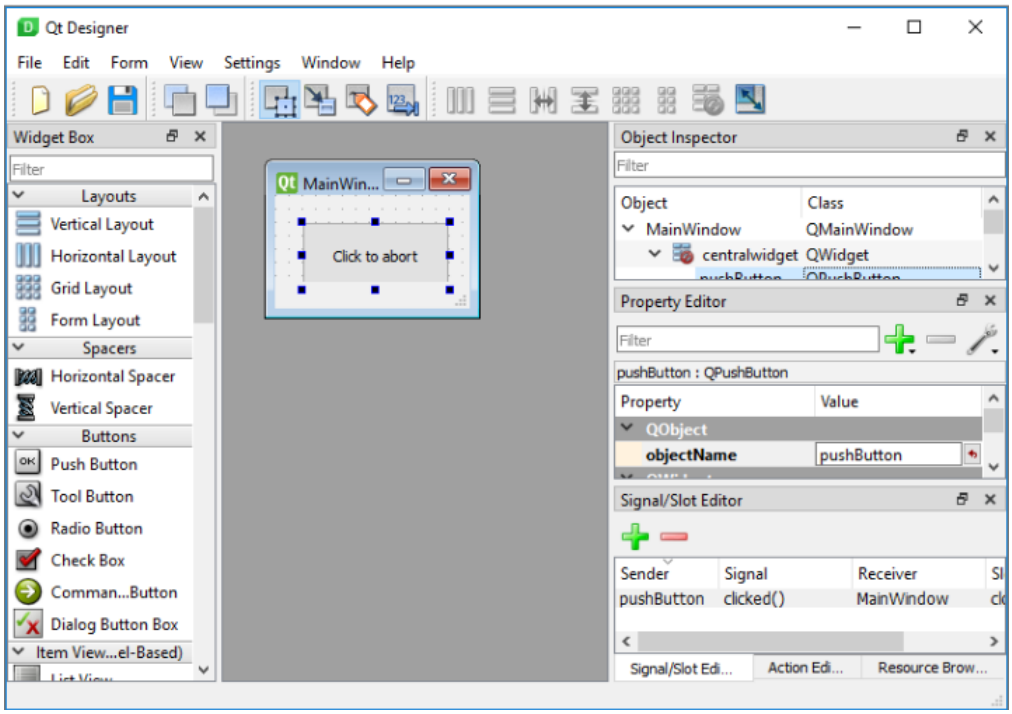
Jak aplikace vypadá lze vidět na obrázku 2.1. Na prvním řádku v kódu je importován modul `QtWidgets`, který obsahuje mimo zde použitý widget „tlačítko“ ještě škálu dalších elementů patřících do grafického rozhraní. Poté je vytvořen objekt `app`, který je instancí třídy `QApplication`, tedy aplikace. Dále je psán kód popisující jak bude aplikace vypadat. Jako první je v něm vytvořeno klikací tlačítko (`QPushButton`), tečka před zmínkou o něm značí, že je převzatý z nějakého modulu, název před tečkou značí z jakého modulu. V závorkách je doplněn řetězec, který bude po interpretaci kódu a vytvoření grafického okna vyobrazen na tomto tlačítku. Dále je nastaven název okna. Funkcionalita klikacího tlačítka na zavření aplikace je nastavena slovem `connect`, které propojuje signály s událostmi. Na posledním řádku vztahujícím se k tlačítku je příkaz, který tlačítko udělá viditelným po spuštění aplikace. Bez příkazu `show` by se sice ostatní příkazy vykonaly, ale neprojevily by se ve vytvořeném GUI. Nakonec je spuštěna smyčka událostí, díky které GUI aplikace běží a čeká na vstupy od uživatele.



Obr. 2.1: Aplikace vytvořená pomocí PyQt5

2.3.2 Práce s designerem

Při práci s designerem využíváme spustitelnou aplikaci Qt Designer spolu s prostředím příkazové řádky a textovým editorem. Po spuštění Qt Designeru se uživatel dostane do prostředí znázorněného na obrázku 2.2:



Obr. 2.2: Prostředí Qt Designeru

Je zde vidět, že jednotlivé widgety jsou vybírány z boční lišty nalevo a umisťovány do šedého prostoru. V liště napravo je možné upravovat funkcionalitu widgetů. Dole jsou položky `Sender`, `Signal`, `Receiver` a `Slot`. S jejich pomocí je možné nastavit reakci widgetu na událost. V tomto příkladě je jako sender zvolen widget „pushButton“, který byl umístěn do hlavního okna. Jako signál byla zvolena událost „kliknutí“. Přijímač je zde hlavní okno, které má být v souladu s volbou v položce `slot` zavřeno. Vpravo nahoře je vidět hierarchie jednotlivých widgetů. Po uložení

aplikace je možné vygenerovat její kód pomocí příkazu v příkazové řádce `pyuic5 -x abort.ui -o abort.py`. Pyuic5 je pomůcka Pythonu sloužící ke spravování „.ui“ souborů, které vytváří Qt Designer. „Abort.ui“ je soubor vytvořený Qt Designerem a „abort.py“ je soubor, do kterého má být vygenerován kód, který vypádá následovně:

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(146, 85)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(20, 20, 111, 51))
        self.pushButton.setObjectName("pushButton")
        MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        self.pushButton.clicked.connect(MainWindow.close)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "
    MainWindow"))
        self.pushButton.setText(_translate("MainWindow", "Click to
    abort"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Generovaný kód je v porovnání s ručně psaným kódem stejné funkcionality dlouhý a nepřehledný, programování GUI aplikace je nicméně v Qt Designeru rychlejší a v místě potřeby je možné zasáhnout do jejího kódu ručně. Ideální je však mít

dopředu dobře promyšlen layout aplikace, aby odpadla nutnost pozdějších zásahů do kódu GUI.

3 Prostředky optimalizace elektromotorů

Cílem této kapitoly je objasnit požadavky na elektrické stroje a nastínit možnosti jak je splnit pomocí dostupných prostředků. V oblasti návrhu elektrických strojů hraje důležitou roli jejich efektivita, která je posuzována podle jejich účinnosti. Účinnost udává, jaký poměr energie přivedené stroji se při její přeměně na jiný typ energie přemění v užitečnou energii. Elektrické točivé stroje lze dělit podle jejich režimu na motory a generátory. Pro motory tedy platí, že účinnost je poměr elektrické energie přeměněné na mechanickou. Pro generátory platí opačný poměr - mechanická ku elektrické energii. Snahou je, aby se účinnost co možná nejvíce blížila jedné. Účinnost stroje je možné při návrhu ovlivnit několika proměnnými - konstrukcí stroje a použitým materiálem plechů, vinutí a izolace. Převážně na těchto skutečnostech dále závisí parametry stroje, kterými je jeho účinnost definována. Jejich optimalizací se snažíme o splnění požadavků na daný stroj.

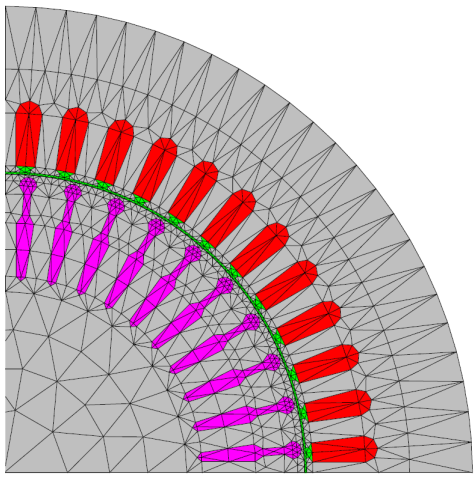
3.1 Úvod do točivých elektrických strojů

Točivé elektrické stroje lze dělit na asynchronní, synchronní a stejnosměrné. Byť fungují na různých principech, je přitom využíváno dvou stejných základních částí strojů - magnetického obvodu a vinutí [15]. Magnetický obvod udává dráhu magnetického toku. Je tvořen plechy z feromagnetického materiálu (litiny, uhlíkaté oceli s příměsemi,...) a vzduchovou mezerou mezi statorem a rotorem. Plechy jsou od sebe pro zamezení ztrát vířivými proudy vzájemně izolovány. Vinutím jsou myšleny vodiče obvykle vyrobené z mědi nebo hliníku díky jejich vysoké vodivosti. Je izolováno od magnetického obvodu, ve kterém je uloženo. K rotačnímu pohybu v elektrických strojích dochází díky magnetické indukci, která průchodem proudu vinutím vytvoří magnetický tok magnetickým obvodem.

3.2 Metoda konečných prvků

Tato numerická metoda vznikla za účelem řešení úloh pružnosti a dynamiky složitějších soustav. Spočívá v rozdělení soustavy na konečný počet takových podoblastí, jejichž chování lze již popsat snadno [16]. Programy provádějící výpočty pomocí metody konečných prvků k tomu využívají tzv. „mesh“, což je síť tvořená hranicemi těchto elementů, které si lze povšimnout na příkladu indukčního motoru na obrázku 3.1. Je zde zřejmé, že je využito určité geometrické symetrie, což snižuje výpočtovou náročnost daného problému.

Při popisu chování elementů je obvykle využit maticový počet. Dále jsou popsané prvky složeny zpět do celku a vzniknou tak matice popisující chování této



Obr. 3.1: Model indukčního motoru se sítí vytvořenou hranicemi elementů před aplikací metody konečných prvků. [17]

soustavy. Metoda zohledňuje vztahy mezi napětími, přetvořeními a posuvy. Řešení má podobu hledaných parametrů v uzlech sítě tvořené hranicemi mezi prvky a platí pouze pro konkrétní řešenou situaci. Při jakékoli změně parametrů je třeba proces řešení opakovat.

3.3 Ansys v návrhu elektromotorů

Jedním z prostředků, jak navrhnout elektrický stroj je použití softwaru Ansys. Jde o program umožňující vytvoření modelu a jeho následnou analýzu. Ansys je využíván v řadě různých odvětví díky pokrytí množství fyzikálních výpočtů spojených s vývojem elektroniky, stavebnictvím či strojírenstvím. Jeho distribuce se specializují na optiku, tekutiny, polovodiče aj. Řada programů od Ansys je dostupná zdarma, ale jsou většinou jednoúčelové a ne zdaleka tak komplexní jako komerční verze programu. Jelikož komerční verze Ansysu jsou rozsáhlé programy se širokou funkcionalitou, proces získání licence je vcelku složitý a etika jejího užívání přísná.

Vlajkovou virtuální laboratoří Ansysu je Ansys Electronics Desktop a jejím primárním účelem jsou simulace nízkofrekvenčních magnetických polí a pohybů těles v nich. Je v něm dostupná časová i frekvenční analýza polí a pracuje s metodou konečných prvků. Je využíván při návrhu senzorů a elektrických strojů včetně motorů a transformátorů.

3.4 Optimalizační algoritmy

Algoritmus obecně se dá popsat jako soubor kroků vedoucí od výchozího stavu k požadovanému výsledku. Je-li řeč o optimalizaci, potom tyto algoritmy cílí na vylepšení vlastností daného subjektu. Problémem některých optimalizačních algoritmů je jejich inklinace pouze k lokálnímu řešení. V zájmu řešitele je obvykle nalezení globálního řešení, což je možné snáze za použití heuristik. V oblasti optimalizace můžeme rozlišovat algoritmy deterministické a stochastické [18]. Deterministický algoritmus se chová předvídatelně - ve stejném počtu kroků dojde ke stejnému řešení. Stochastický algoritmus využívá heuristického prohledávání daného prostoru. Heuristické je takové, které zahrnuje náhodu a pro přesnější výsledky vyžaduje zkušenost. Tyto jsou obvyklé v přírodě, kde dochází k optimalizaci samovolně, proto je řada optimalizačních algoritmů inspirována jejími vypozorovanými tendencemi. Mezi ně patří genetický algoritmus a samo-organizující migrující algoritmus (SOMA).

3.4.1 Genetický algoritmus

Evoluční algoritmy, mezi které je řazen genetický algoritmus, byly vypozorovány na chování populace určitého druhu. Reprodukce jejích členů je řízena heuristicky a vede ke zlepšení celkových vlastností a životaschopnosti populace. Tyto algoritmy jsou podle [18] užívány v úlohách o mnoha lokálních optimech, protože mají vyšší pravděpodobnost nalezení globálního optima.

Genetický algoritmus je jedním z evolučních algoritmů. Generace je jedním cyklem genetického algoritmu. Na začátku cyklu je každý jedinec ohodnocen a je zváženo, zda by mohl být vybrán jako rodič. Po výběru rodičů je aplikováno křížení, ze kterého vzniknou jedinci nové generace. Na potomky je aplikována operace mutace, která zavádí do genetického vývoje populace variabilitu.

3.4.2 Samo-organizující migrující algoritmus

SOMA je inspirován přesunem smečky za potravou. Jeden jedinec ve smečce je vždy nejblíže tomuto optimu a ostatní se k němu přesouvají. Nejprve je vygenerována a vyhodnocena výchozí populace a vůdce, což je jedinec s nejnižší hodnotou účelové funkce - je „nejblíž potravě“. Účelová funkce se zde používá k vyhodnocování jedinců v populaci. Čím je její hodnota pro daného jedince nižší, tím je jedinec vhodnější pro to, aby byl následován - je blíže ideálnímu řešení. Algoritmus je tedy založen na migraci inteligentních jedinců prohledávajících prostor řešení [18]. Samo-organizace v názvu algoritmu plyne z povahy pohybu populace - může se při procesu hledání řešení rozpadat na menší skupiny až na jednotlivce a opět se spojovat v početnější skupiny. Výhodou tohoto algoritmu je jeho rychlá konvergence k minimální

hodnotě účelové funkce. Není však imunní vůči lokálním minimům. Podle [19] je proto vhodné proces opakovat a ověřit stejnost jeho výsledků, popřípadě algoritmus iniciovat s dostatečným počtem jedinců (pro aplikaci v oblasti točivých elektrických strojů možno uvažovat okolo padesáti).

Pro implementaci tohoto algoritmu jsou podle [18] využity parametry PathLength, Steps, PRT, PopSize a Migration. PathLength souvisí se vzdáleností, do jaké jedinec migruje, která je vztažená na vůdce. Např. pokud je tento parametr roven 1, znamená to že se migrující jedinec zastaví právě na místě vůdce. Počet ohodnocení jedince v průběhu jeho migrace je dán také parametrem Steps. Jeho hodnota udává krok migrujícího jedince. Je vhodné ho volit s ohledem na parametr PathLength tak, aby podíl PathLength a Steps nebylo celé číslo - docházelo by jinak k dvojitému ohodnocení stejného řešení. Z hodnoty PRT je vytvořen tzv. perturbační vektor, který je polem jedniček a nul. Vlastní hodnota PRT udává pravděpodobnost nabytí hodnoty 1 pro každou složku PRT vektoru. Náleží-li migrujícímu jedinci složka hodnoty 1, znamená to že nebude migrovat v přímém směru vůdce. Je to tedy jakýsi prvek náhody. Parametr PopSize udává počet jedinců výchozí populace, tito jedinci jsou náhodně „rozhozeni“ po stavovém prostoru. Hodnota parametru Migration udává počet cyklů algoritmu - v každé migraci je zvolen podle hodnoty účelové funkce jeden vůdce, ke kterému zbytek populace migruje.

4 Synchronní motor s permanentními magnety

Podle [19] lze synchronní stroje dělit do tří kategorií podle buzení. V minulosti převládaly motory s budicím vinutím uloženým v rotoru, které bylo napájeno stejnosměrným proudem. Dalším typem jsou reluktanční motory fungující na principu změny magnetického odporu. Tyto v posledním desetiletí přitahují značnou pozornost, stejně jako synchronní stroje s permanentními magnety. Mezi jejich výhody patří vysoká účinnost a vysoká odolnost proti přetížení v porovnání s jinými typy synchronních strojů. O jejich rozvoj se v nedávné době zasloužil výzkum v oblasti magnetů ze vzácných zemin. Synchronní stroje s permanentními magnety (PMSM) rovněž využívají ovládacích algoritmů (regulátorů), které umožňují jejich provoz při nejvyšší možné účinnosti.

4.1 Princip PMSM

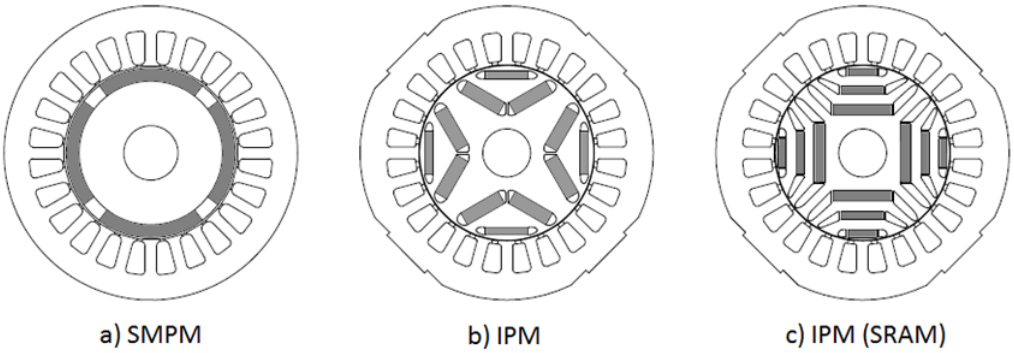
Funkce třífázových elektrických strojů je založena na vzájemném vlivu točivého magnetického pole statoru a magnetického pole v rotoru. Motor, ve kterém se rotor otáčí stejnou rychlostí jako točivé pole statoru se nazývá synchronní. V případě PMSM je rotorové magnetické pole konstantní díky přítomným permanentním magnetům. Permanentní magnety ze vzácných zemin se v oblasti synchronních motorů vyžívají přibližně 50 let [20]. Přes svoji vysokou cenu byly používány pro lineární charakter své demagnetizační křivky. Směs samaria a kobaltu vystřídaly neodymové magnety, které mají pro použití v PMSM dobré magnetické vlastnosti (jsou odolné vůči demagnetizaci). Podle [20] není u tohoto uspořádání nutné snímání polohy rotoru. Vyšších výkonů a nižší hmotnosti při stejné velikosti motoru je dosaženo absencí budicího vinutí, ve kterém vznikají ztráty.

4.2 Konstrukce PMSM

Stator je tvořen elektrotechnickými plechy tvořícími magnetický obvod motoru, ve kterém je v drážkách uloženo třífázové vinutí. Statorové plechy jsou od sebe vzájemně izolovány, stejně tak vinutí od statoru. Na statorové plechy je použit takový materiál, aby bylo co nejvíce zamezeno vzniku vířivých proudů. Na vinutí se pro své dobré vodivé vlastnosti a tedy omezení ztrát používá měď.

Dle [19] mohou být magnety zapuštěny v rotoru (Interior Permanent Magnet, zkráceně IPM) nebo umístěny na povrchu rotoru (Surface Permanent Magnet, zkráceně SPM) - viz. obrázek 4.1. IPM má v porovnání s SPM významné výhody. Za-

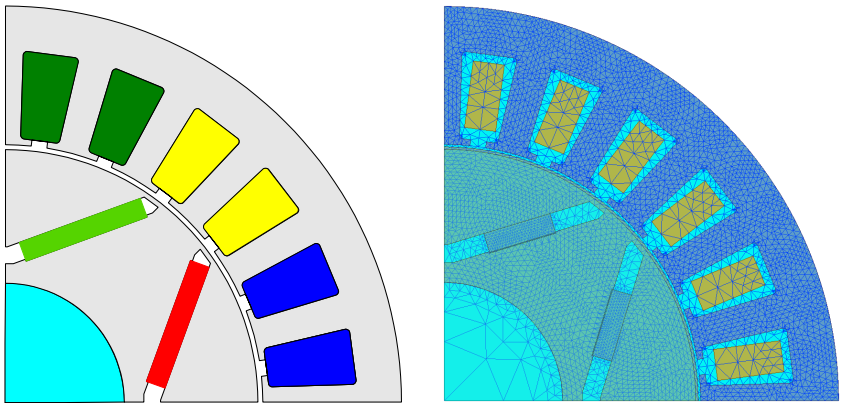
puštění magnetů do rotoru je snazší na provedení, není nutné použití lepicích medií. Také jsou odolnější vůči demagnetizaci při zkratu mezi fázovými vinutími, popř. mezi vinutím a uložením motoru.



Obr. 4.1: Konstrukce SMPM, IPM a SRAM [21]

4.3 Motor pro optimalizaci

Pro praktickou část práce byl vybrán motor typu IPM s magnety zapouzdřenými v rotoru umístěnými do V - viz. obrázek 4.2:



(a) (b)

Obr. 4.2: a) Výchozí geometrie, b) model k konečně-prvkovou sítí [19]

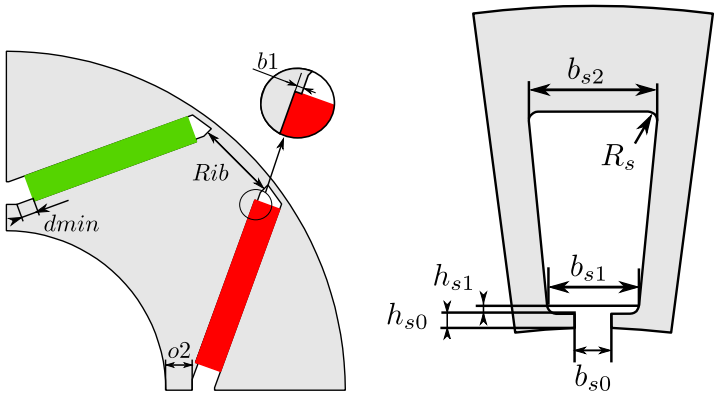
Na obrázku 4.2 b) je dobře viditelná síť mezi elementy vytvořená v Ansysu před aplikací metody konečných prvků. Motor má 4 póly a 24 statorových drážek, ve kterých je umístěno vinutí. Rotor je vyroben z plechů.

Konkrétní parametry motoru jsou uvedeny v tabulce 4.1:

Parametr	Značka	Hodnota
Jmenovitý moment	M_{N}	28 Nm
Jmenovité sdružené napětí	U_{S}	400 V
Jmenovitý účiník	$\cos\varphi$	0.87
Jmenovité otáčky	n	1500 min^{-1}
Vnější průměr statoru	D_{out}	200 mm
Délka	L_{stk}	120 mm
Počet drážek/pólů	$Q_{\mathrm{s}}/2\mathrm{p}$	24/4

Tab. 4.1: Jmenovité parametry motoru [19]

Parametry, které jsou předmětem optimalizace, jsou uvedeny na obrázku 4.3 a v tabulce 4.2. Jedná se o celkem 10 rozměrů, 4 parametry pro rotor a 6 pro stator.



(a) (b)

Obr. 4.3: a) Rotor, b) stator [19]

Parametr [mm]	Rib	dmin	b1	o2	b_{s0}	b_{s1}	b_{s2}	h_{s0}	h_{s1}	R_{s}
Výchozí geom.	10	20	0.2	5	4	10	14	1.5	1	1

Tab. 4.2: Výchozí rozměry [19]

4.4 Hlediska optimalizace

Objektiva pro optimalizaci u elektrických strojů obecně jsou volena tak, aby z nich mohla být objektivně hodnocena kvalita těchto strojů. Podle [19] jsou obvykle voleny

veličiny jako točivý moment, zvlnění točivého momentu a účinnost stroje. Za pomoci optimalizačních algoritmů je možné stroj navrhnout tak, aby měl maximální účinnost při nejvyšším průměrném momentu a nejnižším zvlnění momentu, protože je testována široká paleta geometrických kombinací. Mezi sledované parametry patří také velikost první harmonické indukovaného napětí a celkové harmonické zkreslení indukovaného napětí (THD). Největší důraz je přitom kladen na zvýšení průměrné hodnoty momentu a snížení zvlnění momentu, protože tyto jsou v úzkém vztahu s účinností. THD je sledováno kvůli přítomnosti vyšší harmonických, které způsobují dodatečné ztráty v železe a mědi. Snížení zvlnění momentu je žádoucí, protože může mít vliv na vibrace v motoru, které zkracuje životnost ložisek.

5 Praktická část tvoření GUI

Cílem této kapitoly je vystihnout proces tvoření grafického rozhraní pro program optimalizující elektromotor. Na začátku práce byl poskytnut soubor s funkcemi pro optimalizaci pomocí SOMA algoritmu a optimalizační skript, obojí psáno v Pythonu. Dále dva excelové soubory. Jeden jako vstup pro skript, druhý pro záznam průběhu a výsledků optimalizace. Pro testování byly poskytnuty dva modely motorů vymodelované v Ansysu. Úkolem bylo navrhnout takové GUI, které plně nahradí vstupní excelový soubor a zobrazí výsledky optimalizace v podobě srovnávacích grafů. Výstupní excelový soubor přitom zůstane zachován.

5.1 Návrh programu

Samotné tvorbě GUI předchází návrh. Rozhraní je navrhováno s ohledem na funkce aplikace a se snahou o snadnou orientaci v ní. Je bráno v potaz, zda je aplikace psána od samého začátku nebo je GUI doděláno až pro fungující aplikaci - postup se v těchto dvou případech mírně liší. V prvním případě je vhodné nejprve navrhnout GUI a poté dopsat funkční složky programu. V případě této práce již aplikace existuje, bylo tedy třeba se nejprve s aplikací blíže seznámit - zjistit podobu jejích vstupů a výstupů včetně datových struktur, do kterých jsou vnitřně ukládány. Je výhodou alespoň na základní úrovni porozumět principu funkce aplikace, což je v případě psaní aplikace od začátku nutností. Po získání přehledu o tom, jak ovládat aplikaci, je již možné navrhnout jak by její grafické rozhraní mohlo vypadat. Měly by v něm být zahrnuty všechny vstupy, kterými je možné ovlivnit chod programu, v podobě vhodných ovládacích prvků. Dále je nutné naprogramovat interakci mezi GUI a funkční složkou programu, například poslat vstupy zadané do GUI do patřičných funkcí nebo co se v programu stane pokud dojde k určité události (kliknutí, přejetí myší apod.). Jelikož program pracuje s motory vymodelovanými v Ansysu, bylo také třeba zajistit komunikaci mezi Pythonem a Ansysem.

5.1.1 Nástroje

K realizaci GUI byl použit Qt Designer pro Python. Pro práci s kódem GUI bylo použito IDE PyCharm. Původní úmysl napsat aplikaci pomocí textového editoru a příkazové řádky byl překonán po odzkoušení PyCharmu. Přes poněkud nepřehledně vyřešený management virtuálních prostředí se PyCharm ukázal jako významný pomocník, především díky debuggeru (nástroj na odladování programu). Zabudovaná konzole se také ukázala jako užitečná - práce s ní je podobná jako s příkazovou řádkou.

Pro chod programu je zapotřebí řada knihoven. Některé z nich jsou v Pythonu zabudovány, mezi takové patří použité knihovny `csv` pro práci se soubory formátu csv, nebo `pathlib` pro manipulaci s adresami souborů. Knihovny pro práci s daty `NumPy` a `pandas` bylo třeba pomocí package manageru přidat. `NumPy` Python obohacuje o datovou strukturu zvanou „pole“ a operace s nimi (jinak jsou v Pythonu namísto polí takzvané „seznamy“). `Pandas` slouží k manipulaci s daty a jejich analýze. Mezi grafické knihovny, které byly použity, patří samozřejmě PyQt, verze 5. Pro zobrazení grafů byla použita knihovna `PyQtGraph`. Ke komunikaci s Ansysem byl využit modul `pywin32`. Pro převedení hlavního skriptu do „.exe“ souboru byl použit modul `pyinstaller`.

5.1.2 Návrh layoutu aplikace

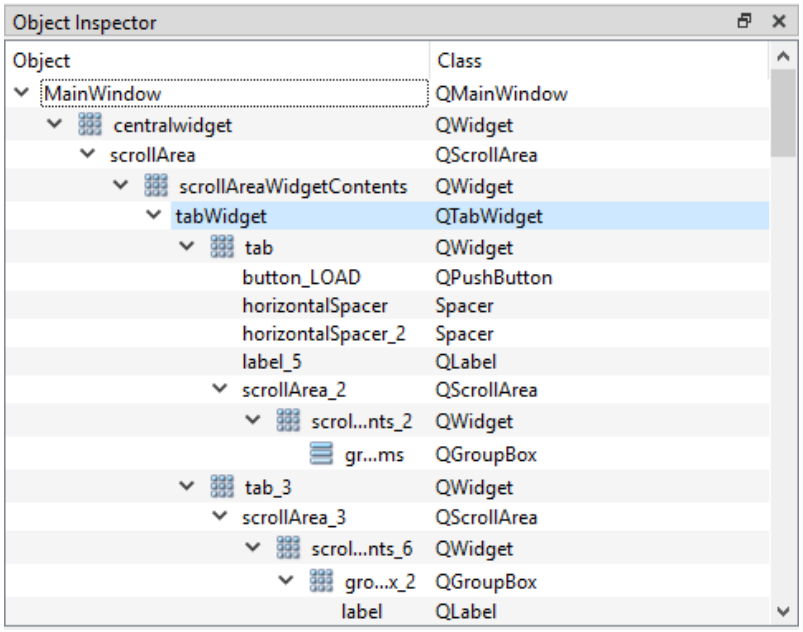
Grafické prostředí bylo navrženo jako systém tří záložek. V první záložce je pouze tlačítko a prázdný tzv. group box, což je widget sdružující skupinu jemu podřízených widgetů. Po kliknutí na tlačítko je vyvoláno dialogové okno s možností výběru souboru s koncovkou „.aedt“, která je vlastní Ansys modelům. Po vybrání modelu se naplní group box zaškrtávacími políčky, každý s názvem parametru, které byly definovány v Ansysu při vytváření modelu.

Ve druhé záložce jsou čtyři group boxy. První z nich je pro rozměry k optimalizaci a je defaultně prázdný. Naplní se až po vybrání modelu v první záložce a to pouze těmi rozměry, které jsou zaškrtnuty. Dále je u každého rozměru zobrazena jeho číselná hodnota a je možné pro něj nastavit minimální a maximální hodnotu, které může nabýt optimalizací. Další group box náleží objektivům optimalizace a parametrům SOMA algoritmu. Zde je možné nastavit cílovou hodnotu průměrného momentu, zvlnění momentu a celkového harmonického zkreslení indukovaného napětí a jejich váhy. Nastavované parametry SOMA algoritmu jsou počet členů populace, počet migrací, PathLength, Steps a PRT - prvek náhody. Ve třetím group boxu jsou widgety umožňující upravit parametry výsledných grafů. Lze nastavit čas a krok analýzy. Čas udává trvání analýzy, krokem je dáno rozlišení. V posledním group boxu se nachází tlačítko Start, které spustí optimalizaci, dále stopky měřící trvání analýzy a progress bar.

Třetí záložka je určena zobrazení výsledků optimalizace. Jsou v ní tři grafická okna. Jedno pro závislost momentu na čase, druhé pro závislost indukovaného napětí na čase. V každém z těchto grafů jsou po skončení optimalizace dva průběhy. Jeden průběh je pro původní model, který vstoupil do optimalizace. Druhý průběh je pro nejlepší model, který byl při optimalizaci nalezen. Třetí graf je závislost hodnoty účelové funkce na fázi optimalizace.

5.2 Realizace layoutu v Qt Designeru

Knihovna PyQt je napsána jako objektově orientovaná. Různé kategorie grafických prvků jsou představovány třídami a s konkrétními widgety je zacházeno jakožto s objekty. Vztahy mezi widgety na základě třídní hierarchie jsou dobře viditelné v Object Inspectoru na obrázku 5.1. Ten je v Qt Designeru připnutý v pravé liště (View -> Object Inspector).

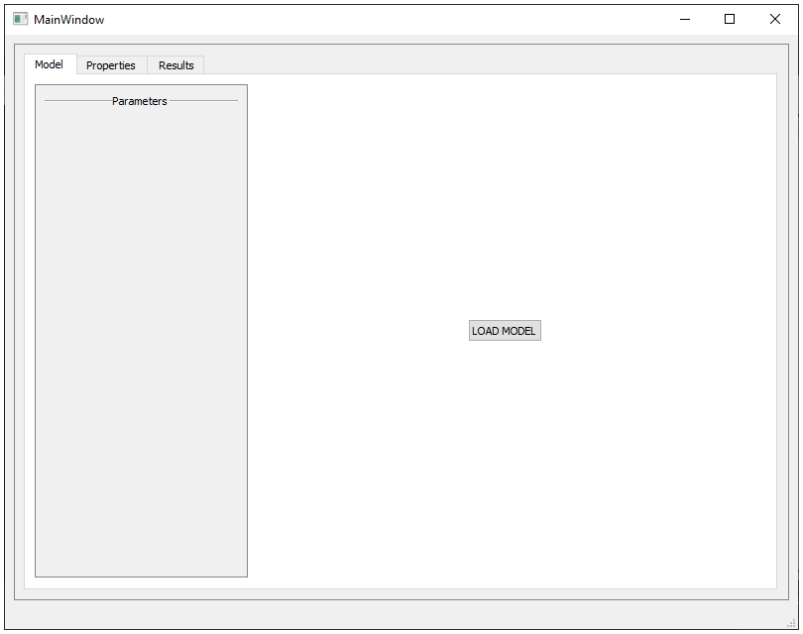


Obr. 5.1: Hierarchie grafických prvků umístěných v hlavním okně

Znalost grafických tříd může být při tvorbě layoutu GUI užitečná. Jako nejlépe schůdný způsob tvoření layoutu je postup od nejvýše postavené třídy postupně k těm nižším. Nejvýše postavenou třídou je zde `QMainWindow`, její instance MainWindow je vytvořena při zvolení File -> New -> Main Window. Spolu s ním je vytvořena instance centralWidget třídy `QWidget`. Funkcí centralWidgetu je zaplnit MainWindow prostorem pro další grafické prvky. Platí, že pokud je nějaký objekt (např. tlačítko) umístěn do nějakého grafického prostředí (např. záložka), stává se tento widget potomkem tohoto prostředí. Tato skutečnost je užitečná při manipulaci s widgety v kódu. Je vhodné jako první věc do centralWidgetu vložit objekt scrollArea - ta umožní při změně velikosti okna aplikace pod určitou hranici vytvoření posuvníku. Jak u centralWidgetu, tak i u posuvníkové oblasti by dále měl být nastaven layout (zde způsob uspořádání widgetů). Ten zajišťuje uspořádání potomků podle zvoleného pravidla. Zde byl v obou případech použit grid layout, který způsobí roztažení grafického prvku po celé ploše kterou mu jeho rodič poskytuje. V případě, že je potomků více, uspořádají se do mřížky.

Dalším pro design důležitým prvkem je tzv. spacer. Lze použít buď horizontální nebo vertikální a jeho funkcí je zachovat rozestupy mezi grafickými prvky. Spacery přitom stejně jako ostatní widgety podléhají pravidlům nastaveného layoutu a výsledný vzhled okna je těžko předvídatelný. Je zapotřebí zkoušet různé kombinace spacerů a layoutů, dokud okno neodpovídá představám tvůrce GUI. Zde je proto vhodné zdůraznit výhodu designeru - efekt vložení widgetu, který manipuluje uspořádání ostatních grafických prvků, je vidět okamžitě.

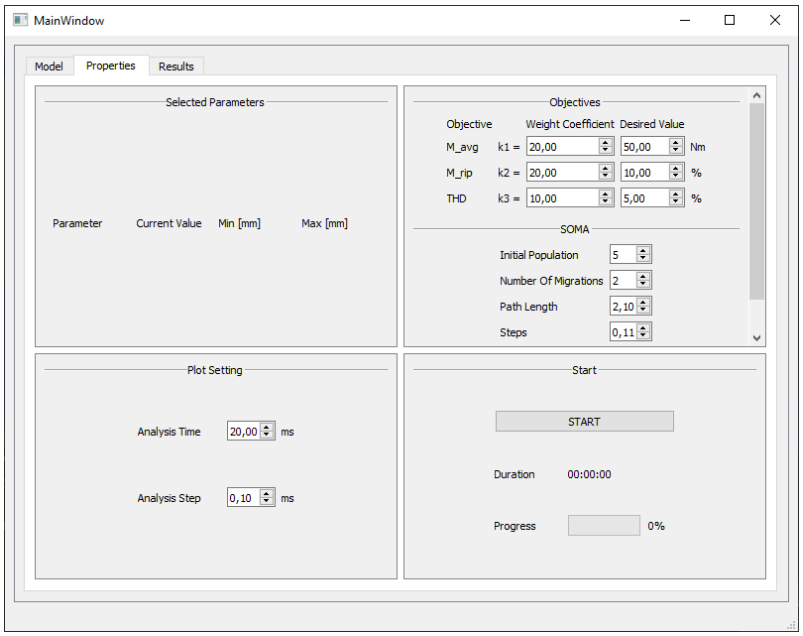
Po layoutu a posuvníkové oblasti byl do centralWidgetu vložen tabWidget, což je systém záložek. V první záložce (obrázek 5.2) byl nastaven také layout do mřížky, který vytvořil vizuálně dobrý výsledek. Záložka není příliš obsáhlá. Je v ní group box. Tento je záměrně nechán prázdný, protože jeho obsah závisí na zvoleném modelu, defaultně není zvolen žádný a čeká se na volbu uživatele. Od tlačítka, které slouží k vyvolání dotazu na model, je oddělen spacerem. Protože není známé množství parametrů, které se v group boxu může objevit, má group box svůj vlastní posuvník. V samotném group boxu je nastaven horizontální layout, protože se předpokládá vypsání parametrů po řádcích.



Obr. 5.2: Záložka Model

Pro druhou záložku (obrázek 5.3) je rovněž nejvhodnější layout v mřížce. Je tvořena čtyřmi group boxy a slouží k nastavování parametrů programu. První group box je defaultně prázdný, jsou v něm pouze nadepsané informace které očekává. Po zvolení modelu se v tomto group boxu vypíší právě ty rozměry, které mají zaškrtnutý check box v první záložce. Spolu s jejich názvy se vypíše jejich aktuální hodnota a také navrhovaná minimální a maximální hodnota, které bude moct bě-

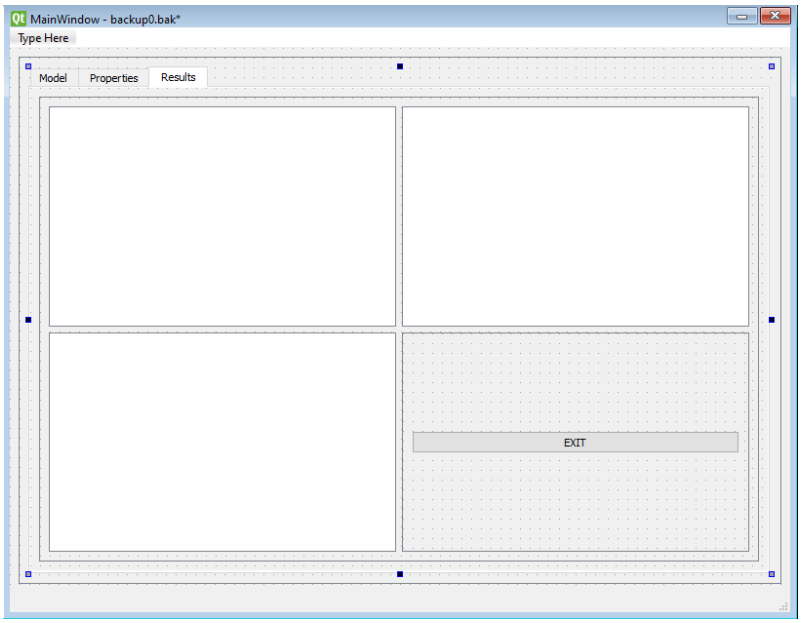
hem optimalizace nabýt. Dále se v této záložce nachází group box rozdělený na dvě části. V první části jsou nastavována objektiva optimalizace. Jsou zde zadávány kýžené hodnoty průměrného momentu, zvlnění momentu a THD spolu s jejich váhovými koeficienty, které udávají jednotlivé důležitosti těchto parametrů v optimalizaci. Ve druhé části se nastavují parametry SOMA algoritmu - počáteční populace, počet migrací, PathLength, počet kroků a PRT vektor. Ve třetím group boxu jsou nastavovány parametry pro výsledné grafy, tj. čas a krok analýzy. V posledním group boxu se nachází tlačítko Start spolu se stopkami k měření trvání optimalizace a tzv. progress barem. Veškeré widgety sloužící vstupům uživatele jsou tvořeny tzv. spin boxy nebo double spin boxy. Do spin boxů lze zadávat pouze celá čísla, do double spin boxů i desetinná čísla. Štítky s nápisy jsou tvořeny labely. Tyto nemůže uživatel měnit.



Obr. 5.3: Záložka Properties

Ve třetí záložce (obrázek 5.4) jsou zamýšleny grafy, což je také řešeno grid layoutem. Graf je možné zobrazit do widgetu graphicsView, které jsou vloženy do grid layoutu celkem tři. Je výhodné si už v designeru nadepsat záložky, štítky (labely), tlačítka a především změnit defaultní objectName (tedy jméno) všech widgetů, se kterými je zamýšleno dále pracovat. Právě jméno widgetu (nikoli název kterým jsou zmíněny v Object Inspectoru) je používáno, pokud chceme na widget referovat v kódu.

V průběhu návrhu layoutu v designeru je možné kontrolovat vzhled okna zkratkou „ctrl+R“, která vyvolá okno tvořené GUI aplikace v takové podobě, jak bude vypadat při běhu programu. Okno má stejnou velikost jako při práci v designeru



Obr. 5.4: Záložka Results

a díky důslednému využíváni posuvníků a layoutů se není třeba obávat změn jeho velikosti za chodu aplikace.

5.3 Struktura programu

Po dokončení návrhu GUI bylo nutné jej zprovoznit tak, aby widgety fungovaly dle jejich účelu. Tato fáze už probíhá mimo Qt Designer, totiž v prostředí PyCharm, sloužící jakožto editor zdrojového kódu. Přestože změny v layoutu GUI v průběhu programování jeho funkcí je spíše komplikací, bylo k několika takovým změnám přistoupeno s ohledem na měnící se požadavky na GUI. Pro lepší udržitelnost programu byl projekt rozdělen do čtyř souborů, jejichž funkce jsou popsány níže.

5.3.1 `gui.py`

Tento soubor obsahuje to, co vygeneroval `pyuic` z výše popsaného „.ui“ souboru. Jedná se o třídu s názvem `Ui_MainWindow`, ve které jsou popsány veškeré widgety, které byly nastaveny v designeru. Při interpretaci Pythonem tedy tento soubor vyvolá přesně to okno, které bylo vytvořeno v designeru. `Gui.py` je použit jako modul pro soubor `main.py`.

5.3.2 `connector.py`

Komunikace mezi Ansysem a GUI byla zajištěna modulem `connector.py`. Obsahuje třídu `Ansys_comm`, která zahrnuje veškeré funkce související s Ansysem - získání parametrů pro první záložku, vytvoření reportů pro srovnávací grafy na začátku i na konci optimalizace, vytvoření kopie modelu a změnění jeho rozměrů na ty, které vyjdou jako optimální. Také je zde obsažena metoda `optimize`, která je jádrem celé optimalizace. Vychází ze skriptu, který byl dodán spolu ze zadáním a jeho psaní není součástí této práce. V modulu se dále nachází třída `Timer` pro stopky a třída `Progresser` pro progress bar. Takto jsou všechny třídy, jejichž instance vykonávají dlouho běžící tasky (vyžadující vlastní vlákna), pohromadě v jednom modulu. V souvislosti s těmito tasky byly využity tzv. signály a sloty z knihovny PyQt5, umožňující spouštět tasky sekvenčně - task se spustí až po skončení předchozího tasku.

5.3.3 `main.py`

Soubor `main.py` se stal hlavním skriptem celého programu. Nachází se v něm třída `Main Window` a blok kódu pro vytvoření a spuštění GUI. Mezi atributy této třídy patří různé pomocné proměnné a instance tříd definovaných v modulu `connector.py`. Tyto instance jsou při spuštění optimalizace předány do vláken, čímž je zajištěn hladký chod programu, aniž by okno „zamrzávalo“. Jelikož je v `main.py` importován modul `gui.py`, lze z `main.py` načítat hodnoty zadané uživatelem do widgetů v GUI. Tyto hodnoty jsou po spuštění optimalizace načteny z widgetů do vhodných datových struktur a poslány do SOMA algoritmu. Dále jsou v modulu definovány funkce GUI - funkce pro vyvolání dialogového okna pro výběr modelu, vytvoření widgetů podle parametrů vybraného modelu, vytváření a skrývání widgetů podle stavu check boxu v první záložce a zobrazení grafů.

5.3.4 `algorithm_functions.py`

Tento modul byl stejně jako originální optimalizační skript dodán spolu se zadáním práce. Je importován do modulu `connector.py`, jehož funkce `optimize` volá funkce z `algorithm_functions.py` stejně, jako tomu bylo u originálního skriptu. Soubor byl ponechán beze změn.

5.4 Chod programu

Spuštěním `main.py` je vlastně spuštěna smyčka událostí hlavního okna aplikace (první vlákno), čekající na nějaký vstup uživatele. Kliknutím na tlačítko LOAD je

vyvoláno dialogové okno vybízející ke zvolení modelu. Po jeho zvolení jsou rozměry, které je možné optimalizovat, vypsány v první záložce. V tuto chvíli je možné kliknout na tlačítko START. Po kliknutí jsou načteny parametry zadané v GUI do datových struktur, které jsou zkopírovány do atributů instancí třídy `Ansys_comm`. Jelikož je funkce `optimize` metodou téže třídy, je možno spustit nad takovou instancí optimalizaci, protože parametry optimalizace jsou uloženy právě v atributech této instance. Jsou spuštěna také vlákna stopek a progress baru. Obojí musí běžet ve vlastním vláknu, protože jsou to tasky běžící po celou dobu optimalizace. Každý dlouho běžící task (zvláště takový, co má běžet zároveň s dalšími) potřebuje vlastní vlákno. V GUI aplikace se tato skutečnost projevuje „zamrznutím“ aplikace, pokud běží takový task v hlavním vláknu spolu s GUI. Dále je spuštěno vlákno pro vytvoření reportů v Ansysu. Po skončení tohoto procesu je vyslán signál `finished`, pomocí kterého se zavolají funkce na updatování grafů momentu a indukovaného napětí v čase a vytvoření nového vlákna, ve kterém poběží optimalizace. Po skončení optimalizace signál `finished` vyvolá funkci pro vytvoření vlákna pro finální reporty pro grafy optimálního modelu. Proces tvoření prvních reportů a proces optimalizace běží sekvenčně kvůli zamezení tzv. „race condition“, což je stav, ke kterému dojde pokud k datům přistupuje proces, aniž by se změnami těchto dat byl hotov předchozí proces. Jinými slovy v krajním případě, kdy byla optimalizace spuštěna s velmi nízkými nároky (populace = 1, migrace = 1) a trvala tedy krátkou dobu, mohlo dojít k tomu, že se finální reporty začaly vytvářet aniž by byly hotovy vstupní reporty. Aby tomuto stavu bylo zamezeno, jsou tasky vykonávány jeden po druhém, což není velký rozdíl od toho kdyby se první analýza a optimalizace prováděly současně vzhledem k tomu, jak krátce trvá analýza se srovnáním s optimalizací spuštěnou s nějakými smysluplnými parametry. Stopky jsou navrženy tak, aby začaly čítat po kliknutí na START a skončily po dokončení finálního reportu. Label se stopkami je updatován každou vteřinu inkrementací o jedničku. Progress bar je updatován každých deset vteřin, ne vždy je inkrementován. Inkrementace progress baru je škálována podle vstupních parametrů SOMA algoritmu.

5.5 Úprava optimalizačního skriptu

Originální skript `SOMA_mxwl.py` byl jen s drobnými změnami zabalen do metody `optimize` a použit. Jedna ze změn spočívala v naplnění datových struktur, se kterými tento skript pracoval. Původně si parametry bral z excel souboru, nacházejícího se ve stejném adresáři jako skript. Do odpovídajících proměnných stačilo přiřadit proměnné naplněné hodnotami z GUI. Při jejich naplňování bylo nutné dbát na to, aby byly právě toho formátu, jako proměnné v originálním skriptu a také dodržet pořadí parametrů. Dále byly ze skriptu odstraněny řádky, související s naplněním

těchto proměnných daty z excel souboru. Nakonec byl upraven obsah proměnných s názvem projektu a designu, který je optimalizován. Původně tam byly dosazeny doslova. Nicméně v GUI je uživateli umožněno vybrat jakýkoli model chce a této skutečnosti bylo nutné obsah těchto proměnných přizpůsobit (byť v současném stavu program pracuje úspěšně pouze se zadaným modelem motoru).

5.6 Výstupy programu

Jako výstupy programu jsou zde brány všechny výsledky, ze kterých lze posoudit úspěšnost optimalizace. V originálním programu to byl především excel soubor obsahující různé informace popsané níže. K tomu byly přidány grafy zobrazující se přímo v GUI a také kopie originálního modelu s optimálními parametry.

5.6.1 Vizuální interpretace výsledků

Po dokončení optimalizace jsou zobrazeny celkem tři grafy - průběh momentu v čase, průběh indukovaného napětí v čase a hodnota cenové funkce v závislosti na migraci (fázi optimalizace). První dva z nich jsou v průběhu optimalizace jednou updatovány - objeví se v nich průběhy originálu optimalizovaného modelu po dokončení vstupní analýzy. Srovnávací průběh již optimalizovaného modelu se zobrazí po dokončení finální analýzy. Na prvním grafu je možno zhodnotit, zda došlo ke zvýšení průměrného momentu a zároveň snížení jeho zvlnění. Na druhém grafu pozorujeme změny THD. V grafu hodnot cenové funkce se pro každou migraci bere ohodnocení nejlepšího jedince dané migrace. Vidíme na něm, jak se jedinci s každou migrací přibližují optimálnímu řešení. Data pro grafy momentů a napětí jsou získávána z vyexportovaných „.csv“ souborů z reportů v Ansysu. Data pro graf cenové funkce jsou získána z výstupního „.xlsx“ souboru.

5.6.2 Excel soubor

Po skončení optimalizace je vytvořen excel soubor s popisem průběhu optimalizace. V prvním listu je seznam jedinců populace vygenerované na začátku optimalizace. Pro každého jedince jsou vypsány jeho hodnoty všech optimalizovaných rozměrů, moment, zvlnění momentu, indukované napětí, THD a jeho cenové ohodnocení. Následuje list ve kterém jsou vypsáni pouze nejlepší jedinci z každé migrace. Na dalších listech jsou vypsány tytéž informace pro průběh každé migrace každého jedince.

5.6.3 Optimální model

Původně program optimální parametry zapisoval pouze do excel souboru. Tato funkce byla rozšířena na zkopírování originálního modelu a dosazení optimálních parametrů, takže po skončení optimalizace je k dispozici jak model originálu, tak optimálního modelu.

6 Optimalizace IPM

V této kapitole je popsán způsob, jak dosáhnout optimálního modelu motoru za pomocí popisovaného programu. Ačkoli implementace samotného optimalizačního algoritmu nebyla předmětem této práce, skutečnost že GUI plní své funkce bude demonstrována právě na optimalizaci zadaného elektromotoru. Budou popsány parametry zadané programu a interpretovány jeho výstupy.

6.1 Jak program používat

Původní program byl volán z příkazové řádky. Se současnou verzí je to možné taky, ale více příhodné je spouštění pomocí „.exe“ souboru vytvořeného z hlavního skriptu pomocí nástroje `pyinstaller`. Níže jsou popsána doporučení jak s programem nakládat dále.

6.1.1 Instrukce

Po spuštění programu optimalizaci nelze spustit, dokud není vybrán vhodný soubor (model motoru vymodelovaný v programu Ansys Electronics Desktop). Je třeba aby byl model ve stejném adresáři jako program, některé funkce s tím počítají. Po načtení modelu je možné zvolit rozměry, které se budou účastnit optimalizace. Ty, které nebudou vybrány, zůstanou nezměněny. Dále je možné upravit minimální a maximální hodnoty, které mohou rozměry nabýt. Program vytvoří jakýsi odhad na základě originální hodnoty rozměru a je vhodné aby uživatel zkontroloval, zda tento odhad odpovídá jeho představám. Toto platí i pro všechny ostatní nastavitelné parametry. Nastavení grafů je také možné změnit, ve výchozím nastavení je čas analýzy nastaven na jednu periodu signálu o frekvenci 50 Hz. Krok i čas analýzy ovlivňují délku analýzy, nicméně to je v porovnání s délkou provádění optimalizace natolik zanedbatelné, že není důvod proč si nenechat zobrazit hladké grafy (s nízkým krokem). Navíc je třeba dávat pozor na to, aby krok byl dostatečně nízký a nedocházelo k aliasingu (zkreslení signálu při nevhodně zvolené frekvenci vzorkování). Volení hodnot ideálního momentu, zvlnění momentu, THD a jejich váhových koeficientů závisí na parametrech optimalizovaného stroje a představách uživatele. Podle těchto hodnot je v průběhu optimalizace dopočítávána hodnota účelové funkce pro jednotlivé jedince.

6.1.2 Souvislosti se SOMA algoritmem

Nutno zdůraznit, že kvalita optimalizace závisí na správně zvolených parametrech SOMA algoritmu. Jedním z nich je výše zmíněná počáteční populace, udávající po-

čet jedinců, kteří budou náhodně rozmístěni ve stavovém prostoru (tzn. budou mít náhodné hodnoty všech optimalizovaných rozměrů v rámci povolených mezí). Čím bude populace větší, tím je větší možnost nalezení globálního minima účelové funkce a tedy dojde k maximální možné optimalizaci. Nevýhodou je zvyšující se časová náročnost. Pro počet migrací rovněž platí, že s jejich zvyšujícím se počtem roste šance nalezení nejlepšího řešení, ale složitost také poroste. Parametr PathLength udává vzdálenost, do které každý jedinec migruje ze své původní pozice směrem k nejlépe ohodnocenému jedinci. Jedinec je při své migraci ohodnocen na místech daných parametrem Steps, dokud nedojde do místa ve vzdálenosti PathLength. Parametry PathLength a Steps je tedy dán počet ohodnocení jedince během jedné jeho migrace - těmito parametry lze tedy také ovlivnit kvalitu a trvání optimalizace. Při zvyšování parametru PathLength anebo snižování parametru Steps tedy dojde ke zvýšení celkového počtu ohodnocovaných kombinací optimalizovaných parametrů. Každé toto ohodnocení zvýší celkovou dobu optimalizace a zároveň zvýší šanci najít nejlepší řešení. Parametr PRT může nabývat hodnot od 0 do 1 včetně a udává nahodilost v migraci jedinců směrem k vůdci.

6.2 Optimalizace

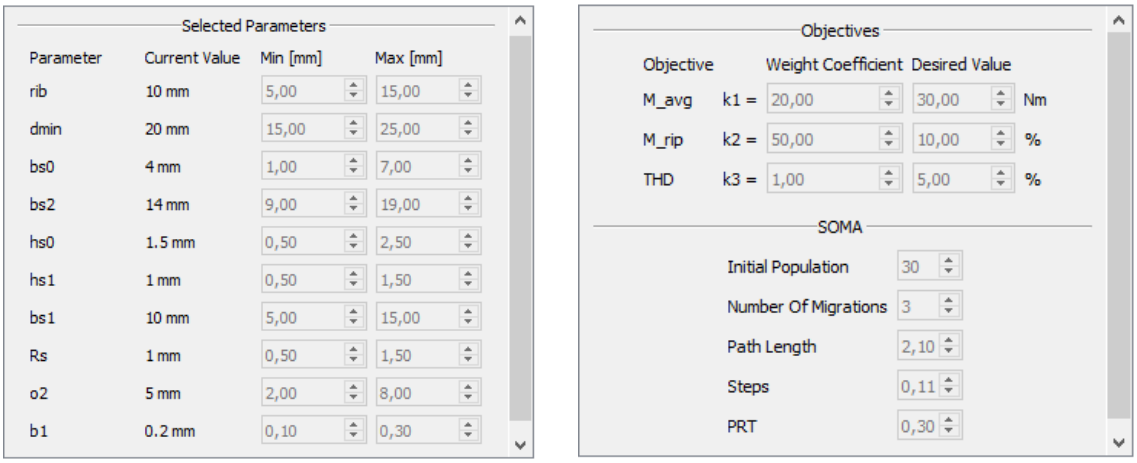
Ukázka funkcí programu bude předvedena na optimalizaci motoru typu IPM. Konkrétní model byl zvolen proto, že originální program byl psán právě pro něj a některé funkce jsou tedy „psány na míru“.

6.2.1 Vstupní parametry

Optimalizace se budou účastnit parametry uvedené v tabulce 4.2. Meze jednotlivých parametrů vstupující do optimalizace lze vidět na výstřižku obrazovky z GUI na obrázku 6.1 a).

Zadané hodnoty momentu, zvlnění momentu, THD a jejich váhové koeficienty jsou zobrazeny na obrázku 6.1 b).

Podle tabulky 4.1 je jmenovitý moment motoru 28 Nm, jako cílový moment byl zvolen moment 30 Nm. Jako nejdůležitější objektivum (nejvyšší váhový koeficient) bylo zvoleno zvlnění momentu s cílovou hodnotou 10 %. Třikrát migrující populace o 30 jedincích by měla být dostatečná k nalezení řešení lepšího, než je to stávající. Dle parametrů PathLength a Steps bude při každé migraci každého jedince ohodnoceno 20 různých řešení. Hodnota PRT zde udává, že je pro každého jedince pravděpodobnost odchylky při migraci 30 %.



(a) (b)

Obr. 6.1: a) Rozměry pro optimalizaci, b) Parametry optimalizace

6.2.2 Výsledek optimalizace

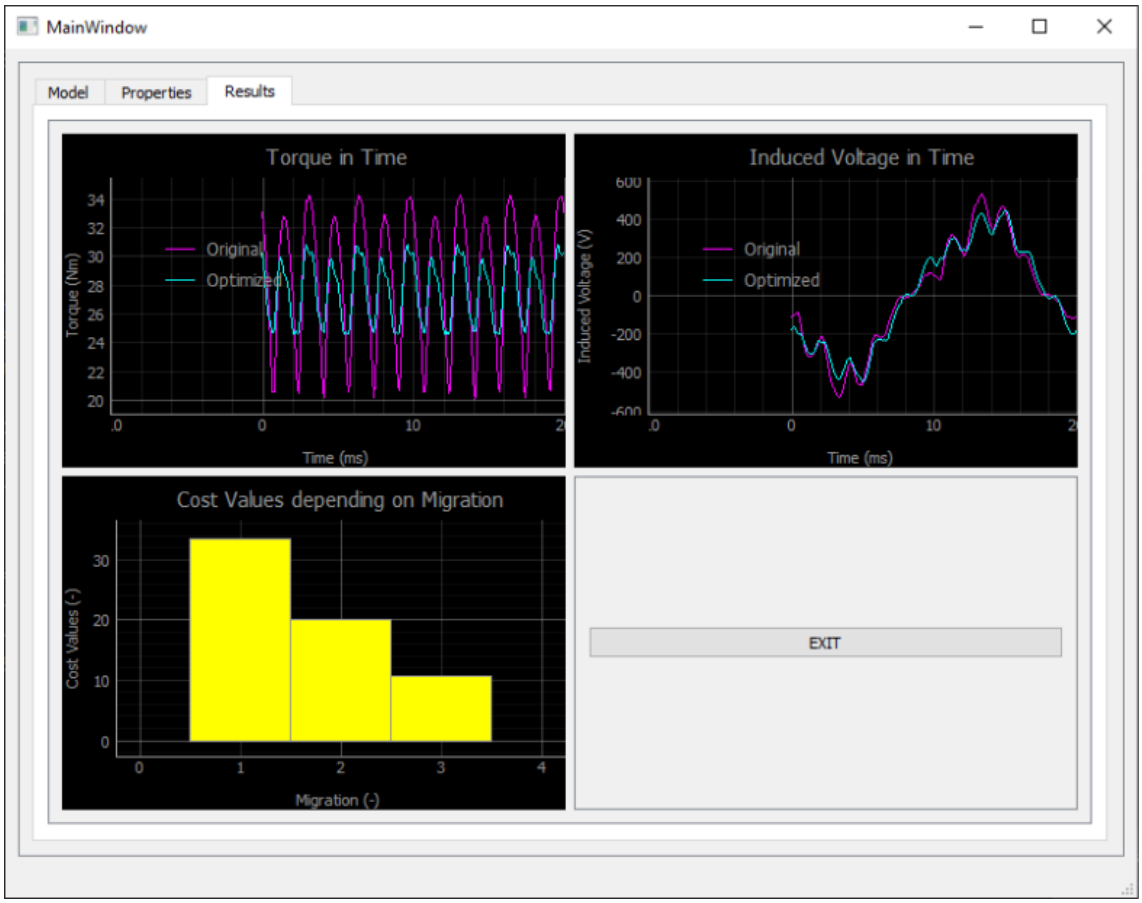
Program úspěšně doběhl v čase 18:45:55 od spuštění. Výsledné grafy je možné vidět na obrázku 6.2, který je výstřižkem vytvořené GUI aplikace. Konkrétní hodnoty jsou zapsány v tabulce 6.1, tyto byly převzaty z výstupního excelového souboru.

Z prvního grafu je patrné, že výsledný model má menší zvlnění momentu než ten originální. Z hodnoty 24.36 % klesl na 10.74 %, což je hodnota blížící se nastavené hodnotě 10 %. Průměrný moment se přitom snížil z 28.13 Nm na 27.65 Nm. Hodnota THD stoupla z původních 14.22 % na 14.66 %. Na posledním grafu lze vidět vývoj nejnižší hodnoty účelové funkce v závislosti na migraci. Ta podle očekávání s každou migrací klesla.

Objektivum	M_{avg}	M_{rip}	THD	$Cena$
Výchozí geom.	28.13	24.36	14.22	136.9
Optim. geom.	27.65	10.74	14.66	10.74
Váha	20	50	1	-
Požadovaná hodnota	30	10	5	-

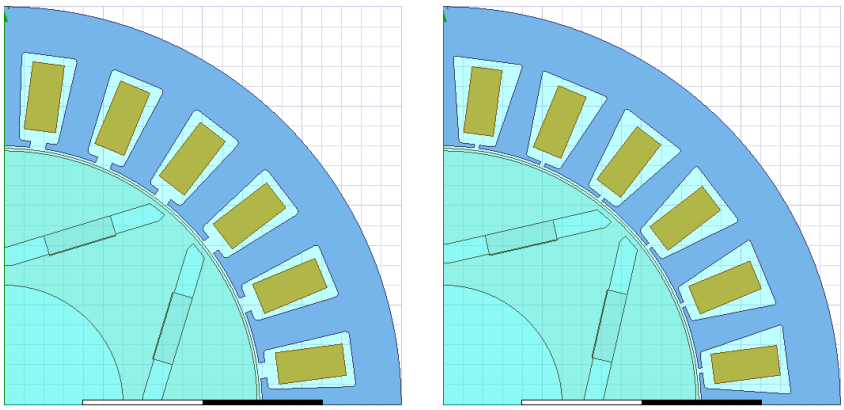
Tab. 6.1: Výchozí hodnoty [19], optimální hodnoty, parametry optimalizace

Je zřejmé, že optimalizace byla skutečně provedena s důrazem na snížení zvlnění momentu. Průměrná hodnota momentu a THD zůstaly téměř beze změny. U THD to lze vysvětlit tím, že vzhledem k zadanému nízkému váhovému koeficientu nebylo dostatečně „důležité“. U průměrného momentu byla nastavena jeho žádaná hodnota



Obr. 6.2: Výsledek optimalizace

natolik blízká té originální, že i řešení s hodnotou momentu blízkou té originální mohlo být snadno ohodnoceno velmi dobře.



(a) (b)

Obr. 6.3: a) Výchozí geometrie, b) optimální geometrie

Na obrázku 6.3 a v tabulce 6.2 lze vidět srovnání původního a optimálního modelu, jsou zde patrné drobné změny v geometrii. Význam jednotlivých rozměrů v rámci modelu lze vidět na obrázku 4.3.

Parametr [mm]	Rib	dmin	b1	o2	b_{s0}	b_{s1}	b_{s2}	h_{s0}	h_{s1}	R_s
Výchozí geom.	10	20	0.2	5	4	10	14	1.5	1	1
Opt. geom	5	21.2	0.2	5.7	1.2	9.4	17.5	0.8	0.8	0.6

Tab. 6.2: Výchozí a optimalizované rozměry

Závěr

Jedním z cílů práce bylo vytvoření grafického uživatelského prostředí pro program vykonávající optimalizaci elektromotoru. Tohoto cíle bylo dosaženo za použití zvoleného jazyka, což byl Python. Tento byl zvolen z důvodu jeho použití v samotném programu pro optimalizaci elektromotoru, pro který bylo grafické prostředí tvořeno. Dalším důvodem je jeho snadná propojitelnost s robustním frameworkem Qt, který byl použit pro programování samotného grafického prostředí. Qt byl vybrán také z důvodu dostupnosti designeru, který programování grafického prostředí značně usnadňuje. Vzhledem k tomu, že program optimalizující elektromotor byl již plně funkční před započetím této práce, bylo nutné se nejprve seznámit s jeho funkcemi důležitými pro vytvoření GUI, především tedy s formátem jeho vstupů. V okamžiku, kdy byl hotový grafický layout aplikace vytvořený v designeru, bylo přistoupeno ke stěžejní části této práce - propojení GUI s programem, při kterém šlo především o nasměrování vstupů z GUI do programu. Zásadním rozhodnutím bylo rozdělit téměř hotový GUI program, který byl do té doby jediným souborem (+ soubory originálního programu), na moduly. V jediném souboru byl původně program udržován z důvodu nedostatečných zkušeností s programováním, nicméně ve fázi téměř hotového programu se tento přístup projevoval jako čím dál méně udržitelný.

Na zadaném motoru typu IPM byla provedena optimalizace za pomocí zadaného programu doplněného o grafické prostředí. Byla tak demonstrována funkčnost nového programu zahrnující jak původní funkce (optimalizace vybraného motoru, volba parametrů optimalizace) tak i rozšíření - volba rozměrů, které se optimalizace budou účastnit, srovnávací grafy, zaznamenání trvání a postupu optimalizace. Samotná optimalizace proběhla úspěšně. Byl tak vytvořen aktualizovaný model zadaného motoru s rozměry pozměněnými tak, že bylo dosaženo objektiv zadaných v rámci grafického prostředí programu. Zadaným objektivům přitom nový model odpovídá poměrně přesně s ohledem na jednotlivé váhové koeficienty objektiv.

Toto by mohlo být podnětem dále s programem experimentovat se snahou o další vylepšování modelu - dosažení co nejvyššího průměrného momentu za co nejnižšího zvlnění momentu a THD. V rámci této snahy by bylo vhodné pouštět program vícekrát za sebou a to jak s pozměněnými parametry tak i se stejnými - i při zadání identických parametrů program může skončit s poměrně odlišnými výsledky. Pro prohledávání stavového prostoru možných řešení modelu co nejvíce „zeširoka“ lze doporučit program spouštět s vysokým počtem jedinců počáteční populace a širokými intervaly hodnot, kterých mohou optimalizované rozměry nabýt. Naopak je-li vhodné nějaké řešení prozkoumat blíže, program lze spustit s intervaly rozměrů okolo hodnot modelu, který nás zajímá. Co se týče ideálního řešení, tak žádný optimalizovaný model nelze s jistotou označit za ten nejlepší, neznáme totiž všechna existující

řešení. Cílem je spíše se co nejvíce přiblížit zadaným požadavkům na model.

Závěrem je také nutno zmínit, že v současné stavu je program vhodný pouze pro konkrétní zadaný model motoru. Na tuto práci by tedy mohlo být navázáno rozšířením v podobě zobecnění programu na všechny točivé stroje. Dále mohlo být uvažováno o přizpůsobení programu i pro jiné platformy, ale vzhledem k tomu, že je program závislý na Ansysu a ten běží bez potíží pouze na Windows, takové rozšíření nemá smysl.

Literatura

[1] *TIOBE Index for December 2019.* [online]. TIOBE [online]. Eindhoven, 2019 [cit. 2019-12-24]. Dostupné z URL: <https://www.tiobe.com/tiobe-index/>.

[2] *Characteristics of 'C' Language.Qt Documentation* [online]. TekSlate [online]. Richmont [cit. 2019-12-25]. Dostupné z URL: <https://tekslate.com/characteristics-of-c-language>.

[3] *Qt Documentation* [online]. Espoo: Free Software Foundation, poslední aktualizace 12. 12. 2019 [cit. 2019-12-23], Dostupné z URL: <https://doc.qt.io/>.

[4] TRANTER, Jeff. *Using Qt with Alternative Programming Languages - Part 1.* [online]. Integrated Computer Solutions [online]. Waltham, 2015, 19. 8. [cit. 2019-12-23]. Dostupné z URL: <https://www.ics.com/blog/using-qt-alternative-programming-languages-part-1>.

[5] *General FAQ.* [online]. WxWidgets - Cross-Platform GUI Library [online]. [cit. 2021-5-3]. Dostupné z URL: <https://www.wxwidgets.org/docs/faq/general/#support>.

[6] KOSTELNÍK, Pavel. *Tvorba moderních multiplatformních rozhraní v jazyce Python.* [online]. Brno: Mendelova univerzita v Brně., 2013. Diplomová práce. Dostupné z URL: <https://docplayer.cz/46371361-Tvorba-modernich-multiplatformnich-rozhrani-v-jazyce-python.html>.

[7] OUSTERHOUT, John. *History of Tcl.* [online]. Tcl Developer Xchange [online]. [cit. 2019-12-24]. Dostupné z URL: <http://www.tcl.tk/about/history.html>.

[8] *Your First Cup.* [online]. Oracle Help Center [online]. Redwood Shores, 2012 [cit. 2019-12-24]. Dostupné z URL: <https://docs.oracle.com/javaee/6/firstcup/doc/gcrky.html>.

[9] *Python Developer's Guide.* [online]. Python [online]. [cit. 2021-5-4]. Dostupné z URL: <https://devguide.python.org/#status-of-python-branches>.

[10] *Kivy [online].* [cit. 2021-5-4]. <https://kivy.org/#home>.

[11] VAN ROSSUM, Guido. *Foreword for "Programming Python"(1st ed.).* [online]. Python.org [online]. Reston, 1996 [cit. 2019-12-24]. Dostupné z URL: <https://www.python.org/doc/essays/foreword/>.

[12] VAN ROSSUM, Guido. *Glue It All Together With Python* [online]. Monterey, Kalifornie, 1998. Dostupné z URL: <https://www.python.org/doc/essays/omg-darpa-mcc-position/>.

[13] PRANTL, Martin. *Creating a Very Simple GUI System for Small Games - Part I.* [online]. GameDev.net [online]. 2014, 6. 5. [cit. 2019-12-24]. Dostupné z URL: <https://www.gamedev.net/articles/programming/general-and-gameplay-programming/creating-a-very-simple-gui-system-for-small-games-part-i-r3652/>.

[14] *Top 5 Best Python GUI Libraries.* [online]. AskPython [online]. [cit. 2021-5-4]. Dostupné z URL: <https://www.askpython.com/python-modules/top-best-python-gui-libraries>.

[15] ONDRŮŠEK, Čestmír. *Elektrické stroje* [online]. Brno: VUT [cit. 2019-11-29]. Dostupné z URL: <https://moodle-archiv.ro.vutbr.cz/pluginfile.php/469145/mod_resource/content/2/skripta_BESB.pdf>.

[16] FUSEK, Martin , a Radim HALAMA. *MKP a MHP* [online]. Ostrava: VŠB, 2011 [cit. 2019-12-25]. Dostupné z URL: <http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/metoda_konecnych_prvku_a_hranicnich_prvku.pdf>.

[17] LEHIKOINEN, Antti. *Cross-section of a 37 kW induction machine.* [online]. In: Antti Lehikoinen [online]. Espoo [cit. 2019-12-25]. Dostupné z URL: <https://www.anttilehikoinen.fi/wp-content/uploads/2016/04/mesh.png>.

[18] VOLNÁ, Eva. *Evoluční algoritmy a neuronové sítě* [online]. Ostrava: Ostravská univerzita v Ostravě, 2012 [cit. 2019-12-26]. Dostupné z URL: <http://physics.ujep.cz/~mmaly/vyuka/MPVT_II/Heuristiky/SOMAdetail-Evolucni_algoritmy_a_neuronove_site+Genetika.pdf>.

[19] KNEBL, Ladislav. *Optimization of Interior Permanent Magnet Synchronous Motor Using Evolutionary Optimization Algorithm. In: Proceedings of the 25st Conference STUDENT EEICT 2019* [online]. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2019, s. 664-668 [cit. 2021-5-4]. ISBN 978-80-214-5735-5. Dostupné z URL: <http://hdl.handle.net/11012/186755>.

[20] KOPECKÝ, Jan. *Návrh synchronního spoke motoru.* . Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. Vedoucí práce Ing. Ladislav Knebl.

[21] BARCARO, Massimo. *PMSM structure* Design Analysis of Interior Permanent Magnet Synchronous Machines for Electric Vehicles. Padova, 2011. Doctoral thesis. Universita degli studi di Padova. Supervisor Ch.Mo Prof. Nicola Bianch

Seznam symbolů a zkratek

GUI grafické uživatelské rozhraní – Graphical User Interface

IDE integrované vývojové prostředí – Integrated Development Environment

SOMA samo-organizující migrující algoritmus – Self-Organising Migrating Algorithm

PMSM synchronní motor s permanentními magnety – Permanent Magnet Synchronous Motor

IPM magety zapuštěné v rotoru – Interior Permanent Magnet

SPM magnety na povrchu rotoru – Surface Permanent Magnet

PMSM synchronní motor s permanentními magnety – Permanent Magnet Synchronous Motor

SMPM permanentní magnet na povrchu rotoru – Surface Mounted Permanent Magnet

SRAM synchronní reluktanční motor – Synchronous Reluctance Assisted Motor

IPMSM synchronní motor s permanentními magnety zapuštěnými v rotoru – Interior Permanent Magnet Synchronous Motor

FEM metoda konečných prvků – Finite Element Method

THD celkové harmonické zkreslení – Total Harmonic Distortion