Vizualizace základních algoritmů řízení

Dominik Pavlica

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Dominik Pavlica**
Osobní číslo:	**A19581**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**Prezenční**
Téma práce:	**Vizualizace základních algoritmů řízení**
Téma práce anglicky:	**Visualization of Basic Control Algorithms**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Nastudujte a rámcově popište princip vytváření tzv. „live scripts" a grafických uživatelských rozhraní (GUI) v programovém systému MATLAB.
3. Vysvětlete princip činnosti základních typů regulátorů: P, PI, PD, PID a 2-polohové regulace.
4. Vyberte vhodné typy dynamických systémů pro názornou demonstraci činnosti uvedených algoritmů řízení a tyto implementujte do programového systému MATLAB/Simulink.
5. Navrhněte a realizujte formou tzv. „live scripts" či GUI několik ukázkových příkladů ilustrujících princip činnosti uvedených typů regulátorů při řízení vybraných systémů.
6. Každou demonstraci doplňte o vhodnou grafickou vizualizaci/animaci a také možnost interaktivně měnit parametry simulace uživatelem – např. typ regulátoru a jeho nastavení.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. BALÁTĚ, J. *Automatické řízení*. Praha: BEN – technická literatura, 2003.
2. NOSKIEVIČ, P. *Modelování a identifikace systémů*. Ostrava: Montanex, 1999.
3. FARANA, R. et al. *Programová podpora simulace dynamických systémů: sbírka řešení příkladů*. Ostrava: VŠB-Technická univerzita, 1996.
4. PERŮTKA, K. *MATLAB – základy pro studenty automatizace a informačních technologií*. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005.
5. ZAPLATÍLEK, K. *MATLAB: grafické uživatelské rozhraní*. Brno: Tribun EU, 2020.
6. MATHWORKS. Live Scripts and Functions –MATLAB & Simulink. In: *MathWorks.com* [online]. 2021 [cit. 2021-11-12]. Dostupné z: https://www.mathworks.com/help/matlab/live-scripts-and-functions.html.

Vedoucí bakalářské práce: **doc. Ing. František Gazdoš, Ph.D.**
Ústav řízení procesů

Datum zadání bakalářské práce: **15. ledna 2022**
Termín odevzdání bakalářské práce: **20. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne: 17.05.2022

Dominik Pavlica, v.r
podpis studenta

ABSTRAKT

Tato bakalářská práce se zabývá demonstrací vlivů základních typů regulátorů na vybraných dynamických systémech. Zobrazování vlivů je realizováno pomocí grafického uživatelského rozhraní (GUI), kde je možnost taktéž nastavovat parametry systému, zvoleného typu regulátoru, simulace a také sledovat animaci konkrétního modelu.

Teoretická část stručně popisuje prostředí MATLAB, tvorbu live scriptů, grafického uživatelského rozhraní (GUI) v prostředí App designer a simulací v prostředí Simulink. Následně jsou popsány vybrané typy regulátorů (P, PI, PD, PID a 2-polohový). Poslední kapitola v teoretické části se věnuje popisu vybraných dynamických systémů použitých pro demonstraci.

V praktické části je nastíněn popis vlastní tvorby simulačních schémat pro prostředí Simulink, následovaný popisem vývoje grafického uživatelského rozhraní (GUI) včetně ukázky finální aplikace a požadavků na její úspěšný provoz.

Klíčová slova: regulátory, vizualizace, GUI, MATLAB, Simulink

ABSTRACT

This Bachelor thesis deals with demonstration of the effects of basic types of controllers on selected dynamical systems. The display of effects is realized by means of a graphical user interface (GUI), where it is also possible to set the parameters of the system, selected type of a controller, simulation and also watch the animation of a specific model.

The theoretical part briefly describes the MATLAB environment, the creation of live scripts, the graphical user interface (GUI) in the App designer and simulations in the Simulink. Then, selected types of controllers (P, PI, PD, PID and 2-position) are briefly described. The last chapter in the theoretical part deals with the description of selected dynamical systems used for the demonstration.

The practical part outlines a description of the creation of simulation schemes for the Simulink environment, followed by a description of development of the graphical user interface (GUI) including also output examples of the final application and requirements for its successful usage.

Keywords: controllers, visualization, GUI, MATLAB, Simulink

Rád bych poděkoval doc. Ing. Františku Gazdošovi, Ph.D. za obrovskou ochotu během vypracovávání této práce, odbornou pomoc, vedení, trpělivost a cenné rady, které mi při vypracovávání této práce sděloval. Dále bych rád poděkoval členům rodiny a přátelům za podporu, kterou jsem od nich dostával při vypracovávání této bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Neustálý vývoj v oblasti automatizace a zrychlování výrobních procesů donutil lidstvo začít hledat způsoby, jak nahradit v určitých fázích řídící činnost člověka. Ten přestal být v náročnějších procesech přesný a efektivní. Proto se postupem času začaly zapojovat do provozu řídící algoritmy, které činnost člověka omezily nebo zcela nahradily.

Algoritmy řízení lze obecně rozdělit na spojité a nespojité a bývají realizovány tzv. regulátory.

U spojitých regulátorů neustále probíhá měření požadované veličiny, na kterou může regulátor v podstatě okamžitě reagovat. V technické praxi patří mezi nejpoužívanější spojité regulátory tzv. PID-regulátor (obsahující proporcionální, integrační a derivační složku). Ten je výhodný především svým jednoduchým nasazením na řízené systémy. Problém může ovšem nastat u komplexnějších procesů a náročnějších požadavcích na kvalitu řízení.

Na rozdíl od předešlého typu regulátoru, nespojitý regulátor nezasahuje do řízeného systém spojitě. Charakteristickou vlastností nespojitého regulátoru je to, že zásah do řízeného systému nabývá určitého počtu hodnot. Nespojité regulátory mají v praxi většinou jednoduchou konstrukci a jsou levné. Nevýhodou nespojitých regulátoru je to, že nejsou schopny udržet regulovanou veličinu na požadované hodnotě, ale typicky pouze okolo této hodnoty kmitají. V technické praxi patří mezi nejpoužívanější nespojité regulátory 2-polohový regulátor. Ten nabývá pouze dvou hodnot v závislosti na regulační odchylce (nejčastěji zapnuto/vypnuto).

V teoretické části této práce si vysvětlíme, jakým způsobem se pracuje s programovým prostředním MATLAB včetně jeho nástrojů pro tvorbu interaktivních dokumentů pomocí live scriptů, grafického uživatelského rozhraní pomocí App designeru a nástavby pro simulaci systémů Simulink. V další kapitole si vysvětlíme, jak pracuje regulátor a uvedeme si několik základních typů regulátorů z obou dříve vzpomínaných rozdělení. Poslední teoretickou části bude popis třech základních modelů dynamických systémů, na kterých celá tato práce staví svoji podstatu znázorňování činnosti jednotlivých typů základních regulátorů.

Praktická část obsahuje postup práce při implementování vybraných modelů do simulačního prostředí MATLAB/Simulink. Dále si představíme postup při tvorbě uživatelského grafického rozhraní (GUI) včetně zobrazení několika jeho částí. Důležitou věcí při používání vytvořené aplikace je znalost podmínek, za jakých tato aplikace může bez problému pracovat, proto zde budou rozebrány zásady používání této aplikace.

O tématu, které se zabývá základními algoritmy řízení systémů bylo napsáno již mnoho bakalářských a diplomových prací. Na různých webových portálech je k nalezení nespočet článků, které rozebírají vlivy jednotlivých regulátorů a jejich složek, proto zde uvedu alespoň několik reprezentativních prací, přičemž úplný výčet by přesahoval rozsah této bakalářské práce, která má primárně jiný cíl. Jedna z bakalářských prací, kterou napsal student Tomáš Hanuš z Fakulty strojního inženýrství na Vysokém učení technickém v Brně [1] měla za cíl simulovat činnost několika základních regulátorů na modelu stejnosměrného motoru. Student si zde zvolil několik typů diskrétních a spojitých PID regulátorů, které implementoval do prostředí Simulink společně s modelem stejnosměrného motoru. K možnosti ovládání parametrů zvolených regulátorů a zvoleného modelu zvolil cestu tvorby grafického uživatelského rozhraní (GUI), kde mimo jiné zobrazoval průběhy pozice motoru v závislosti na použitých regulátorech a následně tyto průběhy slovně popisoval. Dalším studentem, který se zabýval vlivy regulátorů na různé systémy je student Martin Bradáč z Fakulty strojního inženýrství na Vysokém učení technickém v Brně. [2] Tento student měl za úkol navrhnout a realizovat model inverzního kyvadla. Model navrhoval a simuloval v prostředí MATLAB/Simulink a chtěl ho řídit pomocí dvou PID regulátorů, pro které hledal ideální parametry jeho složek. V závěru své práce došel ke stanovisku, že použití dvou PID regulátorů není vhodné a jeden PID regulátor, který řídil balancování kyvadla by bylo potřeba nahradit stavovým regulátorem. Další zajímavou bakalářskou práci dělal student Milan Míšenský z Fakulty elektrotechniky a informatiky na Univerzitě Pardubice [3]. Cílem práce tohoto studenta bylo vytvořit grafické uživatelské rozhraní (GUI) v prostředí MATLAB pro laboratorní soustavu GUNT RT 010, což je soustava, která umožnuje měřit vlastnosti systému a provádět regulační experimenty. Student ve své práci zmiňuje práci s prostředím MATLAB včetně tvorby grafického uživatelského rozhraní (GUI). Student na laboratorní soustavě reguluje výšku hladiny pomocí PID regulátoru a data získána z tohoto modelu předává do grafického uživatelského rozhraní, kde na základě získaných dat simuluje změnu výšky hladiny v zásobníku.

Tato bakalářská práce se odlišuje od ostatních výše uvedených prací tím, že jsou zde zpracovány tři modely dynamických systémů najednou a na těchto modelech si uživatel může vyzkoušet řízení několika základními algoritmy řízení. Všechny simulace modelů jsou navíc obohaceny o vizuální animaci modelu. Velký důraz je především kladen na co největší umožnění měnit parametry zvolených soustav a regulátorů uživatelem. Aplikace by měla být přínosná především pro méně znalé uživatele v oblasti regulace, kteří by si z této aplikace

mohli odnést základní informace a praktické představy, jak probíhá taková regulace systémů pomocí základních typů regulátorů.

I. TEORETICKÁ ČÁST

1 PROGRAMOVÝ SYSTÉM MATLAB

Název MATLAB původně vznikl z anglického slovního spojení MATrix LABoratory. MATLAB jakožto programový balík je vyvíjen americkou společností MathWorks a byl představen v roce 1984.

Dnes je MATLAB používán miliony lidí k analýze dat, vývoji algoritmů a vytváření simulačních modelů. MATLAB poskytuje řešení v oblastech, jako je např. aplikovaná matematika, zpracování signálu a komunikace, počítačové vidění, strojové učení, robotika a v mnoha dalších oblastech. V praxi bývá hojně využíván v leteckém a automobilovém průmyslu. [4]

1.1 Verze MATLABu

Od roku 1984 kdy byl MATLAB představen uplynulo již bezmála 38 let. Za tuto dobu vyšlo nespočet nových verzí MATLABu. Zde bych rád uvedl několik verzí, které považuji za zásadní ve vývoji MATLABu včetně těch nejdůležitějších změn, které v dané verzi nastaly.

Jedná se o tyto verze [5]:

- MATLAB 1.0 – První verze MATLABu vydána v roce 1984.
- MATLAB 3.5 – Tato verze byla spustitelná na systému MS-DOS. Vydána v roce 1990.
- MATLAB 4 – Tato verze byla spustitelná na systémech Windows 3 a MAC. Vydána v roce 1992.
- MATLAB 5.0 – Verze, kterou bylo možné spustit na všech platformách. Vydána v roce 1996.
- MATLAB 6.5 R13 – Zahrnovala nové možnosti ohledně grafického uživatelského rozhraní pro import dat. Vydána v roce 2002.
- MATLAB 7.6 R2008a – Vylepšení pro objektově orientované programování. Vydána v roce 2008.
- MATLAB 7.12 R2011a – Přináší nové možnosti práce s čísly, a to díky nové funkci *rng*, která generuje náhodná čísla. Vydána v roce 2011.
- MATLAB 8.4 R2014b – Přidány vylepšené uživatelské nástroje, nové funkce a balíčky pro podporu programování (např. pro Python). Vydána v roce 2014.
- MATLAB 8.6 R2015b – Verze, která přinesla nové funkce pro práci s grafy. Vydána v roce 2015.

- MATLAB 9.0 R2016a – Přidán nástroj App Designer pro vytváření a navrhování aplikací. Zahrnovala i novou funkci, díky které bylo možné zastavit běh během provádění. Vydána v roce 2016.
- MATLAB 9.2 R2017a – Přidán cloudový MATLAB (MATLAB online). Vydána v roce 2017.
- MATLAB 9.6 R2019a – Přidání vylepšení pro umělou inteligenci a analytiku. Vydána v roce 2019.
- MATLAB 9.7 R2019b – Zahrnuje aktualizace umělé inteligenci, nové prvky pro modelování. Vydána v roce 2019.

1.2 O programu

MATLAB je interaktivní prostředí, což si lze představit jako komunikaci mezi uživatelem a programem. Uživatel zadá požadavek, který odešle systému MATLAB a ten na něho odpoví. Tímto si uživatel a prostředí vyměňují informace.

Základním předpokladem pro práci s prostředím MATLAB je využívání matematického jádra, které v MATLABu běží a stará se o základní numerické výpočty. Obsahuje funkce pro celočíselné operace, desetinné operace nebo např. komplexní čísla. [6]

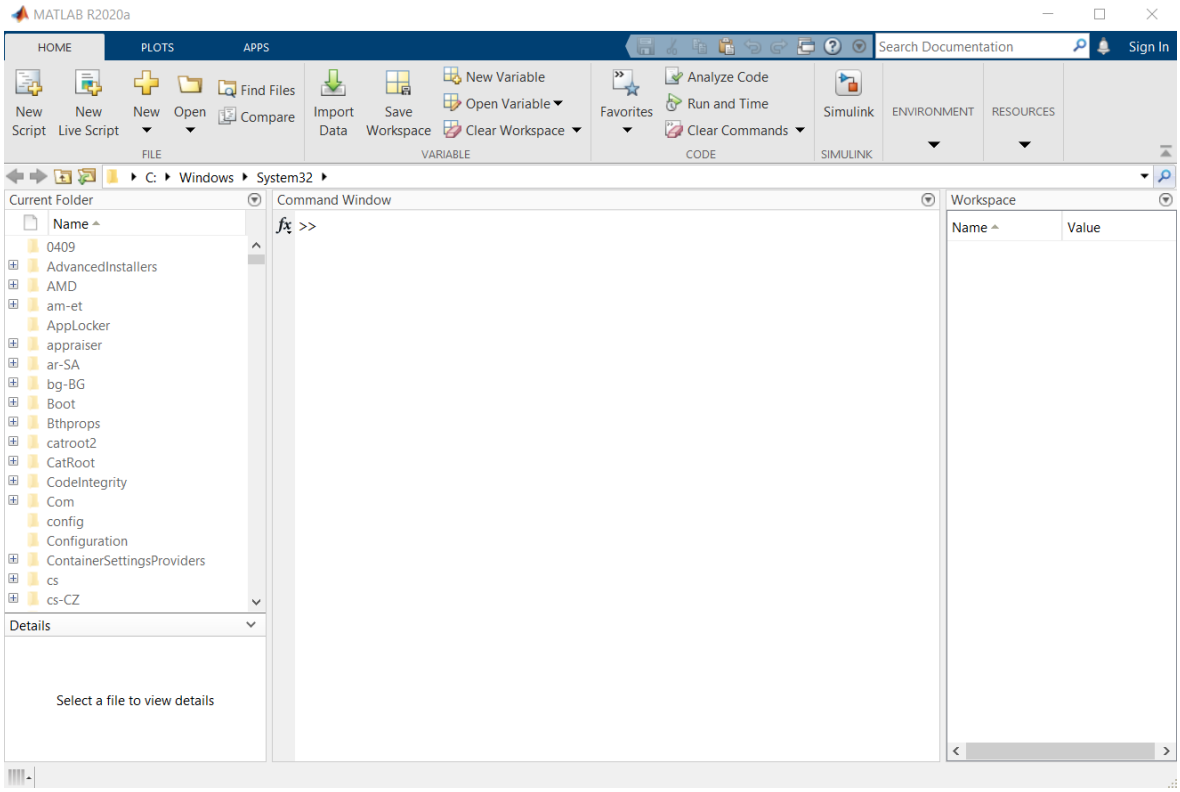
MATLAB není tak populární pouze kvůli matematickým operacím, které dokáže provádět, ale především kvůli svým aplikačním knihovnám, kterých má v dnešní době přes 100. Mimo aplikační knihovny nabízí také simulační prostředí Simulink, které je hojně využíváno k simulacím nejrůznějších modelů. Mezi nejznámější aplikační knihovny a toolboxy můžeme zařadit např. finanční analýzu a modelování, návrh a analýzu řídících systémů, měření a testování a spoustu dalších. [4]

MATLAB se za svou dobu působnosti na trhu podstatně rozvinul a nelze vyjmenovávat všechny možnosti, které nám matlab nabízí.

Veškeré informace, které jsou zde uvedeny pro popis prostředí MATLAB jsou primárně určené pro verzi R2020a. Využívané prostředky MATLABu, které jsou použité v této práci jsou dostupné i v mnoha starších nebo naopak novějších verzích MATLABu.

1.3 Prostředí programu

Po spuštění programu *MATLAB* dochází k zobrazení základního okna prostředí, které je ukázáno na Obrázku 1.



Obrázek 1 - Prostředí MATLAB po spuštění

Základní prostředí nám nabízí hned několik oken, kterým by bylo dobré věnovat pozornost. Záložce *HOME*, *PLOTS* a *APPS* se budeme věnovat samostatně v kapitole 1.3.1.

Current Folder

Okno, které je v základním nastavení na levé straně prostředí slouží pro zobrazení aktuální otevřené složky programem. Uživatel má zde možnost procházet libovolně dostupné složky, které má uložené v počítači a jednotlivé soubory otevírat, upravovat, přejmenovávat nebo například mazat. Pokud uživatel otevírá soubor, který nepatří pod systém *MATLAB*, tak je otevřen v podporovaném prostředí, které daný soubor podporuje. V případě otevírání souborů, které patří pod systém *MATLAB* se najde odpovídající prostředí a soubor se otevře v některém z prostředí *MATLABu*.

Command Window

Hlavní část základního prostředí tvoří okno, které by se doslovně dalo přeložit jako „příkazové okno“, což přesně charakterizuje funkcionalitu této části. Jedná se okno, které čeká na váš vstupní příkaz. Pokud zadáte podporovaný příkaz, tak dostanete zpětnou odpověď na váš požadavek. V případě, že zadáte neznámý výraz pro *MATLAB*, tak budete upozorněni, že zadaný příkaz nezná. Jelikož se jedná o příkazový řádek, tak zde nelze budovat nějaké souvislejší programové struktury. Pro rozsáhlejší programy jsou zde možnosti programů typu *script*, *live script* nebo *aplikace* (grafické uživatelské rozhraní). Těmto možnostem se tato práce věnuje podrobněji v kapitole 1.3.1, 1.4 a 1.5.

Workspace

Jedná se o poslední okno zobrazené v základním prostředí *MATLAB*, které dává uživateli přehled o proměnných, se kterými aktuálně pracuje. Proměnná má vždy svůj název, datový typ a hodnotu. Do *workspace* se data mohou přidat různými způsoby mezi které patří např. vložení dat z datových souborů uživatelem, voláním MATLABových funkcí, které si ukládání vyžadují. Posledním častým případem je, že si uživatel sám proměnné vytvoří, aby si do nich mohl ukládat svůj obsah, který bude chtít předávat mezi různými souborovými typy v prostředí *MATLAB*.

1.3.1 Menu

Náplní této práce není podrobné rozebírání prostředí *MATLABu* ovšem vnímám jako žádoucí zmínit podstatné funkcionality se kterými se velmi pravděpodobně každý uživatel tohoto nástroje dostane do styku. Souborovým systémům *live script, app* (grafické uživatelské rozhraní) a *simulink* se práce věnuje samostatně v podkapitole 1.4, 1.5 a 1.6.

Záložka home

Jednou ze tří záložek, které v úvodním dialogovém okně máme je záložka *home*. Ta je pro uživatele klíčová, protože nabízí ty nejdůležitější ovládací funkce, které uživatel pro práci potřebuje. Mezi tyto funkce patří ukládání souboru, vytváření nových souborů, otevírání již vytvořených souborů, vkládání dat, vytváření nových proměnných a spoustu dalších možností.

New Script

Pokud zvolíte možnost vytvoření nového scriptu, tak se otevře nové dialogové okno, které slouží pro správu souborového systému *script* s příponou *.m*.

Script obecně slouží pro sekvenční volání jednotlivých řádků kódu. Na rozdíl od dříve zmiňovaného *Command Window* je zde výhoda provedení všech řádků kódu jako celku.

New – Function

Otevře se nové dialogové okno, které slouží pro správu souborového systému *function* se stejnou příponou jako dříve zmiňovaný *script*. Rozdíl spočívá v označení daného typu souboru jako funkce.

Funkce vždy obsahuje klíčové slovo *function* což nám říká, že se jedná o funkci. Funkce má vždy svůj unikátní název a celá sekvence příkazů musí být ukončena klíčovým slovem *end*. Funkce může obsahovat návratové proměnné, které se zapisují před znak =, který je před názvem funkce. Návratové proměnné slouží k získání obsahu, který funkce vrátí. Dalším nepovinným parametrem jsou vstupní proměnné, které na rozdíl od návratových proměnných předávají do těla funkce hodnoty. Vstupní proměnné se zapisují do kulatých závorek za názvem funkce. Funkce umožnuje rovněž jako u *scriptu* sekvenční zapisování příkazů. Funkce je následně možné volat v jiných souborech.

New – Live Function

Otevře se nové dialogové okno, které slouží pro správu souborového systému *live function* s příponou *.mlx*.

Tento typ funkce představuje možnost, jak přehledně programovat jednotlivé funkce v kombinaci s formátovaným textem a následně tyto funkce používat v souborovém systém *live script.* Funkce i její volání je stejné jako již dříve zmiňovaná obyčejná funkce.

New – Class

Otevře se nové dialogové okno, které slouží pro správu souborového systému *class* s příponou *.m*.

Třídy slouží pro vytváření objektů, které zapouzdřují data a operace prováděné s těmito daty. Třída má strukturu, která obsahuje její název, sekci *properties*, kde jsou definované proměnné typu objekt. Následuje sekce *methods*, která obsahuje funkce, které pracuji s vytvořenými objekty.

Záložka plots

Druhou záložkou v úvodním dialogovém okně je záložka *plots*. Tato záložka nabízí možnosti pro rychlou tvorbu nejrůznějších grafů.

Selection

Zobrazuje názvy proměnných, které má uživatel označené pro vykreslení. Označování proměnných probíhá pouze tak, že si uživatel klikne na libovolnou proměnnou v sekci *workspace*. Uživatel má možnost označit i více proměnných a tím vytvořit kombinovaný graf.

Plots

Zobrazuje výčet dostupných grafů, které je možné vytvořit vzhledem ke zvoleným proměnným. Podstatné pro možnost vytvořit graf je datový typ proměnné včetně jejího obsahu. V případě nesmyslných požadavků na vykreslení může dojít k tomu, že žádný graf nebude nabízen.

Options

Uživatel má zde na výběr mezi možnostmi *reuse figure* nebo *new figure*. *Reuse figure* znamená, že pokud má uživatel již otevřené okno s vykresleným grafem a zvolí vykreslení nového grafu, tak původní graf bude překreslen novým grafem. *New figure* znamená, že vždy při novém požadavku na vykreslení grafu bude vytvořené nové okno s požadovaným grafem.

Záložka apps

Poslední ze tří záložek v základním dialogovém okně je záložka *apps*. Tato záložka dává uživateli možnost jednoduše doinstalovat nové aplikace do prostředí MATLAB.

File

V této sekci má uživatel možnost instalovat další aplikace do prostředí *MATLAB.* Aplikace slouží k ulehčení práce v dané problematice. Aplikace si může uživatel sám vytvořit nebo např. stáhnout z webových stránek společnosti *MathWorks*.

Apps

Přehled všech nainstalovaných aplikací v prostředí *MATLABu.* Uživatel má možnost danou aplikaci spustit a používat.

1.4 Live script

Takzvané *Live scripty* kombinují formátovaný text a grafiku s naprogramovanou funkcionalitou. Společně tvoří spojení, s kterým je možné pracovat v interaktivním prostředí nazvaném *Live editor*. [7]

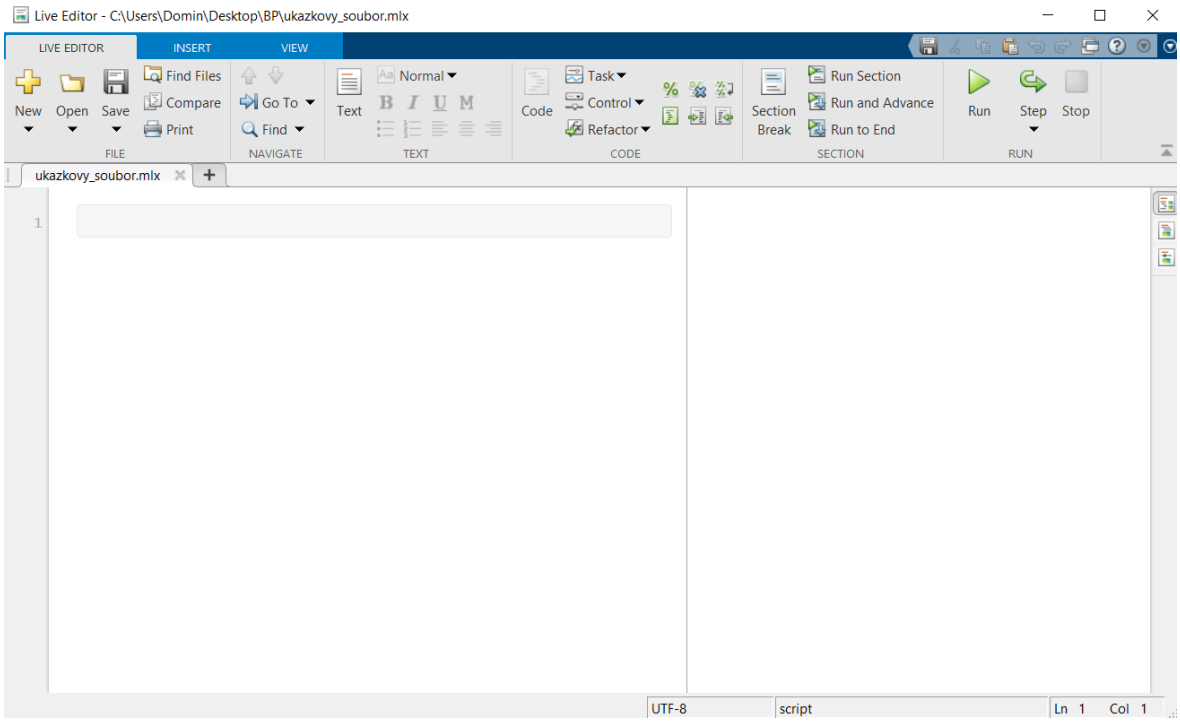
Live scripty je možné na první pohled rozeznat podle přípony souboru. Ty používají pro svoje ukládání tzv. *Live code* souborový systém s příponou *.mlx*. [7]

Velkou výhodou používání *live scriptů* není pouze kombinování kódů s grafikou a formátovaných textem, díky kterému může uživatel jednoduše popisovat danou problematiku a následně ji na implementovaném příkladě předvést včetně jejího výstupu, ale také zaručená kompatibilita novějších verzí *Live code* se staršími verzemi *Live code*. U *Live scriptů* je rovněž zaručena kompatibilita zobrazování znaků mezi různými jazyky, které má uživatel nastavené v *MATLABu* jako výchozí. [7]

Do prostředí *Live editoru* (vytvoření live scriptu) je možné dostat se dvěma způsoby:

1. Nejjednodušším způsobem je vytvořit nový soubor pomocí záložky *New – Live Script*
2. Druhým způsobem je zavolání příkazu *edit {název_souboru}.mlx* v *Command window*.

Po vytvoření *Live Scriptu* se otevře okno *Live Editoru*, které je ukázáno na Obrázku 2.



Obrázek 2 - Prostředí Live editoru po spuštění

Pro první spuštění *Live Scriptu* by měl uživatel znát tři základní věci:

1. V levé části dialogového okna se nachází okénko pro psaní kódu a formátovaného textu.
2. V pravé části dialogového okna se nachází okénko, které zobrazuje výstup ze spuštěného kódu *live scriptu*.
3. V horní části záložek se nachází nástroj pro spuštění *live scriptu (RUN)*. *Live script* lze spustit i po částech, a to pomocí postranních panelů, které se uživateli zobrazí vedle okénka pro psaní kódu v moment kdy napíše první řádek kódu.

Pokud si uživatel chce vyčistit výstupní dialogové okno od výstupu od předešlého spuštění, tak to může provést pomocí kliknutí pravým tlačítkem v pravém dialogovém okně a vybrat možnost vymazání konkrétního výstupu pomocí *Clear Output* nebo vymazat veškerý záznam v dialogovém okně pomocí *Clear All Output*.

1.4.1 Formátovaný text

Formátovaný text patří k hlavním aspektům, proč si vybrat zrovna Live Scripty v prostředí *MATLAB*. Díky formátovanému textu budí program dojmu, že se jedná o interaktivní dokument, kde si uživatel může nastudovat určitou problematiku a na příkladu rovnou vyzkoušet.

Formátovaný text má různé vizuální podoby, a ne vždy se musí jednat pouze o textovou formu, ale uživatel má možnost využít i výrazovou formu úpravy, obrázky nebo hyperlink (odkaz).

Formátovaný text měl vždy pevně stanovenou barvu písma, velikost písma a styl písma. Změna přišla ve verzi R2021a, kde si uživatel může styly písma změnit. Práce je zaměřena na starší verzi, proto na příkladech bude zmíněný základní styl písma.

Formátovaný text lze tvořit v programu několika způsoby. Prvním způsobem je použití speciálních znaků pro jednotlivé typy úprav. Druhým způsobem je použití klávesových zkratek. Posledním způsobem je použití horního menu v dialogovém okně.

Nyní budou představeny důležité typy a způsoby formátování textu.

Title

Neboli titulek se používá při tvorbě formátovaného textu. Titulek má největší styl písma, tučné písmo a má oranžovou barvu. Titulek je možné vytvořit pomocí znaku *# {váš_text}* a za posledním znakem je potřeba použít klávesu Enter. Druhým způsobem je označení textu a použití klávesové zkratky Ctrl + Alt + L.

Heading 1

Neboli nadpis první úrovně. Má menší styl písma než titulek, taktéž má tučné písmo a černou barvu. Nadpis první úrovně je možné vytvořit pomocí znaku *## {váš_text}* a za posledním znakem je potřeba použít klávesu Enter. Druhým způsobem je označení textu a použití klávesové zkratky Ctrl + Shift + 1.

Heading 2

Neboli nadpis druhé úrovně. Má menší styl písma než nadpis první úrovně, taktéž má tučné písmo a černou barvu. Nadpis druhé úrovně je možné vytvořit pomocí znaku *### {váš_text}* a za posledním znakem je potřeba použít klávesu Enter. Druhým způsobem je označení textu a použití klávesové zkratky Ctrl + Shift + 2.

Heading 3

Neboli nadpis třetí úrovně. Má menší styl písma než nadpis druhé úrovně, taktéž má tučné písmo a černou barvu. Nadpis třetí úrovně je možné vytvořit pomocí znaku #### *{váš_text}* a za posledním znakem je potřeba použít klávesu Enter. Druhým způsobem je označení textu a použití klávesové zkratky Ctrl + Shift + 3.

Section break

Section break neboli sekce sloužící k rozdělení programu. Tyto sekce umožňují kód rozdělit na několik částí, které uživatel může následně spouštět bez nutnosti spouštět celý program a tím dát možnost uživateli procházet si jednotlivé části programu. Při tvorbě nové sekce máte možnost si vybrat, zda chcete sekci s nadpisem nebo bez. Pokud budete chtít vytvořit sekci s nadpisem můžete tak učinit pomocí znaku *%% {váš_text}* a za posledním znakem je potřeba použít Enter. Zde je nutné mít mezi znakem procento a vašim textem mezeru. Pokud budete chtít novou sekci bez nadpisu, tak pouze napíšete %% a potvrdíte klávesou Enter. Druhým způsob je, že označíte text a dáte klávesovou zkratku Ctrl + Alt + Enter. V případě, že budete chtít sekci bez nadpisu stačí pouze zmáčknout Ctrl + Alt + Enter v místě kde chcete novou sekci.

List

Pokud budete chtít vytvořit odrážky, tak máte možnost je vytvořit pomocí znaku *. Jakmile napíšete znak hvězdičky, tak je potřeba použít Mezerník pro vytvoření odrážky. Odrážku je také možné vytvořit pomocí klávesové zkratky Ctrl + Alt + U.

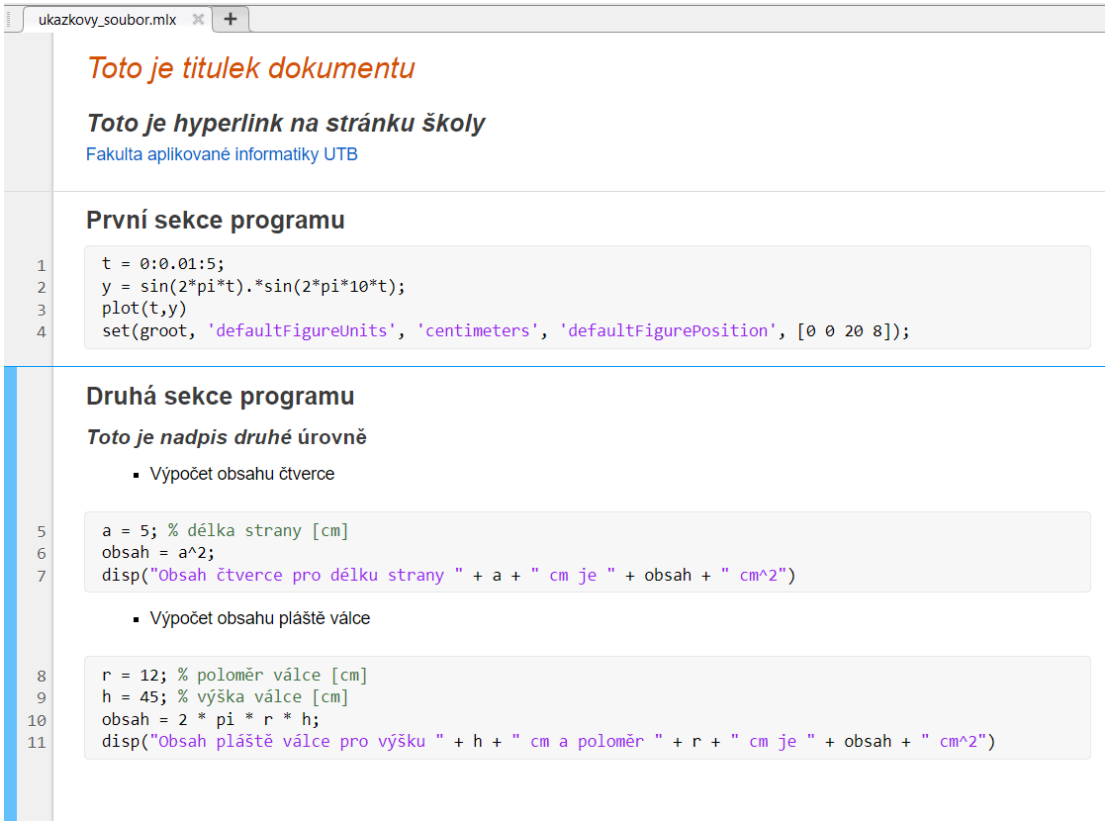
Hyperlink

Do svého dokumentu můžete vkládat i odkazy. Můžete vložit do dokumentu buď pouze odkaz na webovou stránku nebo pomocí zkratky CTRL + K otevřít tabulku pro vytvoření *hyperlinku*. Zde si můžete navíc nastavit pod jakým slovním spojením bude odkaz ukrytý.

Code

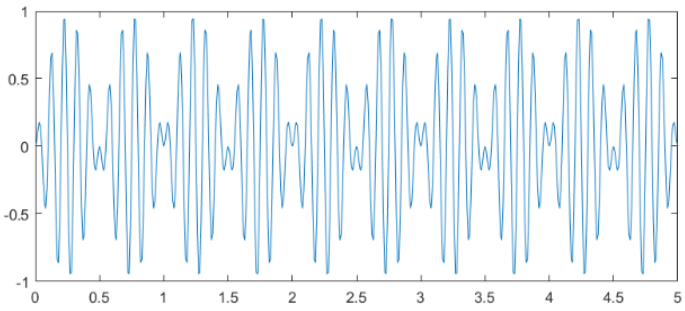
V moment, kdy chcete začít psát programový kód musíte zmáčknout klávesu Alt + Enter. Pokud, se budete chtít přepnout zpět do psaní formátovaného textu, tak opět použijete zkratku Alt + Enter.

Jednotlivé představené možnosti pro tvorbu formálního textu v prostředí *Live editor* jsou předvedeny na Obrázku 3.



Obrázek 3 - Ukázkový program vytvořený v Live scriptu

Výstup z programu, který byl ukázán na předešlé straně je na Obrázku 4.




```
Obsah čtverce pro délku strany 5 cm je 25 cm^2

Obsah pláště válce pro výšku 45 cm a poloměr 12 cm je 3392.9201 cm^2
```

Obrázek 4 - Výstup z programu na Obrázku 3

1.5 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní (GUI) je známo taktéž pod slovem aplikace. Grafické uživatelské rozhraní umožnuje tzv. „point-and-click“ kontrolu nad vašim softwarem. Aplikace vám také umožní ovládat daný software bez nutnosti znát programovací jazyk, ve kterém byl software napsán. [15]

V prostředí *MATLAB* se dají aplikace tvořit několika způsoby. Některé způsoby si nyní uvedeme.

1. Pomocí interaktivního přístupu s využitím funkce *GUIDE*, kde uživatel umisťuje grafické objekty do vykreslovacího okna a nastavuje jejich vlastnosti. Následně propojí tento grafický návrh GUI s požadovaným programovým kódem.
2. Modernějším způsobem je tvorba grafického uživatelské rozhraní pomocí tzv. *App designeru.* Ten je nástupcem předešlého průvodce *GUIDE*. *App designer* nabízí mnohem více možnosti při tvorbě grafického uživatelského rozhraní. Pracuje na

stejném principu jako *GUIDE*, kde si uživatel do kreslícího okna přetahuje různé objekty, u kterých může nastavovat jejich vlastnosti a dále programovat jejich chování.

3. Podstatně náročnější a málo používanou metodou je způsob tvorby uživatelského grafického rozhraní, kdy se celé chování aplikace včetně vykreslovacího okna naprogramuje v podobě scriptu. V tomto případě je nutnost znát daleko lépe programovací jazyk *MATLABu* a související funkce pro tvorbu GUI.

Zmíněný způsob *GUIDE* je první interaktivní metodou pro tvorbu grafického uživatelského prostředí, která byla přidána v roce 1996. Tento způsob je ale již zastaralý, není tak důvod k jeho dalšímu použití. Firma MathWorks plánuje v budoucnu tuto službu již nepodporovat. Uživatelé sice budou moci aplikaci pomocí *GUIDE* otevřít, ale nebudou ji moci upravovat. [8]

App designer představuje nejmodernější a nejrychlejší způsob tvorby uživatelského grafického rozhraní v *MATLABu*. Do prostředí *MATLABu* byl tento způsob přidán v roce 2016. [9]

Jelikož způsob *GUIDE* již není aktuální a způsob vytváření grafického uživatelského rozhraní v podobě scriptu pomocí programovacího jazyka je velmi náročný a neefektivní, tak se budu v práci věnovat pouze *App designeru*, který je i v praktické části použit.

1.5.1 App designer

App designer je interaktivní nástroj, který pomocí tzv. metody „drag and drop“ volně přeloženo jako „přetáhni a pusť“ umožní uživateli volně sestavovat aplikaci podle svojí představivosti. Na výběr má uživatel velkou spoustu komponent, které by se mohli do takové aplikace hodit. Komponenty si vybírá z knihovny, kterou App designer nabízí. Ovšem toto není pouze jediný úkol App designeru. App designer umožnuje programovat samostatnou funkci dané aplikaci ve svém editoru. Uživatel může naprogramovat chování každé komponenty. [15]

Aby bylo jasnější, proč by uživatel měl využít programování v App designeru uvedu následující příklad. Uživatel bude mít v plánu aplikaci, která po zadání hodnot spočítá čistou mzdu. Uživatel si do vykreslovacího okna přetáhne objekty, které potřebuje. Zde se velmi pravděpodobně bude jednat o objekty typu vstupního pole, kde uživatel zadá vstupní hodnoty a výstupní pole pro výstupní hodnotu (spočítanou čistou mzdu). Nyní je řada na programovací části, kde uživatel v kódu pomocí definovaných funkcí získá hodnoty ze vstupních polí a musí čistou mzdu spočítat na základě nějakého vzorce. Jakmile bude mít konečnou hodnotu spočítanou, tak tuto hodnotu předá výstupnímu poli, které výsledek zobrazí v uživatelském grafickém rozhraní.

App designer soubory (aplikace) je možné na první pohled rozeznat podle přípony, ta je v případě *App designer* souboru typu .mlapp.

Pro vytvoření projektu pro tvorbu samotné aplikace za pomocí *App designeru* má uživatel dvě možnosti:

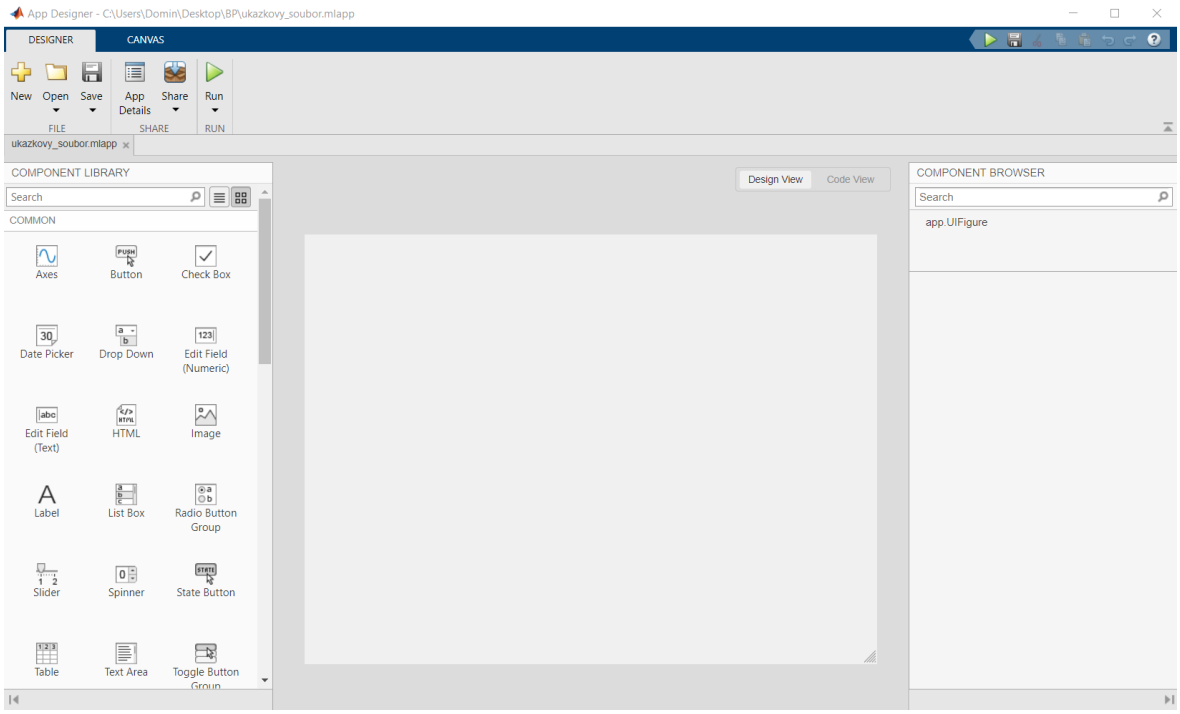
1. Založení projektu pomocí horního menu v základním dialogovém okně *New – App*
2. Méně používaným způsobem je zavolání klíčového slova *appdesigner* v *Command window*.

Po vytvoření nového projektu se otevře dialogové okno, které dává uživateli možnost otevřít již vytvořený projekt v *App designeru*, otevřít si některý z mnoha volně dostupných příkladů provedení aplikace nebo vytvořit novou aplikaci.

Pokud uživatel zvolí novou aplikaci, tak má na výběr ze tří možností:

1. Blank App – naprosto čistá aplikace bez žádného přednastavení.
2. 2-Panel App with Auto-Reflow – aplikace, která je rozdělena na dva panely (části). Aplikace vytvořená v tomto nastavení je responzitivní, tedy mění svou velikost v závislosti na velikosti obrazovky.
3. 3-Panel App with Auto-Reflow – aplikace, která je rozdělena na tři panely (části).

Po vytvoření prázdné *App designer* aplikace se nám otevře dialogové okno, které je ukázáno na Obrázku 5.



Obrázek 5 - Prostředí App designeru po založení prázdného projektu

Pro první spuštění *App designeru* by měl uživatel znát několik základních věci:

1. V levé části se nachází sekce *COMPONENT_LIBRARY*, tato sekce obsahuje základní objekty, které může uživatel použít v grafickém uživatelském rozhraní.
2. V prostřední částí dialogového okna je tzv. vykreslovací okno (pracovní plocha). Do tohoto prostoru uživatel přetahuje objekty z knihovny. Velikost vykreslovacího okna si může upravit např. pomocí roztáhnutí okrajů vykreslovacího okna.
3. V prostřední části dialogového okna v pravém horním rohu je možnost přepínání mezi módy *Design View* a *Code View*. *Design View* je základní dialogové okno, kde uživatel interaktivně vkládá objekty do vykreslovacího pole. *Code view* je dialogové okno, kde uživatel programuje chování jednotlivých komponent nebo celé aplikace.
4. V pravé části je panel s názvem *COMPONENT BROWSER*. V tomto okně má uživatel přehled všech objektů, které do vykreslovacího okna přidal.

   Po založení projektu zde vidíme objekt s názvem *app.UIFigure*. To je název naší vykreslovací plochy, která je App designerem přidána automaticky. Zde v tomto panelu budou přibývat nové položky, jakmile budete přesouvat nové objekty do

vykreslovacího okna. Každá položka v COMPONENT BROWSERU musí mít svůj unikátní název.

5. Spuštění celé aplikace probíhá za pomocí tlačítka *RUN*, které je v horní části menu.

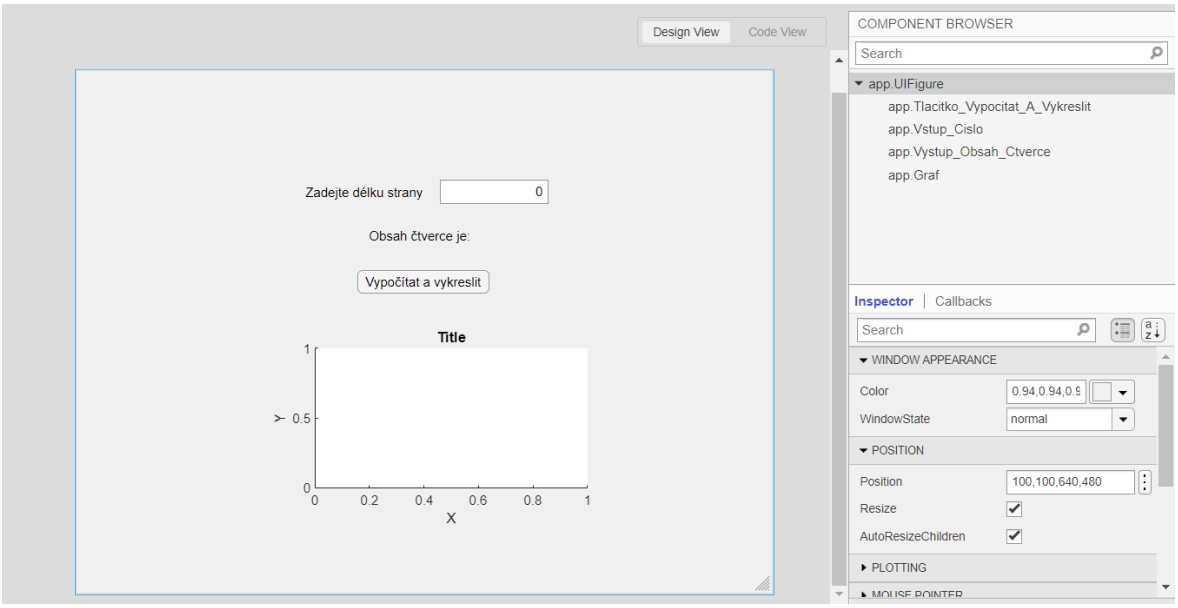
1.5.2 Ukázka postupu při tvorbě jednoduché aplikace v App designeru

Není možné projít veškeré možnosti *App designeru* a ani to není cílem této práce. Proto vás zde provedu kompletním vytvořením jednoduché aplikace. Z následujících obrázků a popisů by mělo být jasné, jak začít budovat jednoduché aplikace.

Cílem této aplikace bude vytvořit spustitelnou aplikaci ve které uživatel zadá délku strany čtverce. Z tohoto údaje se vypočítá obsah čtverce a výsledek zobrazí zpět do grafického uživatelského rozhraní. Z údaje délky strany se do grafu vykreslí čtverec, kde délka osy X a Y bude odpovídat délce strany.

Projekt pro novou aplikaci máme již vytvořený z předešlé ukázky. Nyní je potřeba si uvědomit co budeme potřebovat za komponenty z knihovny *App designeru*. Uživatel bude muset zadat délku strany, proto prvním objektem volíme *Edit Field (Numeric)* což je vstupní pole, které dovolí uživateli zadat pouze čísla. Aby výpočet a vykreslení proběhl až v moment kdy uživatel stiskne tlačítko, tak musíme přidat objekt *Button*. Pro vypsání výsledku výpočtu použijeme textový objekt *Label*. Poslední objektem v naší aplikaci bude graf, který je označován v *App designeru* jako *Axes*.

Nyní jsme si všechny objekty přetáhly do vykreslovacího okna. V sekci COMPONENT BROWSER nyní vidíme celkem 5 objektů z čehož 4 objekty patří pod již dříve zmiňovaný app.*UIFigure.* Pro větší přehlednost pro následující práce si tyto 4 objekty přejmenujeme tak, abychom měli hned jasno z čím pracujeme. Výsledek předešlého slovního popisu je patrný na Obrázku 6.



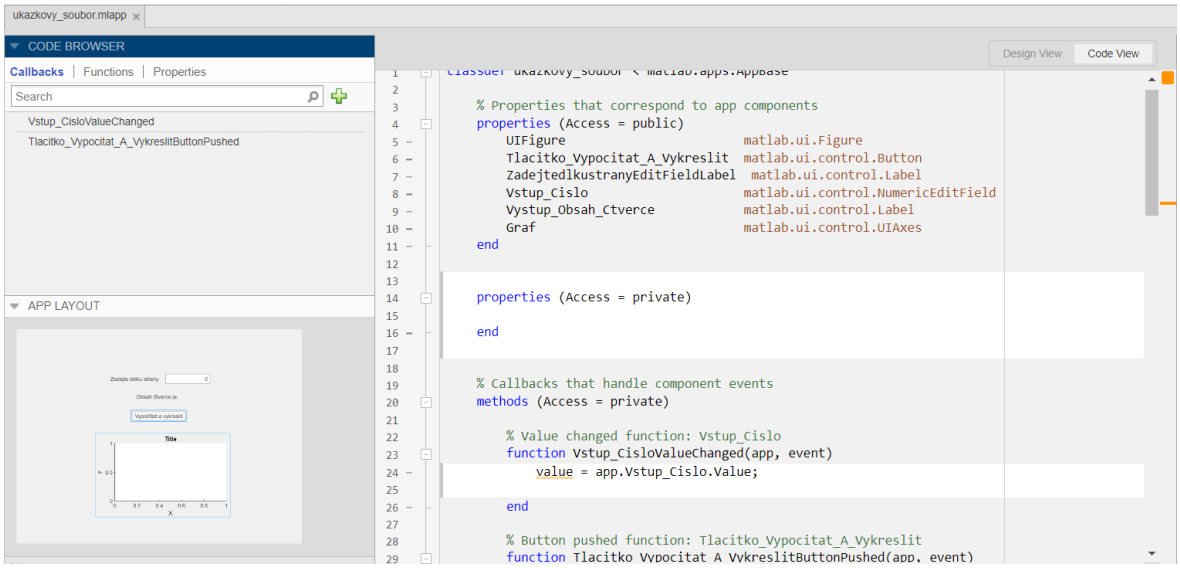
Obrázek 6 - Ukázkové rozvržení aplikace

Nyní když máme objekty připravené je potřeba si vysvětlit tzv. *Callbacks* funkce. Callbacks funkce pracují v momentě, kdy uživatel nějakým způsobem interaguje s daným objektem. Callbacks funkce mají pouze objekty se kterými se dá interagovat. V našem ukázkovém příkladě máme 4 rozdílné objekty. *Callbacks* funkce zde budeme potřebovat u objektu *Edit Field (Numeric)* a *Button*, kde budeme čekat na odezvu ze strany uživatele. U objektu *Axes* a *Label* žádné *Callbacks* funkce potřebovat nebudeme, protože tyto dva objekty nám budou sloužit pouze jako zobrazení našich výsledků.

Nyní když již víme, u jakých objektů chceme *Callbacks* funkce vytvořit, tak je můžeme vytvořit pravým kliknutím na zvolený objekt. Nyní se nám otevře menu, kde zvolíme možnost *Callbacks* a *Add ValueChangeFcn callback*. Po vybrání této možnosti se v programové části vytvoří obslužná funkce pro tento objekt. Přepneme se zpět do *Design View* a stejným postupem vytvoříme Callback funkci pro tlačítko. V tomto případě nebudeme mít na výběr *Add ValueChangeFcn* callback ale *Add ButtonPushedFcn* callback.

Když máme vytvořené obě obslužné funkce objektů, tak se můžeme pustit do implementace programové logiky v záložce *Code View*.

Po přepnutí do záložky *Code View* uvidíme podobné dialogové okno jako na Obrázku 7.



Obrázek 7 - Záložka Code view v našem projektu

Na levé straně je okno s názvem *CODE BROWSER*. V tomto okně máme přehled *Callbacks* funkcí, kde rovněž můžeme volat další různé typy *Callback* funkcí jako např. *Callback* funkce která se zavolá po startu aplikace. V záložce *Functions* máme přehled vytvořených funkcí v programu a rovněž si zde můžeme funkci pomocí tlačítka plus automaticky vytvořit. Poslední záložkou je *Properties*. *Properties* jsou tzv. globální proměnné, které je možné volat v celém programu.

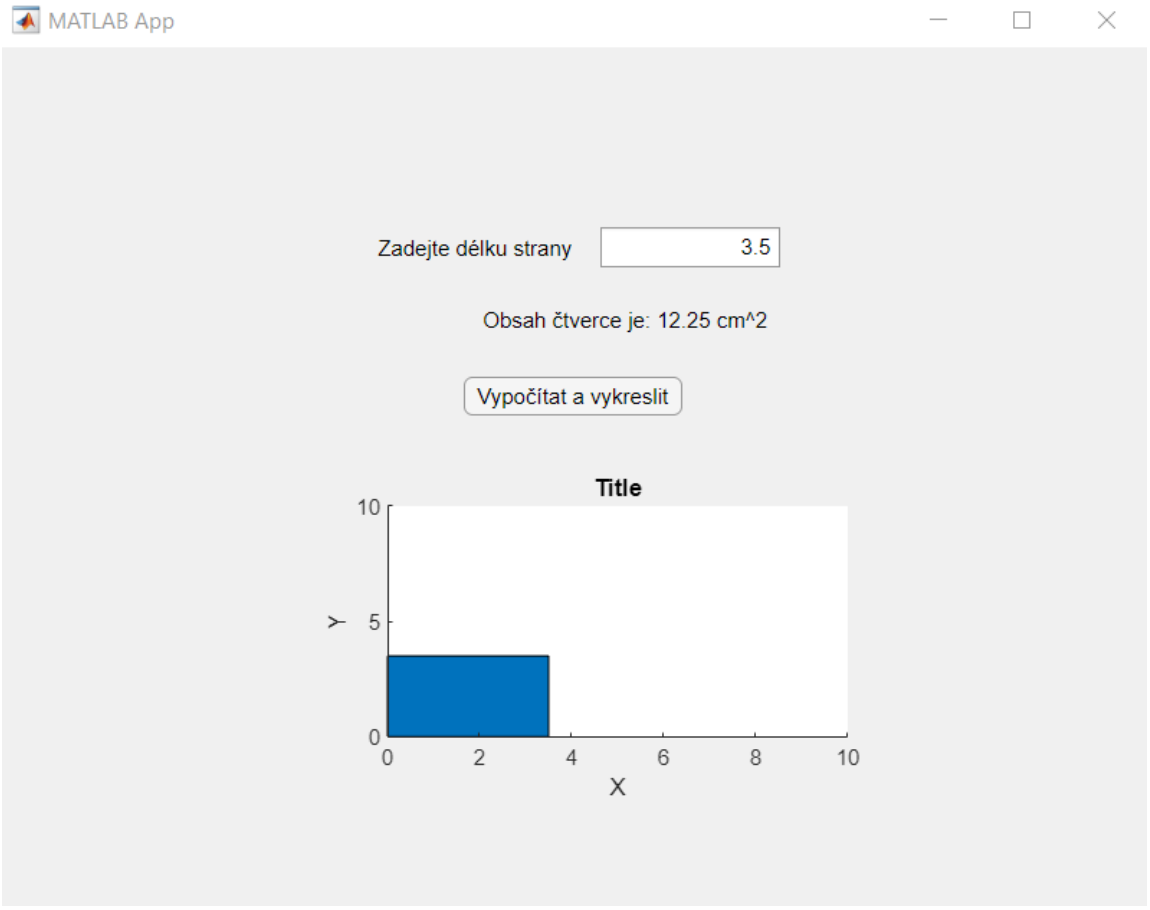
Když nyní víme, jak funkci vytvořit, tak si jednu takovou vytvoříme. Funkce bude počítat obsah čtverce a bude vypisovat výsledek do našeho grafického uživatelského rozhraní.

Stejně tak si vytvoříme globální proměnnou, do které si budeme ukládat číslo udávající délku strany čtverce, kterou zadá uživatel.

Na následující straně je ukázaná implementovaná logika zmiňovaného úkolu. Kód je komentován, proto ho nebude více rozebírat v této práci.

```
properties (Access = private)
    delka_strany = 0; % Globální proměnná
end

methods (Access = private)

    function vypocet_obsahu_ctverce(app)
        obsah = app.delka_strany * app.delka_strany; % Výpočet obsahu čtverce

        xlim(app.Graf, [0 10]); % Nastavení osy X od 0 do 10
        ylim(app.Graf, [0 10]); % Nastavení osy Y od 0 do´10

        app.Vystup_Obsah_Ctverce.Text = "Obsah čtverce je: " + num2str(obsah) +
             "cm^2"; % Uložení nového řetězce do našeho Label objektu

        % Vykreslení čtverce
        area(app.Graf, [0 app.delka_strany], [app.delka_strany app.delka_strany])
    end
end

% Callbacks that handle component events
methods (Access = private)


    % Value changed function: Vstup_Cislo
    function Vstup_CisloValueChanged(app, event)
        app.delka_strany = app.Vstup_Cislo.Value; % Vždy když uživatel změní
          % vstupní hodnotu, tak ji předáme do naší globální proměnné
    end


    % Button pushed function: Tlacitko_Vypocitat_A_Vykreslit
    function Tlacitko_Vypocitat_A_VykreslitButtonPushed(app, event)
        vypocet_obsahu_ctverce(app) % Zavolání naší vytvořené funkce
    end
end
```

Aplikaci nyní spustíme pomocí tlačítka *RUN*. Nastavíme hodnotu délky strany a klikneme na tlačítko vypočítat a vykreslit. Výsledná aplikace včetně výstupu je ukázána na Obrázku 8.



Obrázek 8 - Výsledná aplikace v App designeru

1.6 Simulink

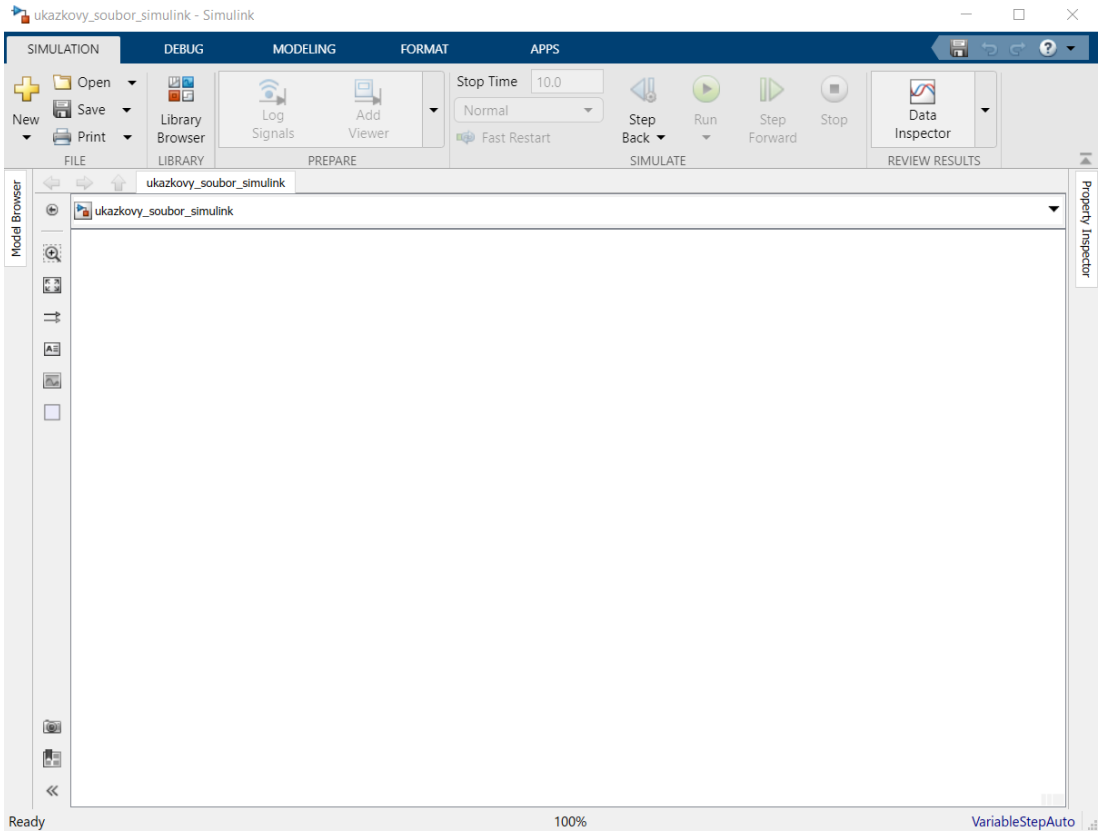
Simulink je grafická simulační nadstavba *MATLABu*. Simulink slouží zejména pro modelování systémů, simulaci a sledování potřebných veličin. Simulink umožnuje rychlé vytváření modelů dynamických systémů ve formě blokových schémat. Bloková schémata jsou sestavena z jednotlivých prvků, které reprezentují určité definované operace. Simulink je hojně využíván v průmyslu, protože umožnuje simulační odzkoušení chování daného systému před jeho skutečným provedením. [16]

Simulink soubory (simulace) je možné na první pohled rozeznat podle přípony, ta je v případě *Simulink* souboru .slx (ve starších verzích to bylo .mdl).

Pro spuštění simulačního nástroje *Simulink* máme dvě možnosti:

1. V základním dialogovém okně *MATLABU* zvolit v horním menu *New – Simulink model*
2. Druhou možností je použít příkaz *simulink* v *Command Window* v základním dialogovém okně.

Po vytvoření nového Simulink souboru se nám otevře dialogové okno, které je ukázáno na Obrázku 9.

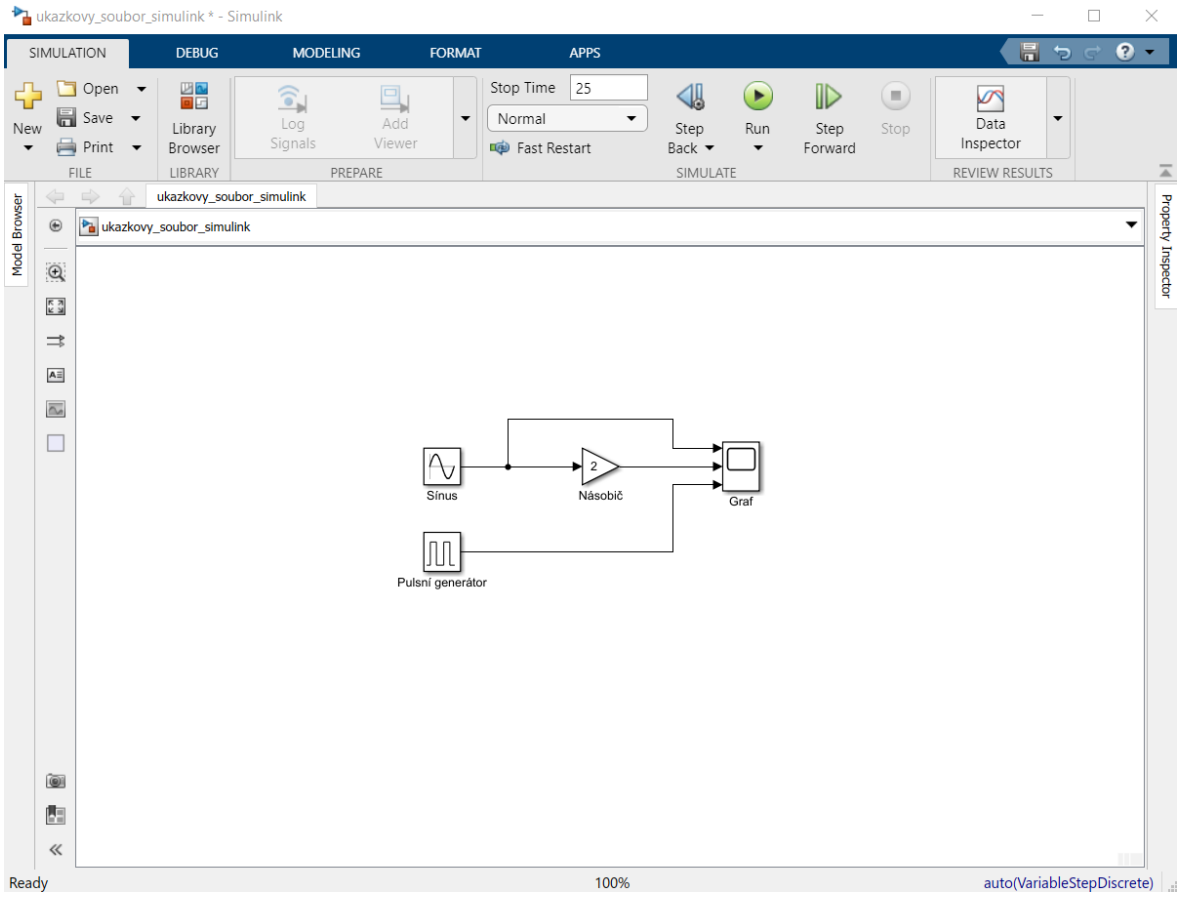


Obrázek 9 - Prostředí Simulinku při vytvoření nového projektu

Pro první spuštění *Simulinku* by měl uživatel znát několik základních věci:

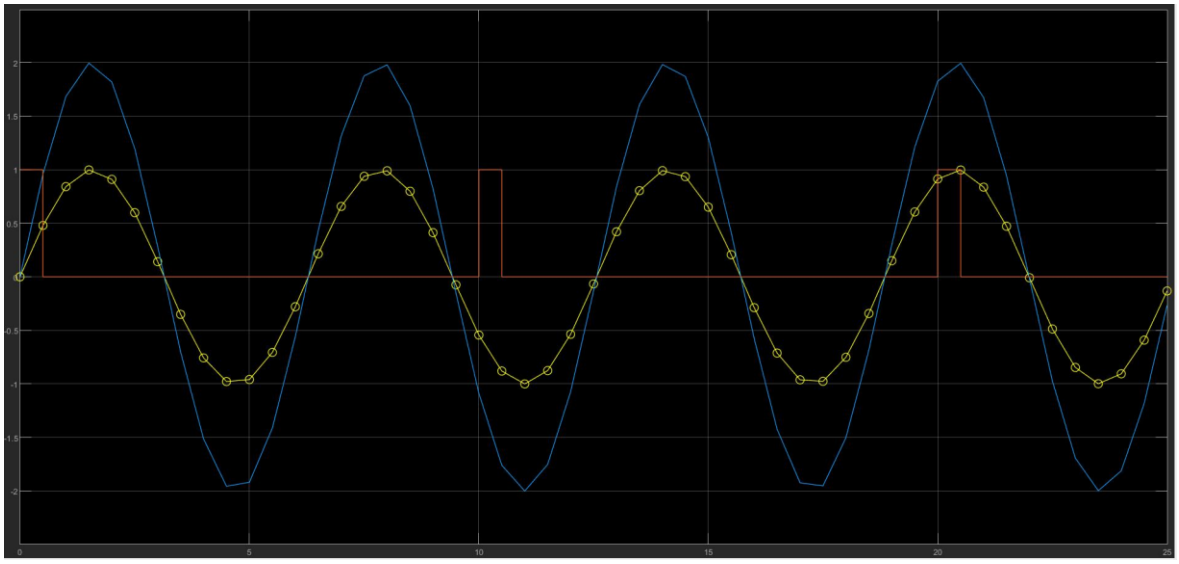
1. V horním menu se nachází záložka s názvem *Library Browser*. Pod touto záložkou se nachází knihovna všech dostupných prvků (bloků), které může uživatel použít při sestavování simulačních schémat.
2. Prvek vložíte do simulačního okna přetáhnutím.
3. Prvky se mezi sebou spojují.
4. Na jednotlivé prvky můžeme kliknout a tím se dostat do nastavení daného prvku.
5. V horním menu se nachází pole pro zadání hodnoty s názvem *Stop Time*. Hodnota v tomto poli určuje, jak dlouho požadujete, aby se systém simuloval.
6. Spuštění simulace provedete stisknutím tlačítka *RUN*, které se nachází v horním menu.

Pro názornou ukázku, bylo vytvořené simulační schéma, které je na Obrázku 10.



Obrázek 10 - Ukázková implementace simulace

Výstup v podobě grafu ze simulačního schématu, které bylo možné vidět na předchozí stránce, je vidět na Obrázku 11. Výstup ve formě tohoto grafu je přístupný v moment kdy uživatel rozklikne prvek *Scope*.



Obrázek 11 - Výstup simulace v podobě grafu

1.6.1 Simscape

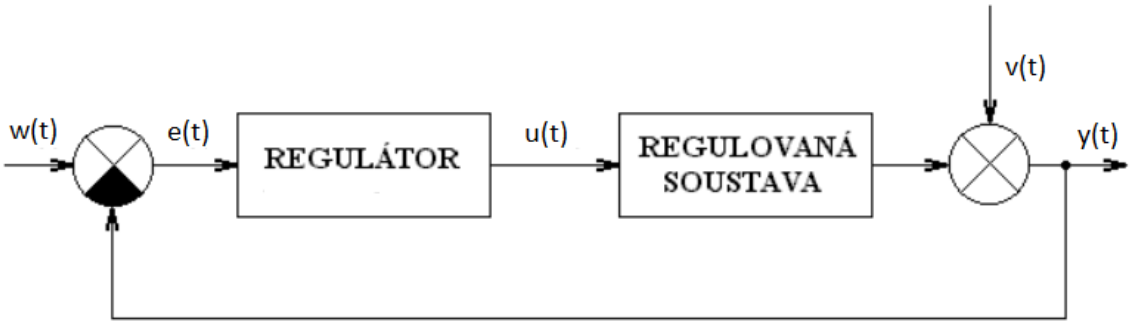
Simscape je další rozšiřující nástroj *Simulinku* pro modelování a simulace. Podobně jako v obyčejném *Simulinku* se zde propojují jednotlivé prvky. *Simscape* zavádí do simulačních obvodů reálné fyzikální veličiny a před-definované sub-systémy. Na rozdíl od *Simulinku* zde není potřeba manuální sestavování příslušných matematických vztahů mezi veličinami, protože *Simscape* si sám tyto hodnoty odvodí. Z knihovny dostupných prvků jsou na výběr prvky mechanické, hydraulické, magnetické a mnoho dalších, ze kterých se pak jednoduše a intuitivně dané systémy postupně skládají. [10]

2 ZÁKLADNÍ TYPY REGULÁTORŮ

Pro demonstraci vlivů základních algoritmů řízení na různé dynamické systémy, byly vybrány následující algoritmy řízení: P, PI, PD, PID a 2-polohový regulátor.

Základní regulační obvod

Ve své bakalářské práci uvažuji základní zpětnovazební obvod, který je znázorněn na Obrázku 12.



Obrázek 12 - Základní zpětnovazební regulační obvod

kde w(t) – žádaná veličina

e(t) – regulační odchylka

u(t) – akční zásah

v(t) – poruchová veličina

y(t) – regulovaná veličina

Regulátor

Regulátorem obecně nazýváme zařízení, které se nachází v regulačním obvodu a kterým se provádí proces regulace. Regulátor nejčastěji pracuje na základě vyhodnocení regulačních odchylky [11]:

$$e(t) = w(t) - y(t), \tag{2.1}$$

kde e(t) – regulační odchylka

w(t) – žádaná veličina

y(t) – regulovaná veličina

Změnou akční veličiny se regulátor snaží regulační odchylku odstranit nebo alespoň minimalizovat.

Regulátory se dají rozdělit podle několika kategorií, například:

Podle přívodu energie [11]:

1. Přímé – tyto regulátory nevyžadují vnější zdroj energie pro svou činnost. Pro ovládání akční veličiny používají vlastní energii.
2. Nepřímé – tyto regulátory vyžadují vnější zdroj energie pro svou činnost.

Podle průběhu výstupní veličiny [11]:

1. Spojité – veličiny jsou spojité v čase.
2. Nespojité – některý z členů regulátoru není spojitý v čase.

Regulovaná soustava

Regulovaná soustava je obecně proces, na kterém se provádí regulace. Výstupní veličinou z regulované soustavy je regulovaná veličina. Do regulované soustavy vstupuje akční zásah z regulátoru, případně poruchové veličiny, pokud na daný systém působí. Všechny veličiny, které vstupují do regulované soustavy mají obecně vliv na časový průběh regulované veličiny. [11]

2.1 P-regulátor

P-regulátor neboli proporcionální regulátor je nejjednodušším spojitým regulátorem. Do regulátoru vstupuje hodnota regulační odchylky, která je vypočítána jako požadovaná hodnota – aktuální hodnota reg. veličiny. Regulační odchylka je potom vynásobená proporcionální konstantou. Pokud hodnota proporcionální složky bude příliš malá, tak regulátor požadované hodnoty nemusí dosáhnout, naopak pokud bude hodnota proporcionální složky příliš vysoká, může způsobit kmitání regulační veličiny. [11]

Rovnice P-regulátoru v časové oblasti:

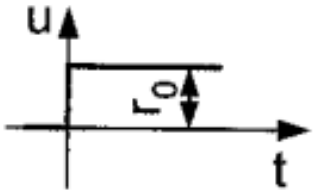
$$u(t) = r_0 e(t), \tag{2.2}$$

kde u(t) – akční zásah

r_0 – proporcionální konstanta regulátoru

e(t) – regulační odchylka

Přechodová charakteristika P-regulátoru je vidět na Obrázku 13.



Obrázek 13 - Přechodová charakteristika P-regulátoru [11]

Přenos regulátoru:

$$G_R(s) = r_0, \tag{2.3}$$

2.2 PI-regulátor

PI-regulátor je nejčastěji tvořen paralelním zapojením P-regulátoru a I-regulátoru. I-složka neboli integrační složka sčítá všechny předchozí regulační odchylky. Pokud bude regulační odchylka kladná, tak se bude integrační složka zvyšovat. V případě, že bude regulační odchylka záporná, tak se bude integrační složka snižovat. Díky integrační složce lze často úplně eliminovat trvalou regulační odchylku (např. v příp. řízení statických soustav na konstantní žádanou hodnotu) [11]

Rovnice PI-regulátoru v časové oblasti:

$$u(t) = r_0 e(t) + r_{-1} \int_0^t e(\tau) d\tau, \tag{2.4}$$

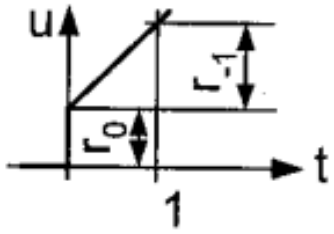
kde u(t) – akční zásah

r_0 – proporcionální konstanta regulátoru

r_{-1} – integrační konstanta regulátoru

e(t) – regulační odchylka

Přechodová charakteristika PI-regulátoru je vidět na Obrázku 14.



Obrázek 14 - Přechodová charakteristika PI-regulátoru [11]

Přenos regulátoru:

$$G_R(s) = r_0 + \frac{r_{-1}}{s}, \tag{2.5}$$

V některých případech regulace se integrační konstanta (r_{-1}) nahrazuje integrační časovou konstantou T_I.

Vztah pro výpočet integrační časové konstanty:

$$T_I = \frac{r_0}{r_{-1}}, \tag{2.6}$$

2.3 PD-regulátor

PD-regulátor je nejčastěji tvořen paralelním zapojením P-regulátoru a D-regulátoru. D-složka neboli derivační složka počítá s rychlostí změny regulační odchylky a vynásobí ji derivační konstantou. Čím rychleji se hodnota regulační odchylky mění tím rychleji narůstá derivační složka. Díky vhodně zvolené derivační složce se obecně zlepší stabilitní vlastnosti regulačního obvodu a můžeme si tak např. dovolit větší zesílení regulátoru. [11] [12]

Rovnice PD-regulátoru v časové oblasti:

$$u(t) = r_0 e(t) + r_1 \frac{de(t)}{dt}, \tag{2.7}$$

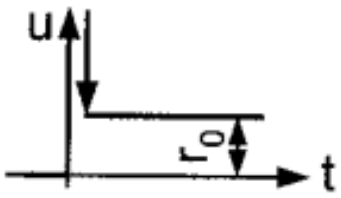
kde u(t) – akční zásah

r_0 – proporcionální konstanta regulátoru

e(t) – regulační odchylka

r_1 – derivační konstanta regulátoru

Přechodová charakteristika PD-regulátoru je vidět na Obrázku 15.



Obrázek 15 - Přechodová charakteristika PD-regulátoru [11]

Přenos regulátoru:

$$G_R(s) = r_0 + r_1 s, \tag{2.8}$$

V některých případech regulace se derivační konstanta (r_1) nahrazuje derivační časovou konstantou T_D.

Vztah pro výpočet derivační časové konstanty:

$$T_D = \frac{r_1}{r_0}, \tag{2.9}$$

Uvedená verze PD-regulátoru představuje tzv. ideální verzi regulátoru. Reálné regulátory jsou doplněné o filtr derivační složky.

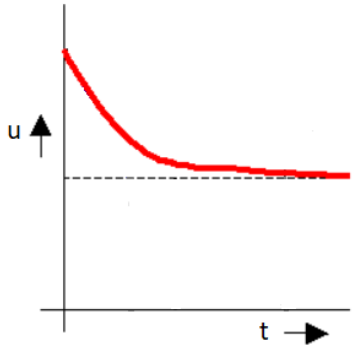
Přenos reálného regulátoru:

$$G_R(s) = \mathrm{r}_0\left(1 + \frac{\mathrm{T_D}s}{\mathrm{NT_D}s+1}\right), \tag{2.10}$$

kde N – filtr derivační složky

T_D – derivační časová konstanta

Přechodová charakteristika reálného PD-regulátoru je vidět na Obrázku 16.



Obrázek 16 - Přechodová charakteristika reálného PD-regulátoru

2.4 PID-regulátor

PID-regulátor je nejčastěji tvořen paralelním zapojení P-regulátoru, I-regulátoru a D-regulátoru. Díky kombinaci všech tří složek se stává PID regulátor, regulátorem s nejlepšími vlastnostmi v porovnání s jednotlivými regulátory. Derivační složka obecně zajistí rychlejší reakci na změnu regulační odchylky a integrační složka bude často zajišťovat nulovou regulační odchylku. [11] [12]

Rovnice PID-regulátoru v časové oblasti:

$$u(t) = r_0 e(t) + r_{-1} \int_0^t e(\tau) d\tau + r_1 \frac{de(t)}{dt}, \tag{2.11}$$

kde u(t) – akční zásah

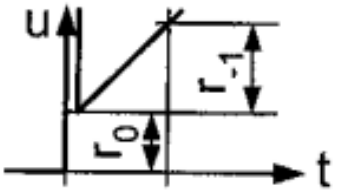
r_0 – proporcionální konstanta regulátoru

e(t) – regulační odchylka

r_{-1} – integrační konstanta regulátoru

r_1 – derivační konstanta regulátoru

Přechodová charakteristika PID-regulátoru je vidět na Obrázku 17.



Obrázek 17 - Přechodová charakteristika PID-regulátoru [11]

Přenos regulátoru:

$$G_R(s) = r_0 + \frac{r_{-1}}{s} + r_1 s, \tag{2.12}$$

V některých případech regulace se integrační konstanta (r_{-1}) nahrazuje integrační časovou konstantou T_I výpočetní vztah byl uveden v rovnici 2.6 a derivační konstanta (r_1) se nahrazuje derivační časovou konstantou T_D výpočetní vztah byl uveden v rovnici 2.9.

Uvedená verze PID-regulátoru představuje tzv. ideální verzi regulátoru. Reálné regulátory jsou doplněné o filtr derivační složky.

Přenos reálného regulátoru:

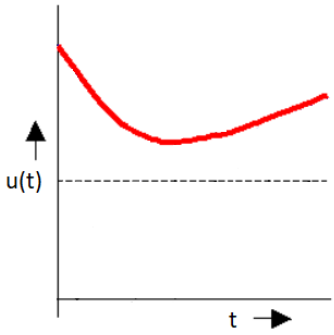
$$G_R(s) = \mathrm{r}_0\left(1 + \frac{1}{\mathrm{T_I}s} + \frac{\mathrm{T_D}s}{\mathrm{NT_D}s+1}\right), \tag{2.13}$$

kde N – filtr derivační složky

T_I – integrační časová konstanta

T_D – derivační časová konstanta

Přechodová charakteristika reálného PID-regulátoru je vidět na Obrázku 18.



Obrázek 18 - Přechodová charakteristika reálného PID-regulátoru

2.5 2-polohový regulátor

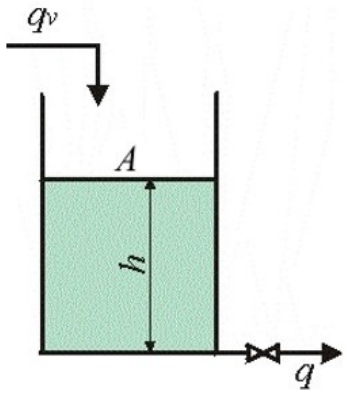
Tento typ regulátorů patří k nespojitým regulátorům. Jeho výstupní signál nepracuje spojitě v závislosti na vstupu regulátoru, ale přepíná svou pozici do mezních poloh např. otevřeno – zavřeno nebo zapnuto – vypnuto. K přepnutí akční veličiny dochází v moment, kdy regulační odchylka překročí přednastavenou horní nebo dolní mez. Regulátor je používán zejména tam, kde není kladen větší důraz na přesnost regulace. [11]

3 VYBRANÉ DYNAMICKÉ SYSTÉMY PRO DEMONSTRACI ALGORITMŮ ŘÍZENÍ

Pro simulaci vlivu uvedených regulátorů jsem si po konzultaci s vedoucím práce vybral celkem tři matematické modely. Každý z těchto modelů byl vybírán tak, aby pracoval na jiném principu. První z modelů, který jsem zvolil je model tekutinový. Jedná se o model otevřeného zásobníku na kapalinu. Druhý z modelů je model tepelný, kde se jedná o elektrické vytápění místnosti. Poslední model je mechanický, kde uvažuji jednoduchý mechanický oscilátor (systém typu „hmotnost-pružina-tlumič“). Každý z těchto uvedených modelů byl popsán analytickým postupem pro získání vnitřního stavového popisu.

3.1 Model otevřeného zásobníku na kapalinu

Model otevřeného zásobníku na kapalinu představuje zásobník, do kterého přitéká kapalina. Ze zásobníku rovněž nějaká část kapaliny odtéká z důvodu umístění ventilu na spodní straně zásobníku. Cílem tohoto modelu je, aby regulátor vhodně nastavil akční zásah pro ovládání přítoku kapaliny do nádrže a tím docílil požadované výšky hladiny.

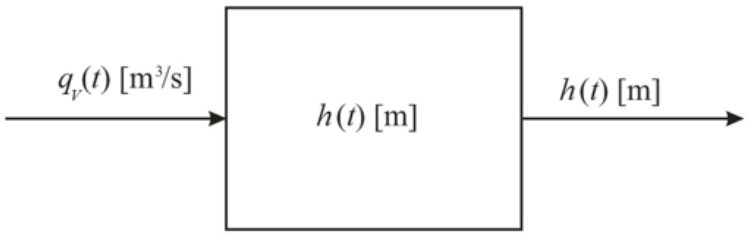


Obrázek 19 - Schématický obrázek otevřeného zásobníku na kapalinu [13]

Popis parametrů ze schématického obrázku:

- q_v – přítok kapaliny [m^3/s]
- A – průřez zásobníku [m^2]
- h – výška hladiny [m]
- q – odtok ze zásobníku [m^3/s]

Z pohledu teorie systémů si můžeme daný proces znázornit dle Obrázku 20, tedy vstupní veličinou bude přítok $q_v(t)$, jedinou vnitřní (stavovou) veličinou bude výška hladiny h(t) v zásobníku, a to bude také výstup z pohledu informací, které nás zajímají dále pro simulaci a řízení.



Obrázek 20 - Systémové schéma otevřeného zásobníku na kapalinu [13]

Bilance modelu:

$$q_v(t) = q(t) + \frac{\mathrm{dV(t)}}{dt}, \tag{3.1}$$

kde q(t) – aktuální výtok kapaliny [m^3/s]

V – objem nádrže [m^3]

Objem nádrže vyjádříme jako průřez zásobníku vynásobený aktuální výškou hladiny.

$$q_v(t) = q(t) + A\frac{\mathrm{dh(t)}}{dt}, \tag{3.2}$$

Abychom si vyjádřili výtok kapaliny z otevřené nádrže, tak použijeme Torricelliho rovnici. [13]

$$q(t) = k\sqrt{\mathrm{h(t)}} \tag{3.3}$$

Při sestavení kompletní rovnice dostáváme:

$$q_v(t) = k\sqrt{\mathrm{h(t)}} + A\frac{\mathrm{dh(t)}}{dt} \tag{3.4}$$

Po přesunutí derivace na levou stranu dostáváme obecně nelineární systém 1. řádu.

$$\frac{\mathrm{dh(t)}}{dt} = \frac{1}{A}\left[q_v(t) - k\sqrt{\mathrm{h(t)}}\right]; \quad h(0) = h^S \tag{3.5}$$

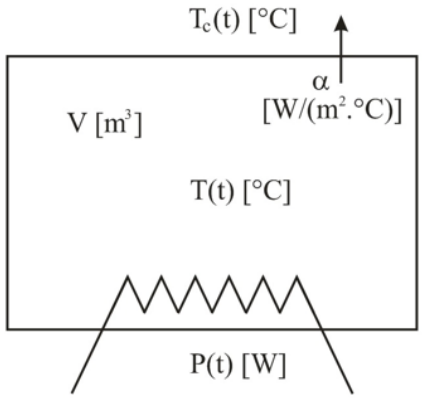
kde $h(0) = h^S$ – počáteční výška hladiny v nádrži

Pro realizaci modelu uvažuji několik zjednodušujících předpokladů [13]:

- Jedná se o otevřený zásobník na kapalinu
- Zásobník má dokonale válcový tvar (konstantní průřez)
- V okolí je konstantní atmosférický tlak

3.2 Model elektrického vytápění místnosti

Model elektrického vytápění místnosti představuje uzavřená místnost, ve které je elektrické vytápění a teplota v místnosti je dále ovlivňována venkovní teplotou. Cílem tohoto modelu je, aby regulátor vhodně nastavil akční zásah pro ovládání výkonu topení v místnosti a tím docílil požadované teploty v místnosti.

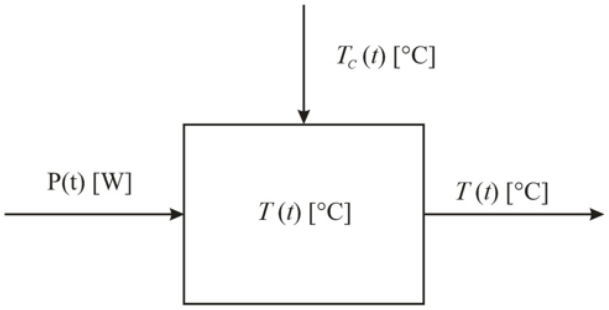


Obrázek 21 - Schématický obrázek elektrického vytápění místnosti [13]

Popis parametrů ze schématického obrázku:

- V – objem místnosti [m^3]
- T(t) – aktuální teplota v místnosti [W]
- α – průměrný koeficient přestupu tepla (stěny + okna + dveře) [W/(m^2.°C)]
- P(t) – aktuální výkon topení [W]
- $T_C(t)$ – aktuální venkovní teplota [°C]

Z pohledu teorie systémů si můžeme daný proces znázornit dle Obrázku 22, tedy vstupní veličinou bude výkon topení P(t), druhá vstupní veličina je poruchová, a to venkovní teplota $T_C(t)$. Jedinou vnitřní (stavovou) veličinou bude teplota v místnosti T(t) a to bude také výstup z pohledu informací, které nás zajímají dále pro simulaci a řízení.



Obrázek 22 - Systémové schéma elektrického vytápění místnosti [13]

Bilance modelu:

$$P(t) = \alpha A[T(t) - T_c(t)] + \frac{d}{dt}[V\rho c_P T(t)] \tag{3.6}$$

kde A – přestupná plocha [m^2]

ρ – hustota vzduchu [kg/m^3]

c_P – měrná tepelná kapacita vzduchu [J/(kg.°C)]

Akumulaci tepla (dle zavedených předpokladů na straně 49) upravíme na tvar:

$$P(t) = \alpha A[T(t) - T_c(t)] + V\rho c_P \frac{dT(t)}{dt} \tag{3.7}$$

Po přesunutí derivace na levou stranu dostáváme obecně lineární systém 1. řádu.

$$\frac{dT(t)}{dt} = \frac{1}{V\rho c_P} P(t) - \frac{\alpha A}{V\rho c_P}[T(t) - T_c(t)];\ \ T(0) = T^S \tag{3.8}$$

kde $T(0) = T^S$ – počáteční teplota v místnosti

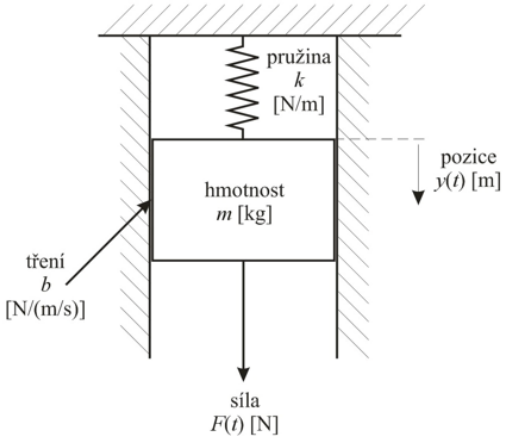
Pro realizaci modelu uvažuji několik zjednodušujících předpokladů [13]:

- Dokonalé promíchání vzduchu
- Konstantní parametry (objem místnosti, hustota vzduchu, měrná tepelná kapacita vzduchu, průměrný koeficient přestupu tepla, přestupná plocha…)
- Zanedbáváme akumulaci tepla ve stěnách, teplo generované přítomnými osobami
- Po dobu simulace je venkovní teplota konstantní

3.3 Model mechanického oscilátoru

Model mechanického oscilátoru představuje těleso o určité hmotnosti fixované pomocí pružiny, které vykonává pohyb v jednom směru na základě působící síly. [14]

Tento model může zjednodušeně představovat např. automobilový tlumič. Cílem pak je, aby regulátor vhodně nastavil akční zásah, tj. působící sílu tak, aby hmotnostní element (např. karoserie auta) zůstal v požadované pozici.

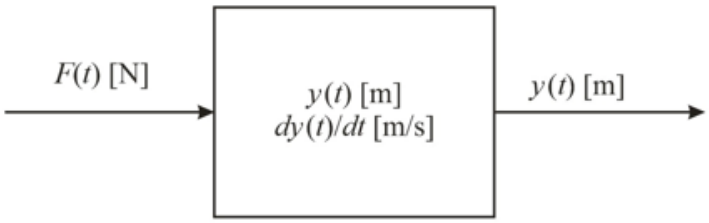


Obrázek 23 - Schématický obrázek mechanického oscilátoru [13]

Popis parametrů ze schématického obrázku:

- k – konstanta tuhosti pružiny [N/m]
- b – koeficient tření [N/(m/s)]
- m – hmotnost [kg]
- y(t) – aktuální pozice [m]
- F(t) – aktuální působící síla [N]

Z pohledu teorie systémů si můžeme daný proces znázornit dle Obrázku 24, tedy vstupní veličinou bude působící síla F(t). Vnitřní (stavovou) veličinou bude pozice y(t) a rychlost v(t). Pozice y(t) je výstup z pohledu informací, které nás zajímají dále pro simulaci a řízení.



Obrázek 24 - Systémové schéma mechanického oscilátoru [13]

Na základě rovnováhy sil a 2. Newtonova zákona lze obecně psát:

$$\sum_i F_i = m\frac{\mathrm{d}^2 y(t)}{dt^2} \tag{3.9}$$

$$F(t) - F_k(t) - F_b(t) = m\frac{\mathrm{d}^2 y(t)}{dt^2} \tag{3.10}$$

kde $F_k(t)$ – síla pružiny [N]

$F_b(t)$ – třecí síla [N]

Po úpravě pak dostáváme lineární diferenciální rovnici 2. řádu. [14]

$$F(t) = m\frac{\mathrm{d}^2 y(t)}{dt^2} + b\frac{dy(t)}{dt} + \mathrm{k}y(t); \quad y(0) = y^S \tag{3.11}$$

kde $y(0) = y^S$ – počáteční pozice hmotnostního elementu

Pro realizaci modelu uvažuji několik zjednodušujících předpokladů [13]:

- Jedná se o ideální pružinu
- Dobře mazaná, kluzná plocha (viskózní tření)
- Konstantní parametry (konstanta tuhosti pružiny, koeficient tření, hmotnost…)

II. PRAKTICKÁ ČÁST

4 PROGRAMOVÁ REALIZACE

Praktickou část bakalářské práce lze rozdělit na dvě části. A to na část, která řeší simulaci vybraných systémů + jejich řízení, a část která řeší tvorbu grafického uživatelského rozhraní včetně propojení se simulacemi.

V této kapitole se budeme věnovat realizaci první části, a to tedy tvorbě simulací, pro které byla použita simulační nástavba MATLABu *Simulink*.

Ve své práci provádím simulaci celkem na třech modelech, které byly popsány v kapitole 3.

Na každém z modelů je možné provést regulaci pomocí *PID-regulátoru* nebo *2-spolohového regulátoru*. Z tohoto důvodu byly simulační soubory rozděleny vždy na dva pro každý model. Jeden soubor reprezentuje simulaci řízení s využitím *PID-regulátoru* a druhý soubor s využitím *2-polohového regulátoru*. Celkem tedy bakalářská práce obsahuje 6 simulačních souborů.

Jednotlivé dynamické systémy jsou modelovány s využitím tzv. subsystému. Subsystém mi umožní rozdělit celý simulační obvod do určitých logických částí. V subsystému vždy řeším pouze realizaci konkrétního dynamického systému. Samotnou regulaci řeším v hlavní části simulačního souboru včetně vykreslování průběhů veličin.

Pro simulaci všech dynamických systémů jsem si zvolil, že chci zaznamenat 225 simulovaných hodnot. Tuto hranici považuji za dostačující jak pro vykreslování zvolených veličin, tak pro účely animací (při zvolení vhodné doby simulace). Tato hranice byla nastavena převážně z důvodu zabránění případného přetížení aplikace (při zvolené delší době simulace a malém simulačním/integračním kroku).

Dobu simulace neboli *Stop Time* si volí uživatel skrze vytvořené grafické uživatelské rozhraní (GUI).

4.1 Používané bloky pro simulaci

V teoretické části bylo vysvětleno, že simulační schémata jsou tvořená prvky (bloky), které reprezentují určité funkce, výrazy nebo operace.

V této podkapitole bych chtěl shrnout všechny použité bloky, které jsem ve své práci použil pro implementaci diferenciálních rovnic popisujících systémy včetně vysvětlení, proč zvolený blok používám. V následujících kapitolách bude představena konkrétní realizace jednotlivých simulačních schémat.

Konkrétní hodnoty jednotlivých prvků si uživatel volí skrze grafické uživatelské rozhraní (GUI).

Typy bloků

Constant

Blok constant používám k vložení konstatní hodnoty do simulačního obvodu.

Sum

Tento blok používám na sčítání nebo odečítání vstupních hodnot. V nastavení tohoto bloku jsem provedl úpravu v parametru *List of signs*, do kterého jsem zapsal výraz podle toho, jestli chci provést sčítání nebo odečítání vstupů. Pokud jsem potřeboval vstupy sčítat zapsal jsem výraz $|++$. Pro odečítání jsem použil výraz $|+-$.

PID Controller

Blok, který používám pro nastavení *PID-regulátoru*. V nastavení parametrů tohoto bloku je podstatný parametr *Controller*. Tento parametr mi říká, jaký typ *PID-regulátoru* budu používat. Zde tedy používám nastavení P, PI, PD a PID. V části *Controller parameters* nastavuji konkrétní hodnoty regulátoru pro konkrétní regulátor. V záložce *Output Saturation* jsem si určil, že chci výstup z regulátoru (akční zásah) omezit. V parametru *Anti-windup Method* jsem si zvolil metodu clamping. Jedná se obecně o metodu, která potlačuje hodnotu výstupu z regulátoru (akční zásah).

Scope

Blok, který používám pro vykreslení průběhů v závislosti na čase. V každém simulačním souboru jsou celkem tři tyto bloky. První blok slouží pro vykreslení regulované veličiny a požadované veličiny. V tomto případě bylo potřeba přidat druhý vstup do tohoto bloku. Druhý *scope* blok slouží pro vykreslení akčního zásahu. A třetí blok pro vykreslení regulační odchylky. V nastavení tohoto bloku jsem provedl změnu v záložce *Main* u parametru *Sample time*. Parametr mi udává tzv. vzorkovací frekvenci neboli hodnotu času po které se provede záznam hodnoty a její vykreslení.

Relay

Blok, který používám pro realizaci funkce *2-polohového* regulátoru. V parametrech tohoto bloku nastavuji hodnotu *Switch on point* a *Switch off point*. Jedná se o hodnoty mezí, které při překročení regulační odchylky nastaví výstup regulátoru na 0 nebo 1.

Gain

Blok zesílení, který používám pro vynásobení vstupní hodnoty konstantou. V parametrech tohoto bloku nastavuji parametr *Gain*, který mi udává hodnotu čísla, kterým se bude vstup násobit.

Integrator Limited

Blok, který používá pro numerické řešení daných diferenciálních rovnic s respektováním definovaných fyzikálních omezení sledovaných veličin. Blok v případě překročení nastavených hodnot vypne integrační činnost a tím pádem se nebude sledovaná veličina dále navyšovat či snižovat. V parametrech tohoto bloku nastavuji parametr *Initial condition* což představuje počáteční hodnotu pro definované stavové veličiny. Dále parametr *Upper saturation* limit a *Lower saturation limit*, které představují maximální a minimální hodnotu sledované veličiny systému.

Sqrt

Blok, který používám pro odmocnění vstupu.

Integrator

Blok používám pro integrační činnost bez omezení.

Product

Blok používám pro vynásobení vstupních hodnot mezi sebou.

Divide

Blok používám pro podělení vstupních hodnot.

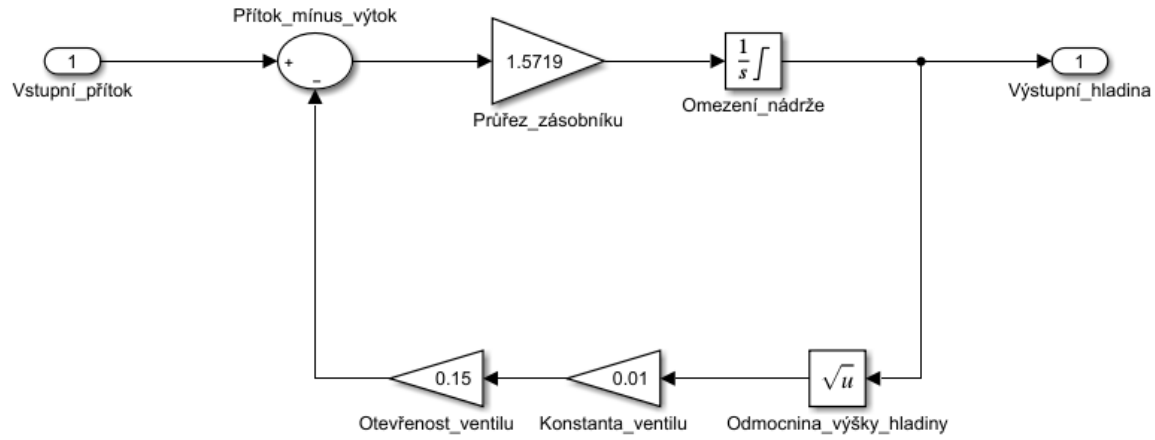
4.2 Model otevřeného zásobníku na kapalinu

Následující podkapitola se věnuje realizaci konkrétního modelu, a to modelu otevřeného zásobníku na kapalinu. Bude představeno zapojení jednotlivých bloků pro realizaci simulace. Na závěr této podkapitoly bude zmíněna tabulka popisující použité bloky pro realizaci simulace včetně hodnot, které se v tomto bloku nastavují.

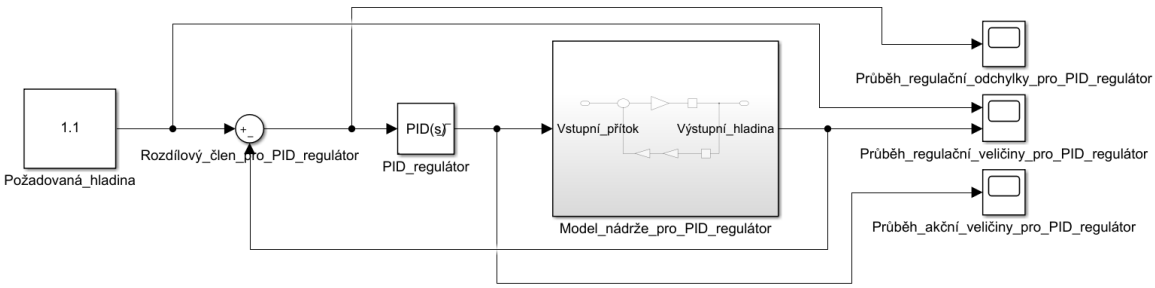
Diferenciální rovnice 3.5, která je dále zmiňována u simulačních schémat byla doplněna o možnost nastavit míru otevření výstupního ventilu.

Realizace simulačních schémat pro PID-regulátor je na Obrázku 25 a 26. Tento simulační obvod se nachází v programu:

BP_Pavlica_Simulink_PID_model_otevreneho_zasobniku.slx



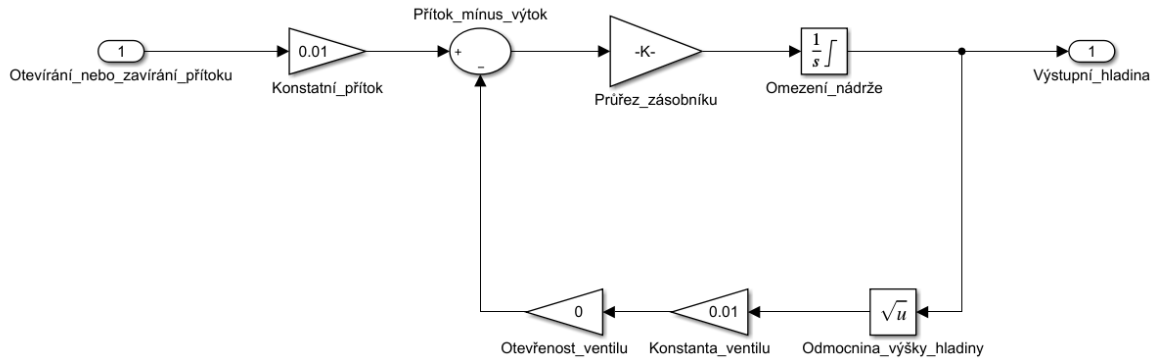
Obrázek 25 – Realizace zásobníku na kapalinu podle diferenciální rovnice 3.5 pro PID-regulátor



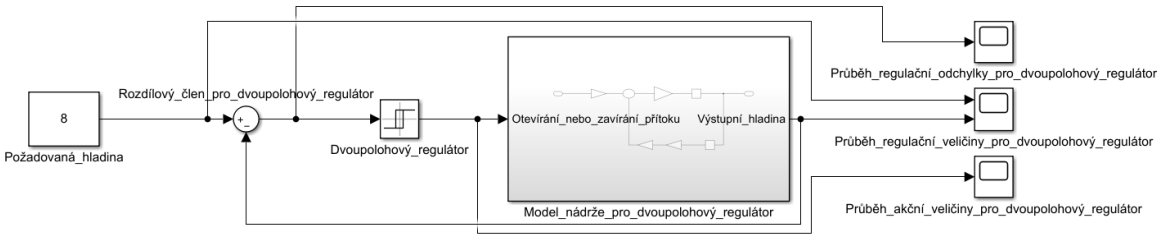
Obrázek 26 - Realizace celého reg. obvodu pro zásobník na kapalinu a PID-regulátor

Realizace simulačních schémat pro 2-polohový regulátor je na Obrázku 27 a 28. Oproti PID-regulaci má 2-polohová regulace doplněn blok definující konstantní přítok, ten lze nastavit pomocí GUI. Tento simulační obvod se nachází v programu:

BP_Pavlica_Simulink_2Polohovy_model_otevreneho_zasobniku.slx



Obrázek 27 - Realizace zásobníku na kapalinu podle diferenciální rovnice 3.5 pro 2-polohový regulátor



Obrázek 28 - Realizace celého reg. obvodu pro zásobník na kapalinu a 2-polohový regulátor

Pro přehled použitých bloků a nastavení jejich parametrů pro danou simulaci je vytvořená následující tabulka.

V sloupci *nastavené parametry bloku* se vyskytují pouze parametry, které byly při vložení bloku změněny.

Pokud se ve sloupci *nastavené parametry bloku* a v příslušném řádku nenachází číselná hodnota, ale slovní spojení, tak se jedná o hodnotu, která je bloku předávána z grafického uživatelského rozhraní (GUI) pod tímto názvem.

Název bloku	**Typ bloku**	**Nastavené parametry bloku**
Požadovaná_hladina	Constant	*Constant value:* Požadovaná hladina
Rozdílový_člen_pro_dvoupolohový_regulátor (stejný i pro PID)	Sum	*List of signs:* |+-
Dvoupolohový_regulátor	Relay	*Switch on point:* Hodnota regulační odchylky pro zapnutí přítoku *Switch off point:* Hodnota regulační odchylky pro vypnutí přítoku
Průběh_regulační_odchylky_pro_dvoupolohový_regulátor (vždy i pro regulovanou veličinu, akční veličinu a všechny typy používaných regulátorů).	Scope	*Sample time:* Délka simulace / 225

Tabulka 1 - Použité bloky pro realizaci otevřeného zásobníku na kapalinu I

Název bloku	Typ bloku	Nastavené parametry bloku
Konstantní_přítok	Gain	*Gain:* Konstantní přítok pro regulaci
Přítok_mínus_výtok	Sum	*List of signs:* |+-
Průřez_zásobníku	Gain	*Gain:* 1 / (π*Průměr zásboníku^2 / 4)
Odmocnina_výšky_hladiny	Sqrt	
Konstanta_ventilu	Gain	*Gain:* Konstanta ventilu
Otevřenost_ventilu	Gain	*Gain:* Otevřenost ventilu / 100
Omezení_nádrže	Integrator Limited	*Initial condition:* Počáteční výška hladiny *Upper saturation limit:* Výška zásobníku *Lower saturation limit:* 0
PID_regulátor	PID Controller	*Controller:* Typ regulátoru *Proportional (P):* Proporcionální složka *Integral (I):* Integrační složka *Derivative (D):* Derivační složka *Filter coefficient (N):* Filtr derivační složky *Saturation – Upper limit*: Horní limit akční veličiny *Saturation – Lower limit*: Dolní limit akční veličiny

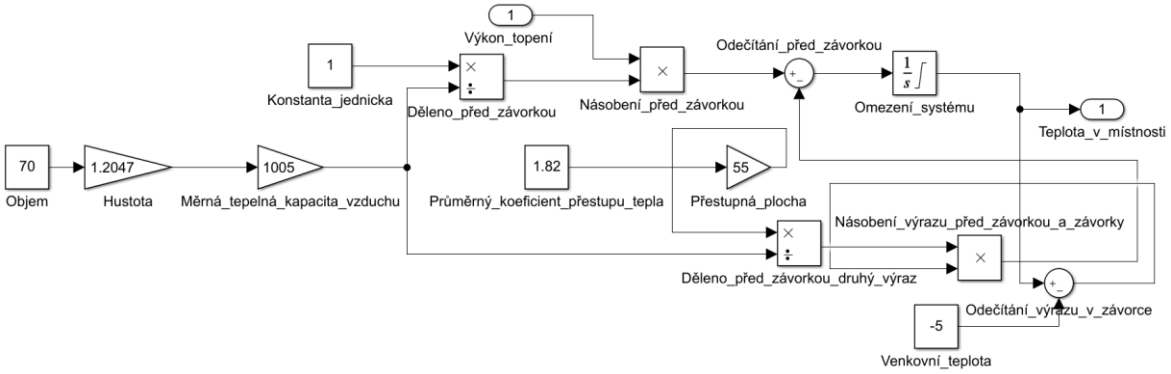
Tabulka 2 - Použité bloky pro realizaci otevřeného zásobníku na kapalinu II

4.3 Model elektrického vytápění místnosti

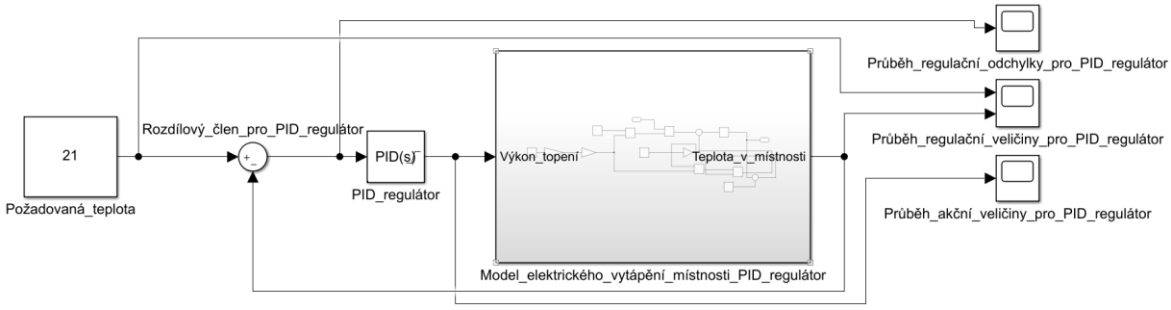
Následující kapitola se věnuje realizaci dalšího modelu, a to konkrétně modelu elektrického vytápění místnosti. Bude představeno zapojení jednotlivých bloků pro realizaci simulace. Stejným způsobem jako u kapitoly 4.2 bude na konci uvedena tabulka, která bude konkrétní pro tento model. Vysvětlení jednotlivých částí tabulky bylo provedeno již v kapitole 4.2.

Realizace simulačních schémat pro PID-regulátor je na Obrázku 29 a 30. Tento simulační obvod se nachází v programu:

BP_Pavlica_Simulink_PID_model_elektrickeho_vytapeni.slx



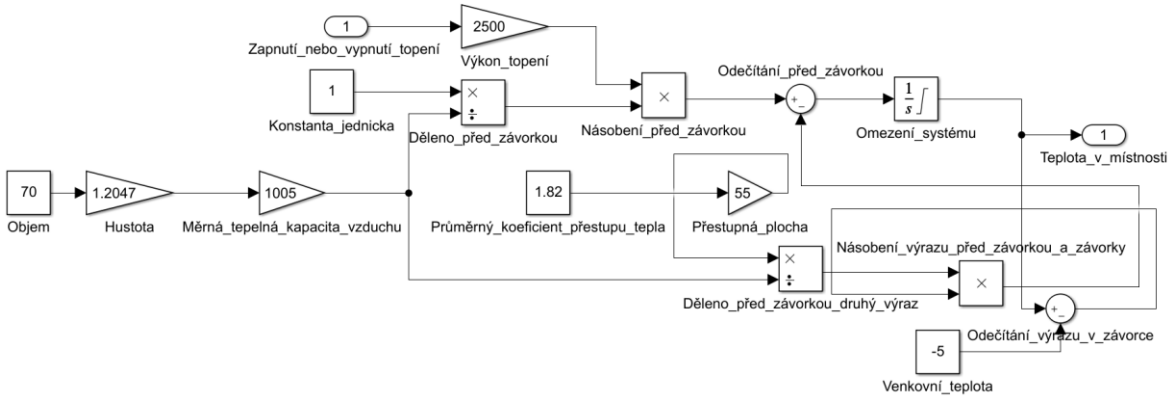
Obrázek 29 - Realizace systému ele. vytápění podle diferenciální rovnice 3.8 pro PID-regulátor



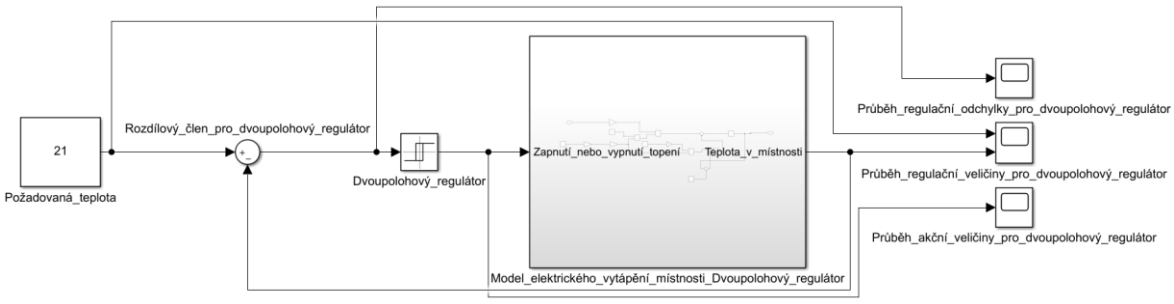
Obrázek 30 - Realizace celého reg. obvodu pro systém ele. vytápění a PID-regulátor

Realizace simulačních schémat pro 2-polohový regulátor je na Obrázku 31 a 32. Oproti PID-regulaci má 2-polohová regulace doplněn blok definující konstantní výkon topení, ten lze nastavit pomocí GUI. Tento simulační obvod se nachází v programu:

BP_Pavlica_Simulink_2Polohovy_model_elektrickeho_vytapeni.slx



Obrázek 31 - Realizace systému ele. vytápění podle diferenciální rovnice 3.8 pro 2-polohový regulátor



Obrázek 32 - Realizace celého reg. obvodu pro systém ele. vytápění a 2-polohový regulátor

Název bloku	Typ bloku	Nastavené parametry bloku
Požadovaná_teplota	Constant	*Constant value:* Požadovaná teplota
Rozdílový_člen_pro_dvoupolohový_regulátor (stejný i pro PID)	Sum	*List of signs:* |+-
Dvoupolohový_regulátor	Relay	*Switch on point:* Hodnota regulační odchylky pro zapnutí topení *Switch off point:* Hodnota regulační odchylky pro vypnutí topení
Průběh_regulační_odchylky_pro_dvoupolohový_regulátor (vždy i pro regulovanou veličinu, akční veličinu a všechny typy používaných regulátorů).	Scope	*Sample time:* Délka simulace / 225
PID_regulátor	PID Controller	*Controller:* Typ regulátoru *Proportional (P):* Proporcionální složka *Integral (I):* Integrační složka *Derivative (D):* Derivační složka *Filter coefficient (N):* Filtr derivační složky *Saturation – Upper limit*: Horní limit akční veličiny *Saturation – Lower limit*: Dolní limit akční veličiny

Tabulka 3 - Použité bloky pro realizaci elektrického vytápění místnosti I

Název bloku	Typ bloku	Nastavené parametry bloku
Výkon_topení	Gain	*Gain:* Výkon topení
Konstanta_jednička	Constant	*Constant value:* 1
Objem	Constant	*Constant value:* Objem místnosti
Hustota	Gain	*Gain:* Hustota vzduchu
Měrná_tepelná_kapacita_vzduchu	Gain	*Gain:* Měrná tepelná kapacita vzduchu
Děleno_před_závorkou	Divide	*Number of inputs:* */
Násobení_před_závorkou	Product	*Number of inputs:* 2
Odečítání_před_závorkou	Sum	*List of signs:* |+-
Omezení_systému	Integrator Limited	*Initial condition:* Počáteční teplota v místnosti *Upper saturation limit:* Maximální teplota v místnosti *Lower saturation limit:* -30
Průměrný_koeficient_přestupu_tepla	Constant	*Constant value:* Průměrný koeficient přestupu tepla
Přestupná_plocha	Gain	*Gain:* Přestupná plocha
Děleno_před_závorkou_druhý_výraz	Divide	*Number of inputs:* */
Násobení_výrazu_před_závorkou_a_závorky	Product	*Number of inputs:* 2
Odečítání_výrazu_v_závorce	Sum	*List of signs:* |+-
Venkovní_teplota	Constant	*Constant value:* Venkovní teplota

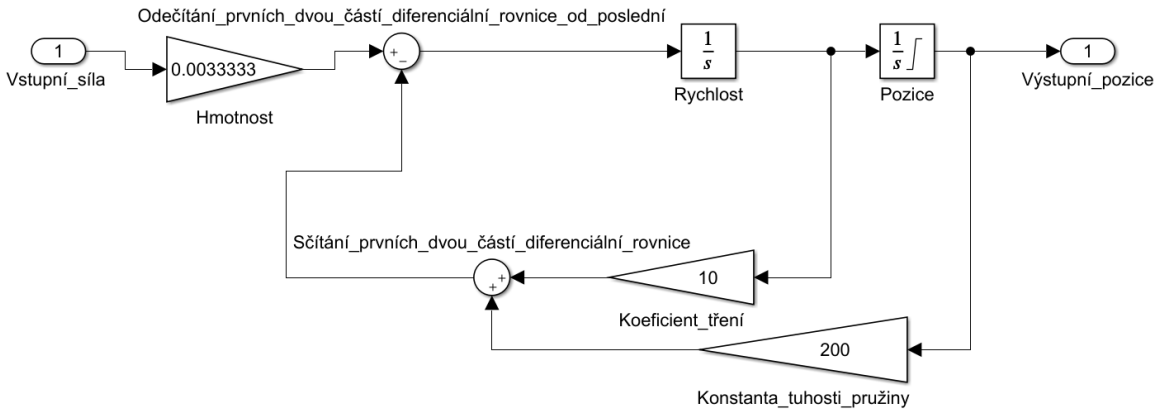
Tabulka 4 - Použité bloky pro realizaci elektrického vytápění místnosti II

4.4 Model mechanického oscilátoru

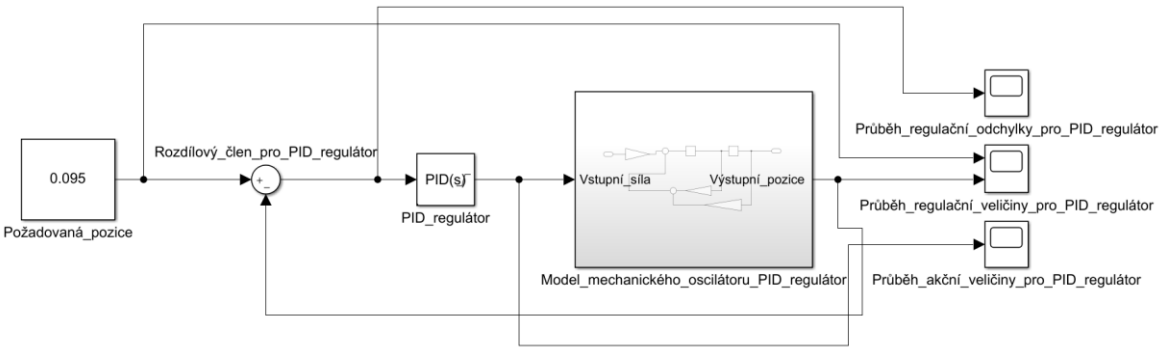
Tato kapitola se věnuje realizaci modelu mechanického oscilátoru. Opět bude představeno zapojení jednotlivých bloků pro realizaci simulace. Stejným způsobem jako u kapitoly 4.2 a 4.3 bude na konci uvedena přehledná tabulka použitých bloků, jejíž bylo provedeno již v kapitole 4.2.

Realizace simulačních schémat pro PID-regulátor je na Obrázku 33 a 34. Tento simulační obvod se nachází v programu:

BP_Pavlica_Simulink_PID_model_mechanickeho_oscilatoru.slx



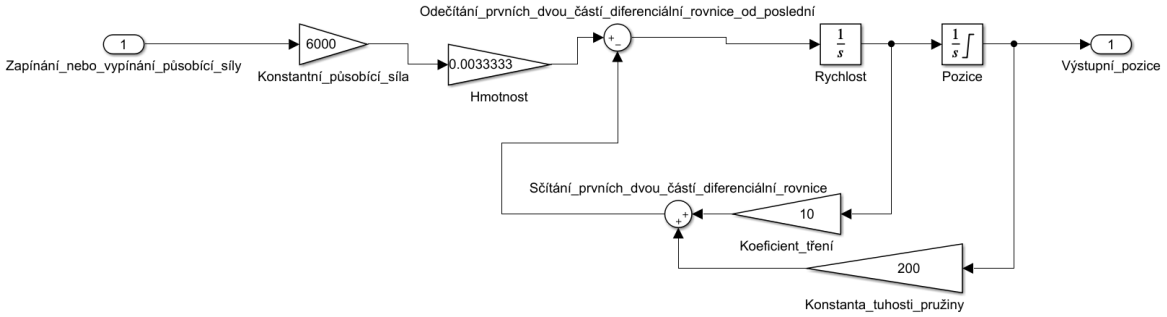
Obrázek 33 - Realizace systému mech. oscilátoru podle diferenciální rovnice 3.11 pro PID-regulátor



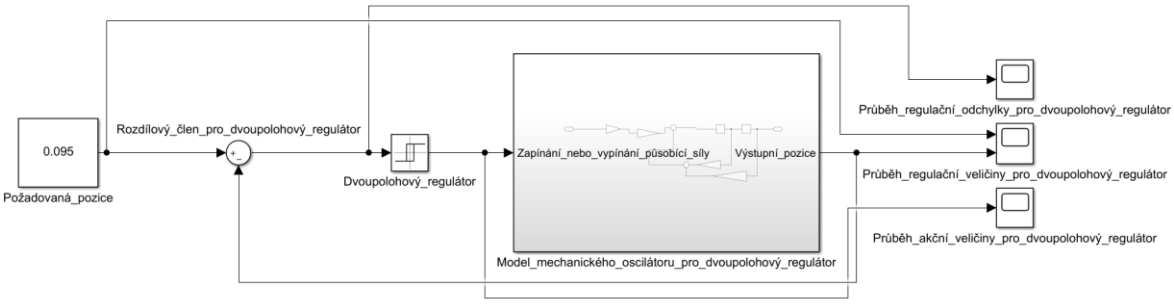
Obrázek 34 – Realizace celého reg. obvodu pro mech. oscilátor a PID-regulátor

Realizace simulačních schémat pro 2-polohový regulátor je na Obrázku 35 a 36. Oproti PID-regulaci má 2-polohová regulace doplněn blok definující konstantní působící sílu, tu lze nastavit pomocí GUI. Tento simulační obvod se nachází v programu:

BP_Pavlica_Simulink_2Polohovy_model_mechanickeho_oscilatoru.slx



Obrázek 35 - Realizace systému mech. oscilátoru podle diferenciální rovnice 3.11 pro 2-polohový regulátor



Obrázek 36 - Realizace celého reg. obvodu pro mech. oscilátor a 2-polohový regulátor

Název bloku	Typ bloku	Nastavené parametry bloku
Požadovaná_pozice	Constant	*Constant value:* Požadovaná pozice
Rozdílový_člen_pro_dvoupolohový_regulátor (stejný i pro PID)	Sum	*List of signs:* |+-
Dvoupolohový_regulátor	Relay	*Switch on point:* Hodnota regulační odchylky pro zapnutí síly *Switch off point:* Hodnota regulační odchylky pro vypnutí síly
Průběh_regulační_odchylky_pro_dvoupolohový_regulátor (vždy i pro regulovanou veličinu, akční veličinu a všechny typy používaných regulátorů).	Scope	*Sample time:* Délka simulace / 225
PID_regulátor	PID Controller	*Controller:* Typ regulátoru *Proportional (P):* Proporcionální složka *Integral (I):* Integrační složka *Derivative (D):* Derivační složka *Filter coefficient (N):* Filtr derivační složky *Saturation – Upper limit*: Horní limit akční veličiny *Saturation – Lower limit*: Dolní limit akční veličiny

Tabulka 5 - Použité bloky pro realizaci mechanického oscilátoru I

Název bloku	Typ bloku	Nastavené parametry bloku
Konstatní_působící_síla	Constant	*Constant value:* Konstatní síla regulátoru
Hmotnost	Gain	*Gain:* 1 / Hmotnost
Odečítání_prvních_dvou_částí_diferenciální_rovnice_od_poslední	Sum	*List of signs:* |+-
Rychlost	Integrator	*Initial condition:* 0
Pozice	Integrator Limited	*Initial condition:* Počáteční pozice *Upper saturation limit:* 0.3 *Lower saturation limit: 0*
Konstanta_tuhosti_pružiny	Gain	*Gain:* Konstanta tuhosti pružiny / Hmotnost
Koeficient_tření	Gain	*Gain:* Koeficient tření / Hmotnost
Sčítání_prvních_dvou_částí_diferenciální_rovnice	Sum	*List of signs:* |++

Tabulka 6 - Použité bloky pro realizaci mechanického oscilátoru II

5 REALIZACE GRAFICKÉHO UŽIVATELSKÉHO ROZHRANÍ

Grafické uživatelské rozhraní jsem implementoval pomocí interaktivního prostředí *App designer* ve verzi *MATLABu* 2020a. *App designer* jsem si vybral z mnoha důvodu. Hlavním důvodem byl fakt, že jsem již *App designer* znal z předmětu *Matlab* a *Simulink*, který jsem absolvoval ve 2. ročníku studia a závěrečný projekt v tomto předmětu jsem rovněž dělal v *App designeru*. Druhým podstatným důvodem bylo, že *App designer* nabízí z pohledu rychlosti a podpory budoucí kompability nejlepší výsledky.

5.1 Používané komponenty pro sestavení aplikace

V *App designeru* jsem využíval základní komponenty, které jsou v knihovně nabízeny.

Komponenty

Tab group

Jednalo se o první komponentu, kterou jsem použil. Tato komponenta mi umožnila vytvořit v jedné aplikaci několik záložek, mezi kterými se uživatel může přepínat a tím zobrazit jiný obsah.

Panel

Panel byl použit pro oddělení obsahu v rámci jedné záložky. Panel mi umožnoval nastavit titulek, který je vždy v horní části příslušného panelu, takže uživatel aplikace hned ví, co v dané části může očekávat.

Edit field (Numeric)

Jedná se o nejčastěji používanou komponentu v mé aplikaci. Jedná se o vstupní pole, které je omezené pouze na zadávání čísel.

Knob

Knob neboli potenciometr používám jako druhý způsob pomoci kterého může uživatel zadávat číselný vstup.

Label

Jedná se o textovou formu zápisu. V této práci našel *label* dvě využití. V prvním způsobu využití používám label pouze jako určitou formu oznámení (text se během používání aplikace nemění). V druhém způsobu používám label jako výstup určitého výsledku za běhu programu, ať už číselného nebo textového (text se během používání aplikace mění).

Drop down

Používán jako rozbalovací seznam pro zvolení přednastavených možností.

Image

Slouží pro vkládání obrázků do aplikace.

Button

Jedná se o tlačítko, které po stisku vykoná určitou činnost.

State Button

Stavové tlačítko. Má dvě pozice, proto ho používám konkrétně pouze ve dvou případech. Jedním z nich je moment, kdy uživatel chce pozastavit animaci. Po prvním stisku tlačítka se animace pozastaví a po druhém stisku znovu spustí. Druhým případem je zobrazení nových možností pro omezení akčního zásahu.

Check box

Jedná se o zaškrtávací políčko. V mé aplikaci našlo využití pouze u možnosti, kdy uživatel má na výběr, zda chce animovat simulaci (postupné vykreslování) nebo rovnou chce získat celkový průběh.

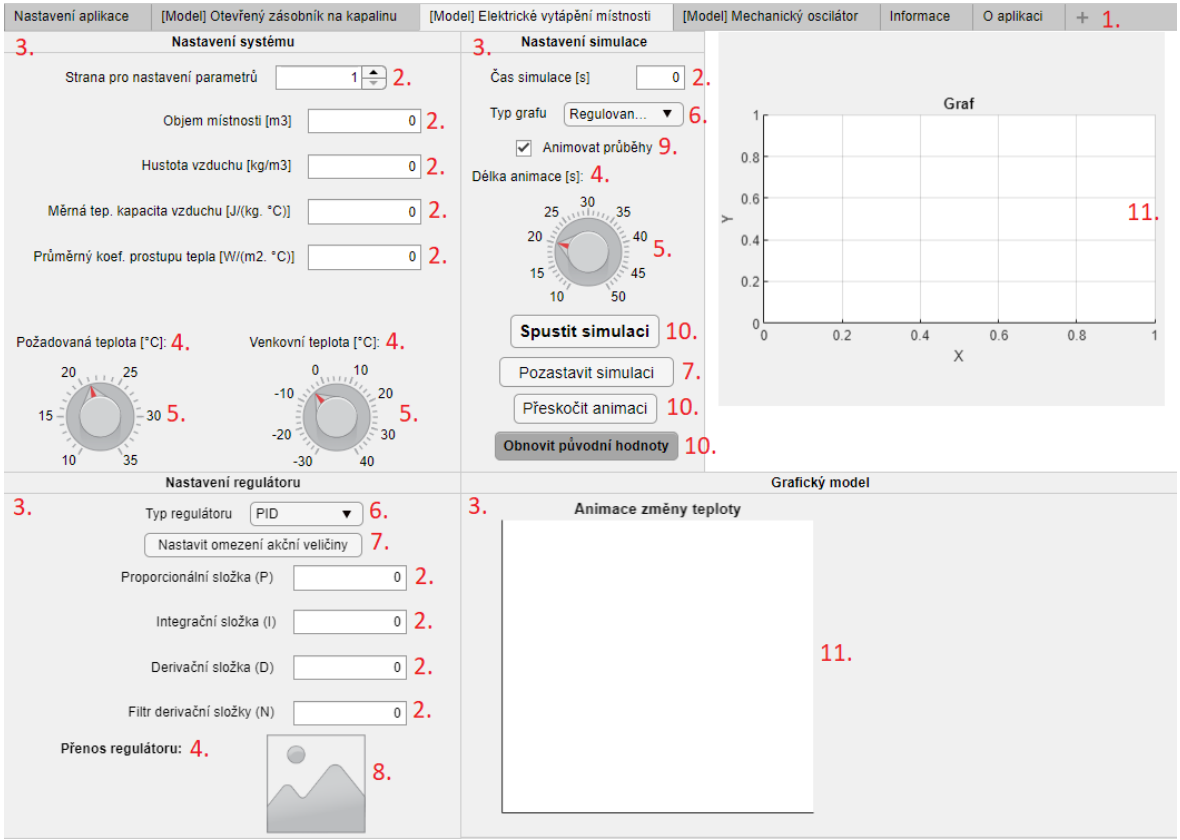
Axes

Slouží pro vykreslování grafu. V mé aplikaci našla tato komponenta využití nejen pro vykreslení průběhu simulací, ale i k zobrazení animací na vymodelovaných modelech.

Spinner

Jedná se o interaktivní možnost, jak uživatel pomocí dvou tlačítek může měnit hodnotu čísla. Tento prvek jsem využil pouze u záložky model elektrického vytápění místnosti. Konkrétně pro možnost přepnutí stránky pro nastavení parametrů (jsou zde dvě stránky).

Navrhnutá aplikace pro záložku model elektrického vytápění místnosti v *design view* v App designeru je vidět na Obrázku 37.



Obrázek 37 - Navržené rozložení komponent v prostředí App designer

Legenda k Obrázku 37.

1. Tab group (záložka)
2. Spinner (inkrementační/dekrementační okno)
3. Panel
4. Label (text)
5. Knob (potenciometr)
6. Drop down (rozbalovací menu)
7. State button (stavové tlačítko)
8. Image (obrázek)
9. Check box (zaškrtávací okénko)
10. Button (tlačítko)
11. Axes (graf)

5.2 Implementované funkce

Hlavní funkčnost aplikace spočívá v implementování kódové části programu. Pro přehlednost programu a pro co nejmenší opakovanost kódu jsem do programové části začlenil několik vytvořených funkcí. Programovaná část aplikace má aktuálně přes 5500 řádků z toho pouze okolo 1500 řádků je vygenerováno App designerem, proto nelze rozebírat komplexně funkčnost a popis jednotlivých funkcí. Z tohoto důvodu alespoň uvedu syntaxi všech vytvořených funkcí včetně popisu jejich práce.

Vytvořené funkce

function kontrola = kontrola_vstupnich_parametru(app, typ_modelu)

Vstupní parametry funkce:

- *app* – třída aplikace
- *typ_modelu* – textový vstup určující o jaký typ modelu se jedná

Výstupní parametr funkce:

- *kontrola* – parametr, který nese hodnotu *true* nebo *false* v závislosti na úspěšnosti kontroly parametrů

Popis funkce:

- Funkce kontroluje splnění všech podmínek
- Pokud jsou všechny podmínky splněny, tak proměnná *kontrola* vrací hodnotu *true* v opačném případě *false*
- V této funkci se kontrolují zadané vstupní hodnoty uživatelem
- Více o těchto podmínkách je uvedeno v tabulkách v jednotlivých kapitolách 5.4, 5.5, 5.6

function kontrola_souboru = Overeni_korenovych_souboru(app)

Výstupní parametr funkce:

- *kontrola_souboru* – parametr, který nese hodnotu *true* nebo *false* v závislosti na úspěšnosti provedené kontroly potřebných souborů

Popis funkce:

- Funkce kontroluje nalezení všech potřebných souborů pro správné ovládání aplikace
- Funkce očekává stejné rozložení složek a souborů ve složce Vizualizace_zakladnich_algoritmu_rizeni jako v moment, kdy uživatel aplikaci stáhnul
- Pokud jsou všechny potřebné soubory nalezeny, tak proměnná *kontrola_souboru* vrací hodnotu *true* v opačném případě *false*

function Zobrazeni_vzorce(app, typ_regulatoru, typ_modelu)

Vstupní parametry funkce:

- *typ_regulatoru* – textový vstup určující o jaký typ regulátoru se jedná
- *typ_modelu* – textový vstup určující o jaký typ modelu se jedná

Popis funkce:

- Funkce zobrazuje vzorec přenosu regulátoru pro zvolený typ regulátoru a konkrétní model
- Zobrazování vzorce je realizováno pomocí přepínaní obrázků

function [simulacni_krok] = Vypocet_poctu_simulacniho_kroku(app, cas_simulace)

Vstupní parametr funkce:

- *cas_simulace* – číselná hodnota označující požadovanou dobu simulace (zadává uživatel skrze GUI)

Výstupní parametr funkce:

- *simulacni_krok* – číselná hodnota označující periodu zaznamenávání sledované veličiny při simulování systému (taktéž integrační krok). Je počítán jako *požadovaný čas / 225*. Kde číslo 225 jsem si pevně stanovil (postupně odhalil) jako vhodný počet simulačních hodnot z pohledu vykreslování průběhu simulací a animací.

Popis funkce:

- Funkce počítá délku simulačního kroku a vrací jeho hodnotu

function [delka_animace] = Vypocet_delky_animace(app, pozadovany_cas)

Vstupní parametr funkce:

- *pozadovany_cas* – číselná hodnota označující požadovanou dobu animování simulovaných průběhů (zadává uživatel skrze GUI)

Výstupní parametr funkce

- *delka_animace* – číselná hodnota označující prodlevu mezi vykreslením jednotlivých hodnot v animaci. Je počítána jako *požadovaný čas / 225*.

Popis funkce:

- Funkce počítá délku pauzy mezi vykreslením hodnot v grafu a grafickém modelu
- Vypočítaný čas se používá v zabudované funkci *pause(čas)*

function [x, y, t1, t2, t3, t4] = Vykresli_graficky_model_otevreneho_zasobniku_na_kapalinu(app, vyplnovat_trubici)

Vstupní parametr funkce:

- *vyplnovat_trubici* – parametr, který nese hodnotu *true* nebo *false* v závislosti na tom, jestli je odtokový ventil otevřený nebo ne

Výstupní parametr funkce:

- x – délka zásobníku na ose X
- y – výška zásobníku na ose Y
- t1, t2 – souřadnice objektu, který animuje přítok do zásobníku
- t3, t4 – souřadnice objektu, který animuje odtok ze zásobníku

Popis funkce:

- Funkce vykreslí model otevřeného zásobníku proporcionálně podle zadané velikosti zásobníku uživatelem v GUI
- Funkce v modelu označí požadovanou výšku hladiny
- Pokud dostane informaci, že ventil je otevřený, vyplní odtokovou trubicí vodou (tj. modrou barvou)

function Vykresli_graf_pro_model_otevreneho_zasobniku_na_kapalinu(app)

Popis funkce:

- Funkce je řízena globálními proměnnými, které funkci říkají, jaký typ regulátoru uživatel vybral a jaký typ grafu chce vykreslit
- Podle typu regulátoru funkce v pozadí spustí simulaci potřebného *Simulink* souboru a získá naměřená data
- Naměřená data následně vykreslí podle požadovaného grafu
- Zároveň s vykreslováním grafu provádí animaci změny výšky hladiny v modelu

function [xKruh, yKruh] = Vykresli_graficky_model_teplomeru_pro_vytapeni_mistnosti(app)

Výstupní parametry funkce:

- *xKruh*, *yKruh* – určují velikost oblasti vykresleného půlkruhu ve spodní části teploměru

Popis funkce:

- Funkce vykreslí grafický model teploměru
- Popíše stupnici teploměru
- Na teploměru označí požadovanou hodnotu a hodnotu tolerance, pokud byla zadána

function zmena_barvy = Zmen_barvu_teplomeru(app, xKruh, yKruh, teplota)

Vstupní parametry funkce:

- xKruh, yKruh – určují velikost oblasti vykresleného půlkruhu ve spodní části teploměru
- teplota – aktuální hodnota teploty

Výstupní parametr funkce:

- změna_barvy – hodnota RGB kódu požadované barvy

Popis funkce:

- Funkce porovnává aktuální hodnotu teploty s teplotou, která odpovídá horní a dolní hranici tolerance
- Pokud je teplota pod toleranci, tak funkce nastaví spuštění souboru zima.gif a nastaví modrou barvu výplně teploměru
- Pokud je teplota v mezích, tak funkce nastaví spuštění souboru vporadku.gif a nastaví zelenou barvu výplně teploměru
- Pokud je teplota nad toleranci, tak funkce nastaví spuštění souboru horko.gif a nastaví červenou barvu výplně teploměru

function Vykresli_graf_pro_model_elektrickeho_vytapeni_mistnosti(app)

Popis funkce:

- Funkce je řízena globálními proměnnými, které funkci říkají, jaký typ regulátoru uživatel vybral a jaký typ grafu chce vykreslit
- Podle typu regulátoru funkce v pozadí spustí simulaci potřebného *Simulink* souboru a získá naměřená data
- Naměřená data následně vykreslí podle požadovaného grafu
- Zároveň s vykreslováním grafu provádí animaci změny teploty na teploměru

function Vykresli_graficky_model_mechanickeho_oscilatoru(app, pozice)

Vstupní parametr funkce:

- *pozice* – hodnota pozice oscilátoru (zadává uživatel skrze GUI)

Popis funkce:

- Funkce vykreslí grafický model oscilátoru
- Na modelu označí počáteční pozici oscilátoru a požadovanou pozici

function Vykresli_graf_pro_model_mechanickeho_oscilatoru(app)

Popis funkce:

- Funkce je řízena globálními proměnnými, které funkci říkají, jaký typ regulátoru uživatel vybral a jaký typ grafu chce vykreslit
- Podle typu regulátoru funkce v pozadí spustí simulaci potřebného *Simulink* souboru a získá naměřená data
- Naměřená data následně vykreslí podle požadovaného grafu
- Zároveň s vykreslováním grafu provádí animaci – např. změny stlačení pružiny

5.2.1 Callbacks funkce

Jedná se o velmi podstatnou část pro chod programu. *Callbacks* funkce jsou vestavěné funkce *App designeru*, které umožní reagovat na určité změny (iniciované uživatelem). Ve své aplikaci používám *callbacks* funkce převážně pro změnu vstupních parametrů modelů. Pokaždé když uživatel změní hodnotu nějakého vstupního parametru nebo zmáčkne tlačítko, tak se zavolá příslušná *callback* funkce a provede se daný kód. Ve svém programu používám ještě další dva typy *callback* funkcí, a to funkci, která se provede v moment kdy uživatel zapne aplikaci nebo když se snaží aplikaci zavřít. Aplikace těchto *callbacks* funkcí obsahuje více jak 90, proto nechci do tohoto tématu příliš zabíhat. Pro nastínění funkce uvedu následující příklad:

```
% Value changed function: O_Z_N_K_Vstup_Typ_Grafu
function O_Z_N_K_Vstup_Typ_GrafuValueChanged(app, event)
  app.Model_O_Z_N_K_Vybrany_typ_grafu = app.O_Z_N_K_Vstup_Typ_Grafu.Value;

  if app.Model_O_Z_N_K_Vybrany_typ_grafu == "Regulovaná veličina"
      title(app.O_Z_N_K_Graficke_Prubehy, "Regulovaná veličina");
      ylabel(app.O_Z_N_K_Graficke_Prubehy, "h [m]");
  elseif app.Model_O_Z_N_K_Vybrany_typ_grafu == "Regulační odchylka"
      title(app.O_Z_N_K_Graficke_Prubehy, "Regulační odchylka");
      ylabel(app.O_Z_N_K_Graficke_Prubehy, "e(t) [m]");
  else
      title(app.O_Z_N_K_Graficke_Prubehy, "Akční veličina");
      ylabel(app.O_Z_N_K_Graficke_Prubehy, "q_v [m"+char(179)+"/s]");
  end
end
```

Kód, který zde byl uveden jako příklad *callback* funkce se vykoná vždy, když uživatel v záložce model otevřeného zásobníku na kapalinu, změní hodnotu typu grafu. Hodnota vybraného typu grafu se uloží do globální proměnné a na základě typu grafu se změní název grafu a popisek osy.

5.3 Záložka nastavení aplikace

Nyní bych rád uvedl několik informací k realizaci jednotlivých záložek. Budu prezentovat pouze podstatné informace, protože aplikace je programově tak obsáhlá, že kdybych chtěl shrnout vše, tak bych se nevešel na 200. stránkovou práci.

Záložka Nastavení aplikace je úvodní záložka, která je uživateli zobrazena při každém spuštění aplikace. Tato záložka je také ze všech nejpodstatnější, protože je prvním krokem k tomu, aby uživatel mohl dále aplikaci používat.

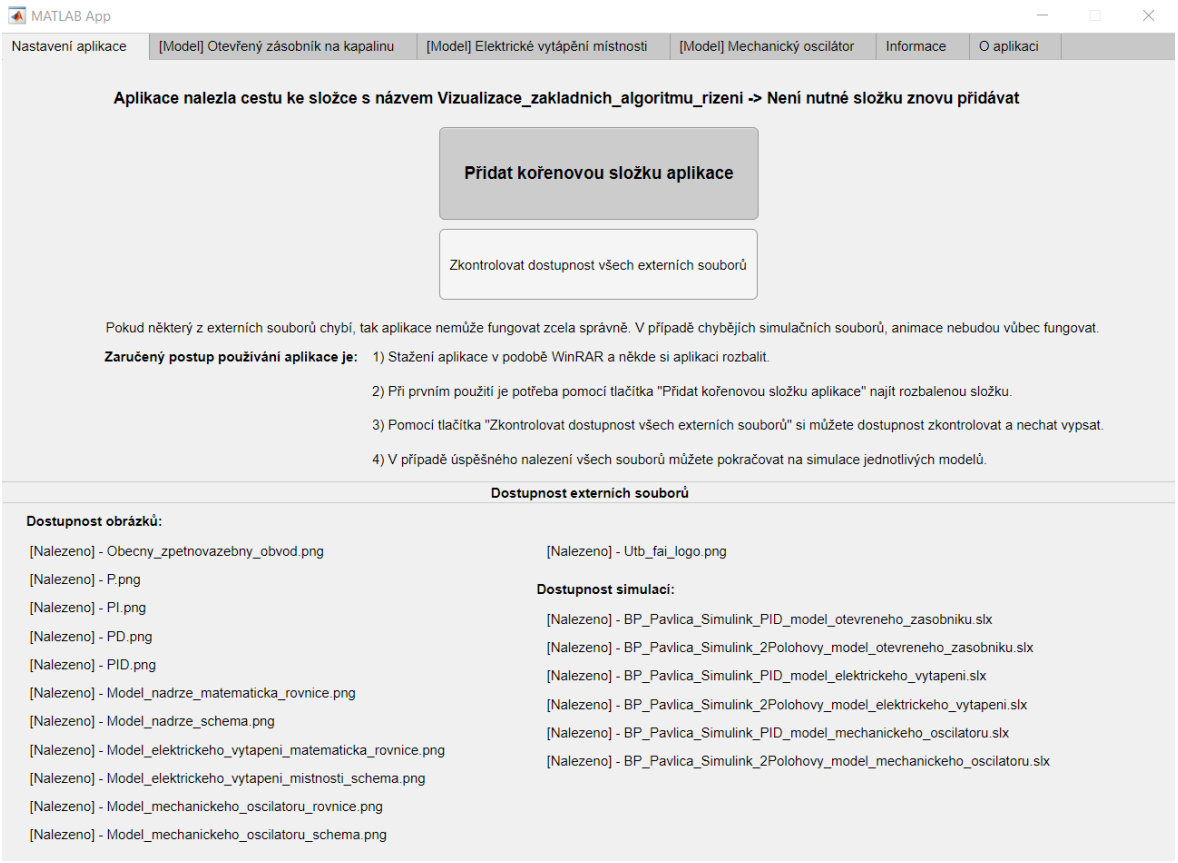
Uživatel je zde informován, zda aplikace v jeho počítači má k dispozici všechny potřebné soubory pro její správný běh.

Při úplně prvním spuštění je uživatel nucen zadat cestu ke stažené aplikaci. Stiskem tlačítka *Přidat kořenovou složku aplikace*, se otevře průzkumník složek, kde by uživatel měl najít staženou složku s názvem Vizualizace_zakladnich_algoritmu_rizeni a tuto složku označit.

Po přidání složky se provede kontrola, zda všechny potřebné soubory jsou nyní k dispozici. Výsledky jednotlivých testů má uživatel k dispozici přímo v aplikaci v této záložce. Pokud není u všech souboru stav *Nalezeno*, tak uživatel pravděpodobně zadal špatnou cestu ke kořenové složce nebo složka neobsahuje všechny potřebné soubory.

Nalezení všech souborů je jednou ze dvou podmínek k tomu, aby uživatel byl schopen spustit simulaci.

Na Obrázku 38 je ukázka záložky Nastavení aplikace při spuštění programu. Na tomto obrázku je ideální stav aplikace, kdy všechny soubory jsou nalezené.



Obrázek 38 - Ukázka spuštěné aplikace – záložka Nastavení aplikace

5.4 Záložka otevřený zásobník na kapalinu

Jedná se o první záložku pro simulování systému. Uživatel má možnost v této záložce simulovat systém otevřeného zásobníku na kapalinu a jeho regulaci.

V sekci nastavení systému si uživatel nastavuje parametry systému.

V sekci nastavení regulátoru si uživatel volí typ regulátoru (P, PI, PD, PID nebo 2-polohový), následně si uživatel nastaví parametry pro zvolený typ regulátoru. V případě, že uživatel chce nastavit omezení akční veličiny stiskne tlačítko *nastavit omezení akční veličiny* a nastaví si zvolenou min. a max. hodnotu.

V sekci nastavení simulace si uživatel nastaví dobu simulace, vybere si typ grafu, může si zvolit možnost, zda chce průběhy animovat nebo chce rovnou znát celý průběh.

Uživatel si dale může určit pomocí potenciometru délku animace. Délka animace odpovídá přibližnému času, jak dlouho bude trvat vykreslení celého průběhu.

Pokud je uživatel připraven simulovat daný systém, stiskne tlačítko spustit simulaci. Po stisku tlačítka se ověří podmínky regulernosti zadaných dat uživatelem. Pokud vše proběhne v pořádku začne se systém simulovat (výsledky se dostaví s určitou časovou prodlevou). Během simulace má uživatel možnost simulaci pozastavit nebo zcela přeskočit pomocí tlačítek v sekci *nastavení simulace*.

Zvolené počáteční hodnoty, které jsou při každém spuštění aplikace nastaveny jsou odzkoušené a lze na nich dobře pozorovat vlivy jednotlivých regulátorů a jejich složek. Pokud uživatel bude chtít znovu tyto hodnoty načíst, tak nemusí vypínat a zapínat celou aplikaci znovu, ale stačí zmáčkout tlačítko *obnovit původní hodnoty*.

Podmínky pro zadané vstupní hodnoty parametrů

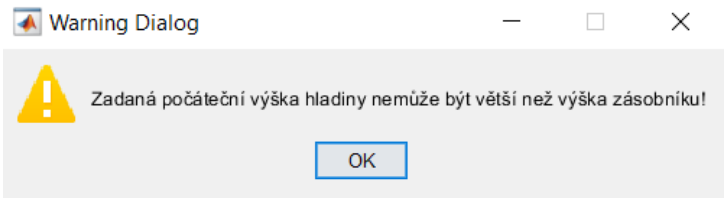
Vstupní parametry	Typ omezení	Hodnota omezení
Průměr zásobníku	>	0
Počáteční výška hladiny	>=	0
Požadovaná výška hladiny	>=	0
Výšky zásobníku	>	0
Čas simulace	>	0
Konstantní přítok	>=	0
Počáteční výška hladiny	<=	Výška zásobníku
Požadovaná výška hladiny	<=	Výška zásobníku
Hodnota regulační odchylky pro zapnutí přítoku	>=	0
Hodnota regulační odchylky pro zapnutí přítoku	>=	Hodnota regulační odchylky pro vypnutí přítoku

Tabulka 7 - Podmínky pro zadané vstupní hodnoty u modelu otevřeného zásobníku na kapalinu I

Vstupní parametry	Typ omezení	Hodnota omezení
Horní limit akční veličiny	>=	0
Horní limit akční veličiny	<=	1
Dolní limit akční veličiny	>=	0
Dolní limit akční veličiny	<=	Horní limit akční veličiny

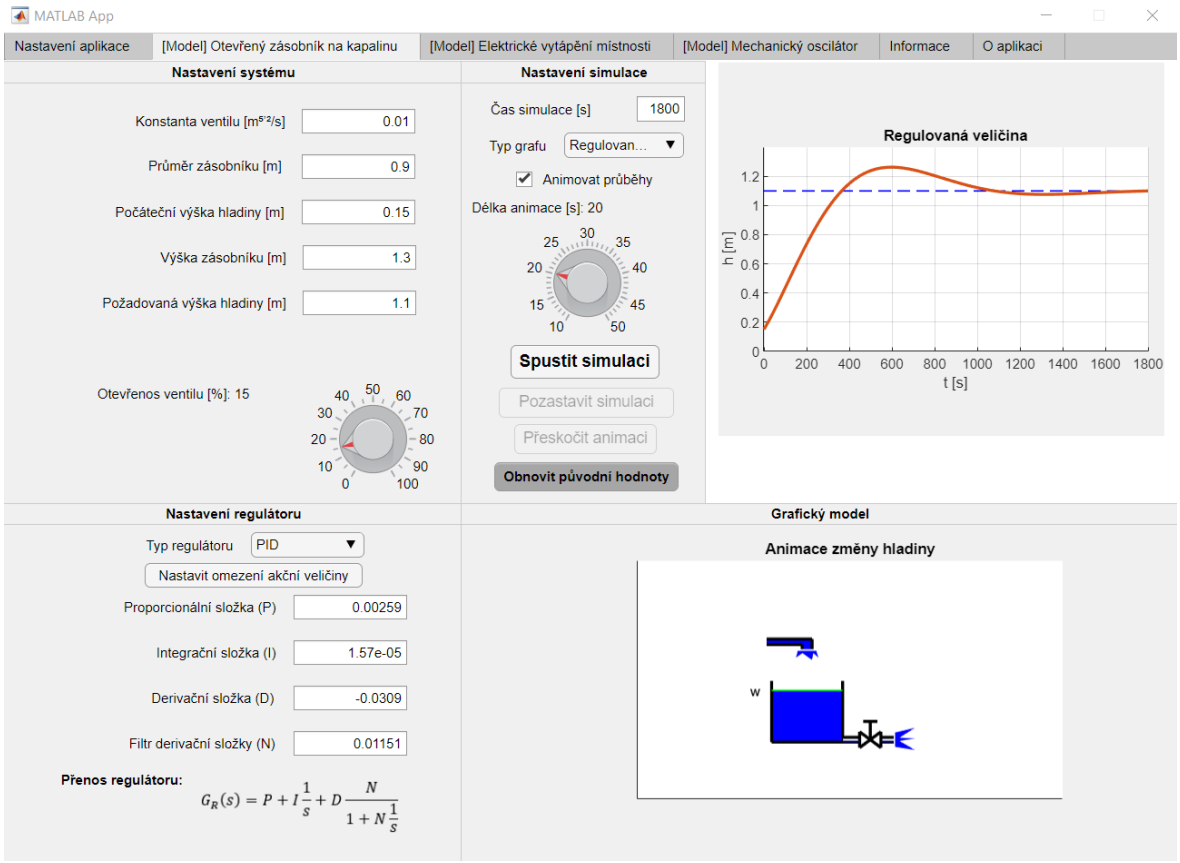
Tabulka 8 - Podmínky pro zadané vstupní hodnoty u modelu otevřeného zásobníku na kapalinu II

V případě, že uživatel nesplní podmínky korektnosti vstupních dat, bude mu zobrazeno chybové hlášení, které ho upozorní na chyby v konkrétních vstupních hodnotách. Konkrétní chybová hláška je vidět na Obrázku 39.



Obrázek 39 - Ukázka chybového hlášení při zadání nepovolených hodnot

Na Obrázku 40 je vidět dokončená simulace včetně animace pro model otevřeného zásobníku na kapalinu. Z nastavení simulace lze vyčíst např. že uživatel si zvolil PID-regulátor, typ sledované veličiny byla regulovaná veličina, chtěl postupné vykreslování animace a délka celkové animace byla přibližně 20s.



Obrázek 40 - Ukázka spuštěné aplikace – záložka modelu otevřeného zásobníku na kapalinu

5.5 Záložka elektrické vytápění místnosti

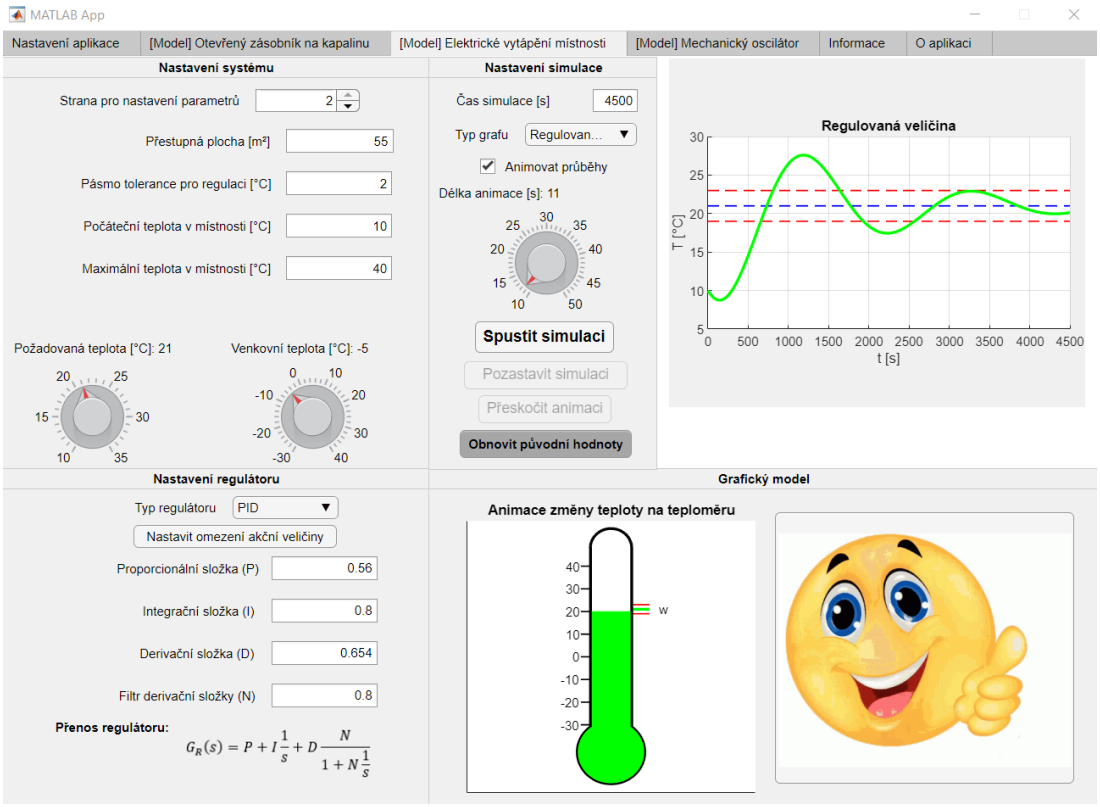
Záložky, které slouží pro simulaci systémů jsou téměř totožné. Změna nastává pouze v části nastavení systémů, kde dochází k nastatování parametrů pro konkrétní model. Princip celé záložky byl podrobněji popsán v kapitole 5.4.

Podmínky pro zadané vstupní hodnoty parametrů

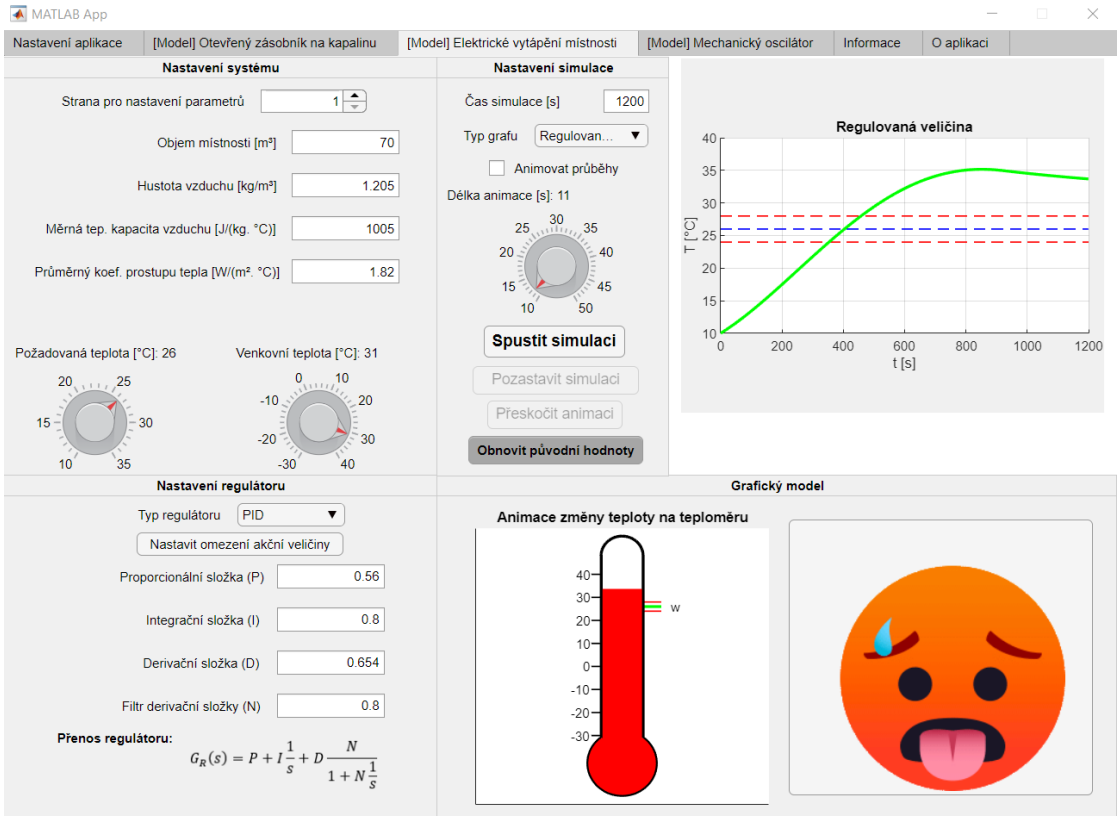
Vstupní parametry	Typ omezení	Hodnota omezení
Objem místnosti	>	0
Přestupná plocha	>	0
Čas simulace	>	0
Pásmo tolerance pro regulaci	>=	0
Výkon topení	>=	0
Hodnota regulační odchylky pro zapnutí topení	>=	0
Hodnota regulační odchylky pro zapnutí topení	>=	Hodnota regulační odchylky pro vypnutí topení
Maximální teplota v místnosti	<=	45
Maximální teplota v místnosti	>=	10
Počáteční teplota v místnosti	>=	-30
Počáteční teplota v místnosti	<=	40
Dolní limit akční veličiny	>=	0
Dolní limit akční veličiny	<=	Horní limit akční veličiny

Tabulka 9 - Podmínky pro zadané vstupní hodnoty u modelu elektrického vytápění místnosti

Na Obrázku 41 a 42 je vidět simulace včetně animace pro model elektrického vytápění místnosti. Rozdíl mezi těmito simulacemi je ve vstupních hodnotách systémů a délky simulace.



Obrázek 41 - Ukázka spuštěné aplikace – záložka modelu ele. vytápění místnosti I



Obrázek 42 - Ukázka spuštěné aplikace – záložka modelu ele. vytápění místnosti II

5.6 Záložka mechanický oscilátor

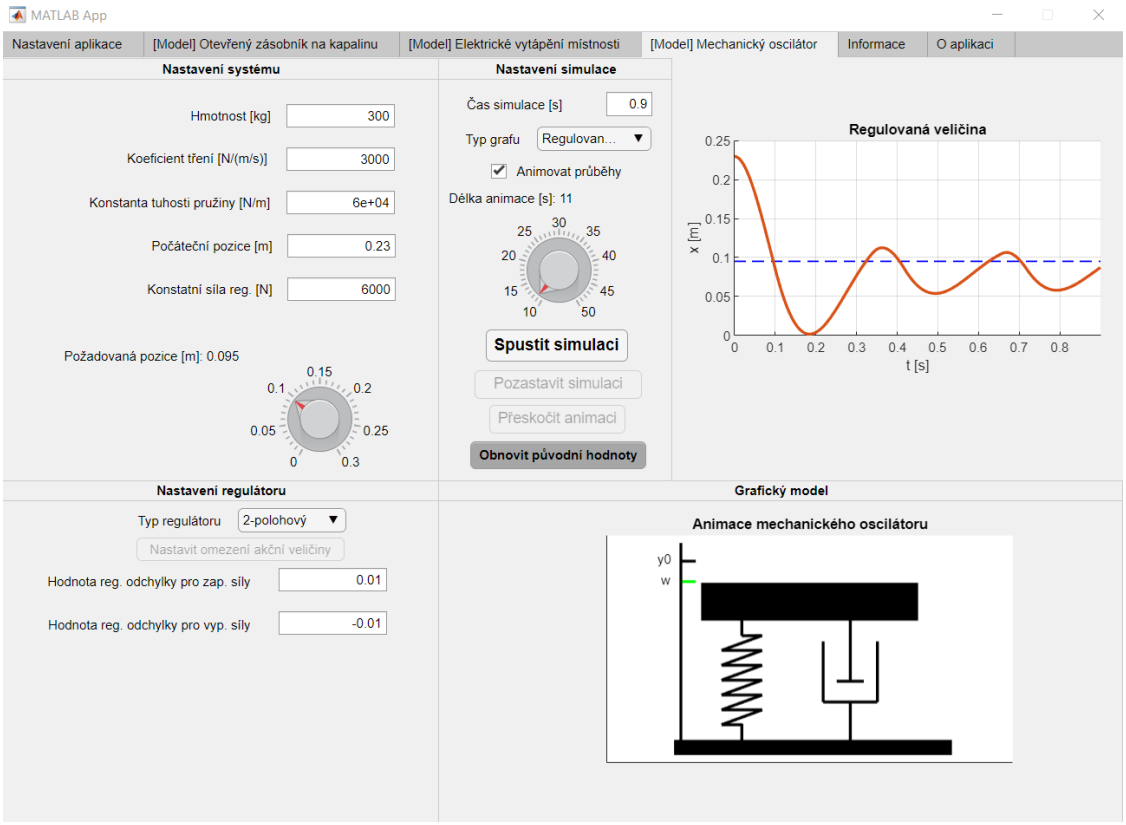
Záložky, které slouží pro simulaci systémů jsou téměr totožné. Změna je pouze v části nastavení systémů, kde dochází k nastavování parametrů pro konkrétní model. Princip celé záložky byl již podrobněji popsán v kapitole 5.4.

Podmínky pro zadané vstupní hodnoty parametrů

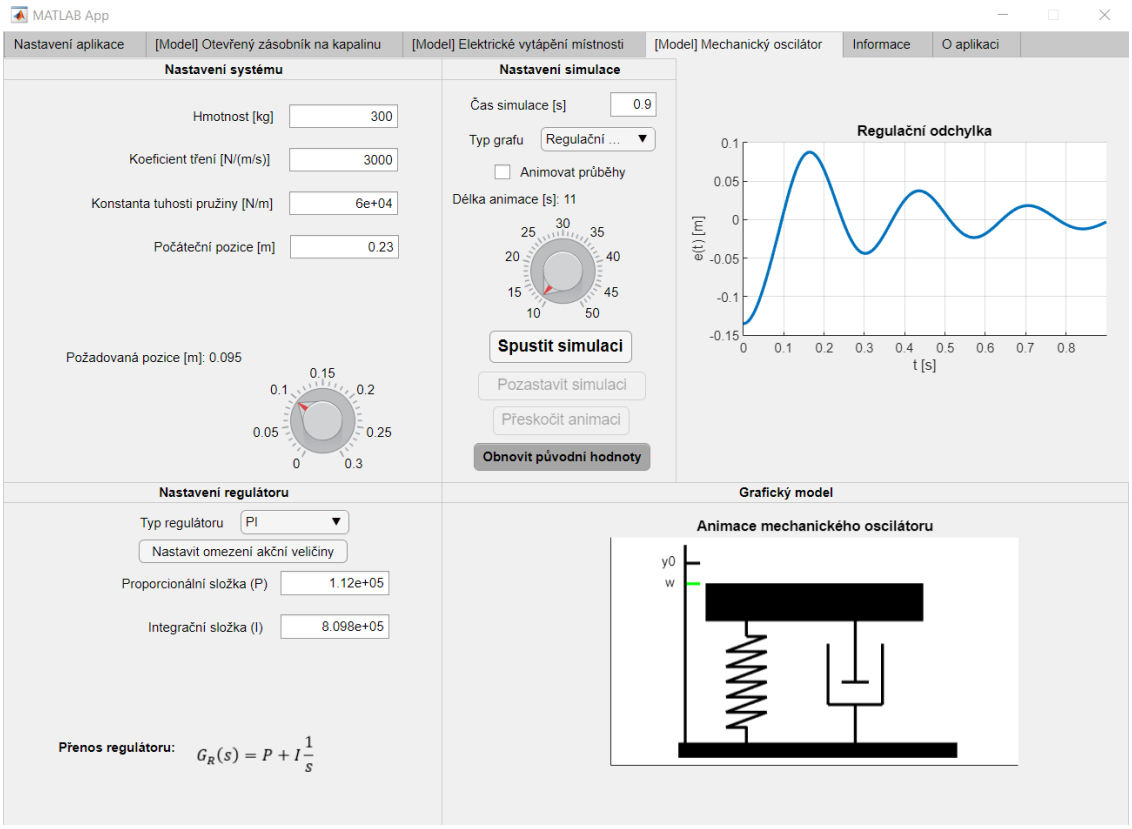
Vstupní parametry	Typ omezení	Hodnota omezení
Hmotnost	>	0
Počáteční pozice	>=	0
Počáteční pozice	<=	0.3
Čas simulace	>	0
Konstantní síla regulace	>=	0
Hodnota regulační odchylky pro zapnutí síly	>=	0
Hodnota regulační odchylky pro zapnutí síly	>=	Hodnota regulační odchylky pro vypnutí síly
Dolní limit akční veličiny	>=	0
Dolní limit akční veličiny	<=	Horní limit akční veličiny

Tabulka 10 - Podmínky pro zadané vstupní hodnoty u modelu mechanického oscilátoru

Na Obrázku 43 a 44 je vidět simulace včetně animace pro model mechanického oscilátoru. Rozdíl mezi těmito simulacemi je ve zvoleném typu regulátoru a v požadovaném typu grafu.



Obrázek 43 - Ukázka spuštěné aplikace – záložka modelu mechanického oscilátoru I

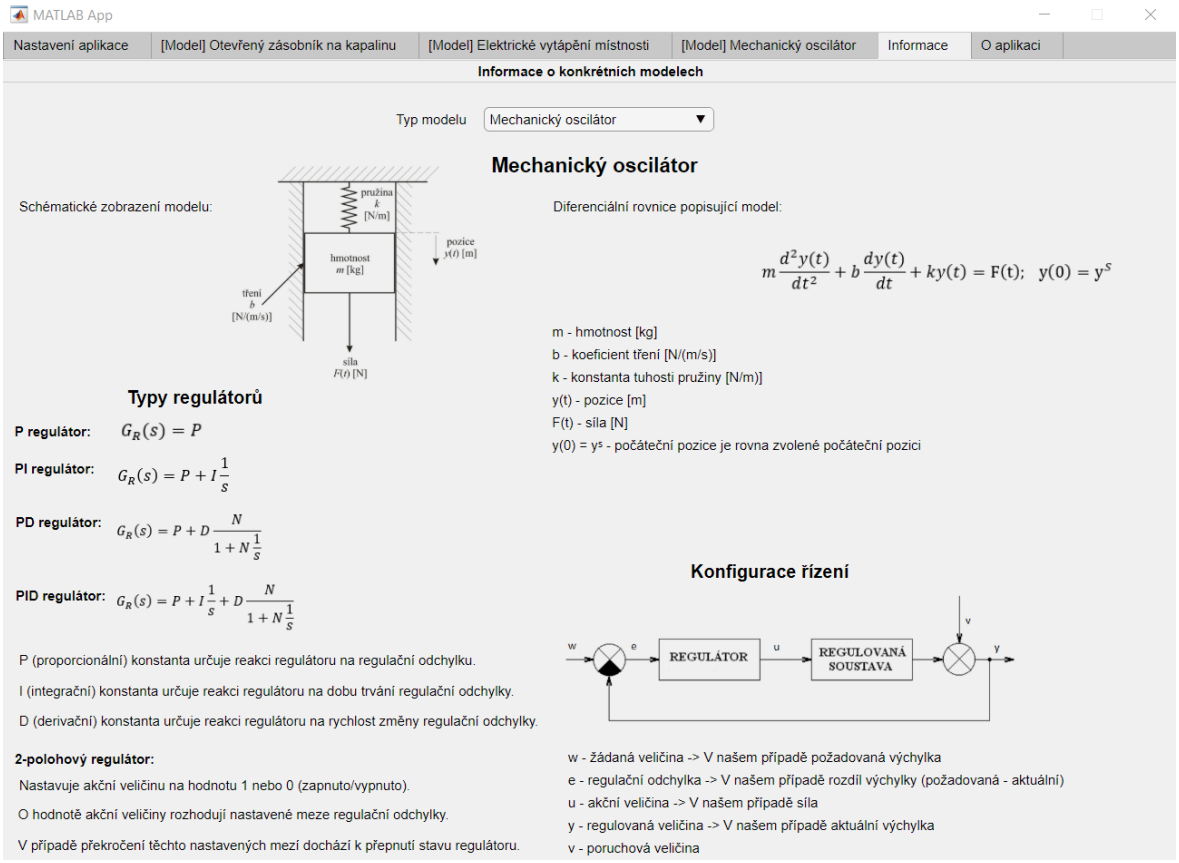


Obrázek 44 - Ukázka spuštěné aplikace – záložka modelu mechanického oscilátoru II

5.7 Záložka informace

Předposlední záložka je nazvaná informace. Zde si uživatel může zvolit některý ze tří modelů a zobrazit si k němu podstatné informace.

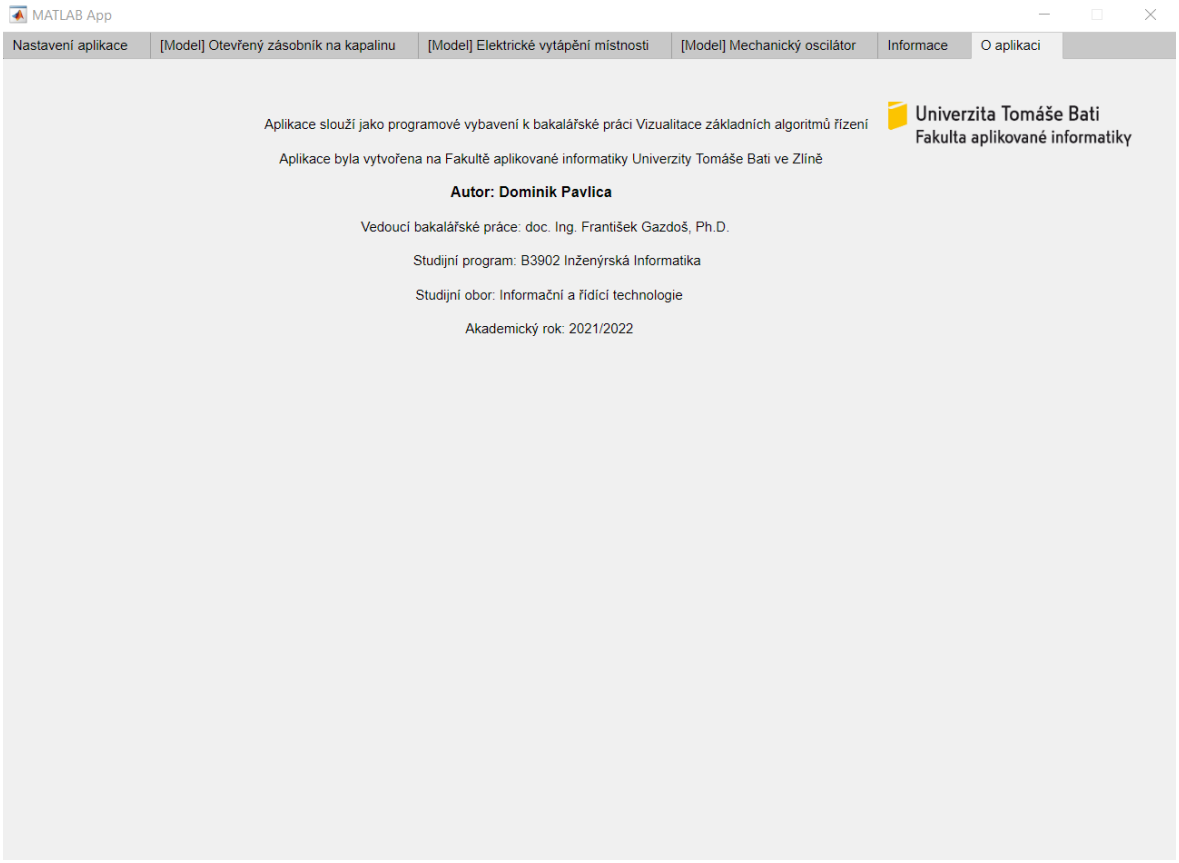
Každý model obsahuje schématický obrázek, diferenciální rovnici popisující zvolený model, popis veličin z dané rovnice, informace o zvolené konfiguraci řízení a informace o každém dostupném regulátoru viz. Obrázek 45 níže.



Obrázek 45 - Ukázka spuštěné aplikace – záložka informace

5.8 Záložka o aplikaci

Poslední záložka nese název o aplikaci a symbolicky uvádí informace o autorovi práce a vedoucím práce, jak je patrné z Obrázku 46 níže.



Obrázek 46 - Ukázka spuštěné aplikace – záložka o aplikaci

ZÁVĚR

Cílem této bakalářské práce bylo na vhodných dynamických systémech demonstrovat činnost základních algoritmů řízení. Vybrané systémy bylo třeba implementovat do simulačního prostředí MATLAB/Simulink a výsledky simulace vhodně zakomponovat do grafického uživatelského rozhraní (GUI), včetně možnosti nastavovat parametry regulátoru a zvoleného systému. V grafickém uživatelském rozhraní (GUI) bylo třeba také navrhnout a implementovat vhodnou animaci pro konkrétní systém, na které bude moci uživatel pozorovat vliv vybraného nastavení a typu regulátoru.

Teoretická část práce obsahuje stručný popis práce s prostředím MATLAB, jeho programovým vybavením Simulink, Live script a App designer. Dále následuje stručný popis použitých regulátorů a teoretickou část uzavírá kapitola popisující vybrané dynamické systémy do této práce.

Praktická část práce se zaměřuje na popis vlastní realizace simulačních schémat zvolených dynamických systémů v prostředí Simulink, včetně popisu použitých konkrétních bloků pro sestavení potřebného simulačního schématu. Druhá část popisuje tvorbu vlastního grafického uživatelského rozhraní (GUI) včetně všech použitých objektů, vysvětlení implementovaných funkcí a představení finální podoby aplikace pro jednotlivé záložky.

Aplikace je určena především pro studenty, ale i pro jiné osoby, které se zajímají o regulaci pomocí (P, PI, PD, PID nebo 2-polohového) regulátoru. Uživatelé této aplikace si mohou vyzkoušet zvolit různé parametry ať už systému, regulátoru nebo simulace a pozorovat dopad těchto změn na regulované veličině, akčním zásahu nebo regulační odchylce, a to pro všechny systémy, které jsem do této práce zakomponoval, přičemž mohou výstupy názorně sledovat i formou animací. Tato práce je rovněž koncipovaná jako otevřená, tedy není problém do budoucna přidat i další zajímavé dynamické systémy, příp. rozšířit ji do jiného jazyka.

Celá tato práce může být dále užitečná nejen pro osoby zajímající se o automatizaci, ale i pro osoby které by chtěli začít používat MATLAB, Simulink, Live script nebo App designer.

SEZNAM POUŽITÉ LITERATURY

[1] HANUŠ, Tomáš. *Realizace a analýza regulačních obvodů v MATLAB/Simulink* [online]. Brno, 2013 [cit. 2022-05-9]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=66184. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky.

[2] BRADÁČ, Martin. *Konstrukce a výroba výukového modelu "Inverzní kyvadlo"* [online]. Brno, 2009 [cit. 2022-05-09]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=16404. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky.

[3] MÍŠENSKÝ, Milan. *Vytvoření uživatelského rozhraní pro soustavu GUNT* [online]. Pardubice, 2017 [cit. 2022-05-09]. Dostupné z: https://dk.upce.cz/bitstream/handle/10195/68112/MisenskyM_VytvoreniUzivatelskeho_DH_2017.pdf?sequence=1&isAllowed=y. Bakalářská práce. Univerzita Pardubice, Fakulta elektrotechniky a informatiky.

[4] MATLAB. In: *Humusoft* [online]. c 1991-2022 [cit. 2022-05-09]. Dostupné z: https://www.humusoft.cz/matlab/details/

[5] PEDAMKAR, Priya. MATLAB Version. In: *EDUCBA* [online]. 2022 [cit. 2022-05-09]. Dostupné z: https://www.educba.com/matlab-version/

[6] PERŮTKA, Karel. *MATLAB: Základy pro studenty automatizace a informačních technologií*. Zlín: Ústav řízení procesů, Institut řízení procesů a aplikované informatiky, Fakulta technologická, Univerzita Tomáše Bati ve Zlíně, 2005. ISBN 80-7318-355-2.

[7] Live Code File Format (.mlx). In: *MathWorks* [online]. c 1994-2022 [cit. 2022-05-09]. Dostupné z: https://www.mathworks.com/help/matlab/matlab_prog/live-script-file-format.html

[8] Comparing GUIDE and App Designer. In: *MathWorks* [online]. c 1994-2022 [cit. 2022-05-12]. Dostupné z: https://www.mathworks.com/products/matlab/app-designer/comparing-guide-and-app-designer.html

[9] ZAPLATÍLEK, Karel. *MATLAB: grafické uživatelské rozhraní*. Brno: Tribun EU, 2020. ISBN 978-80-263-1606-0.

[10] Simscape. In: *Humusoft* [online]. c 1991-2022 [cit. 2022-05-09]. Dostupné z: https://www.humusoft.cz/matlab/simscape/

[11] BALÁTĚ, Jaroslav. *Automatické řízení*. Praha: BEN - technická literatura, 2003. ISBN 80-7300-020-2.

[12] VALTER, Jaroslav. Plynulá regulace PID. In: *Regulace od Jardy* [online]. 2006 [cit. 2022-05-11]. Dostupné z: https://valter.byl.cz/plynula-regulace-pid

[13] GAZDOŠ, František. *Řízení technologických procesů*. Zlín, 2018. Výuková prezentace na FAI UTB.

[14] NOSKIEVIČ, Petr. *Modelování a identifikace systémů*. Ostrava: Montanex, 1999. ISBN 80-7225-030-2.

[15] Create apps with graphical user interfaces in MATLAB. In: *MathWorks* [online]. c 1994-2022 [cit. 2022-05-12]. Dostupné z: https://www.mathworks.com/discovery/matlab-gui.html

[16] MATLAB & SIMULINK. In: *Humusoft* [online]. c 1991-2022 [cit. 2022-05-12]. Dostupné z: https://www.humusoft.cz/matlab/simulink/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MATLAB	MATrix LABoratory
GUI	Graphical User Interface – Grafické uživatelské rozhraní
P	Proporcionální
PI	Proporcionálně integrační
PD	Proporcionálně derivační
PID	Proporcionálně integračně derivační

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Přiložené CD obsahuje:

- Adresář **Vizualizace_zakladnich_algoritmu_rizeni**
  - Soubor **BP_Pavlica_GUI_final.mlapp** – hlavní aplikace
  - Adresář **Obrazky**
    - Soubor **Obecny_zpetnovazebny_obvod.png**
    - Soubor **Utb_fai_logo.png**
    - Adresář **Prenos_regulatoru**
      - Soubor **P.png**
      - Soubor **PD.png**
      - Soubor **PI.png**
      - Soubor **PID.png**
    - Adresář **Model_otevreneho_zasobniku**
      - Soubor **Model_nadrze_matematicka_rovnice.png**
      - Soubor **Model_nadrze_schema.png**
    - Adresář **Model_mechanickeho_oscilatoru**
      - Soubor **Model_mechanickeho_oscilatoru_rovnice.png**
      - Soubor **Model_mechanickeho_oscilatoru_schema.png**
    - Adresář **Model_elektrickeho_vytapeni**
      - Soubor **Model_elektrickeho_vytapeni_matematicka_rovnice.png**
      - Soubor **Model_elektrickeho_vytapeni_mistnosti-nosti_schema.png**
      - Soubor **horko.gif**
      - Soubor **vporadku.gif**
      - Soubor **zima.gif**
  - Adresář **Simulink**
    - Soubor **BP_Pavlica_Simulink_2Polohovy_model_elektrickeho_vytapeni.slx**
    - Soubor **BP_Pavlica_Simulink_2Polohovy_model_mechanickeho_oscilatoru.slx**

- Soubor **BP_Pavlica_Simulink_2Polohovy_model_otevreneho_zasobniku.slx**
- Soubor **BP_Pavlica_Simulink_PID_model_elektrickeho_vytapeni.slx**
- Soubor **BP_Pavlica_Simulink_PID_model_mechanickeho_oscilatoru.slx**
- Soubor **BP_Pavlica_Simulink_PID_model_otevreneho_zasobniku.slx**

- Soubor **fulltext** – text bakalářské práce ve formátu PDF/A