ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ
V PRAZE

Ústav přístrojové a řídicí techniky

FAKULTA
STROJNÍ
ČVUT V PRAZE

DIPLOMOVÁ PRÁCE

Úlohy pro virtuální laboratoř v JavaScriptu

Bc. Martin Hubáček 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hubáček** Jméno: **Martin** Osobní číslo: **475023**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Automatizační a přístrojová technika**

Specializace: **Automatizace a průmyslová informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Úlohy pro virtuální laboratoř v JavaScriptu

Název diplomové práce anglicky:

Modules for virtual laboratory in JavaScript

Pokyny pro vypracování:

- Popište stávající úlohy ve virtuální laboratoři vlab.fs.cvut.cz
- Popište výhody a nedostatky stávajícího řešení virtuálních úloh a řešení v JavaScriptu
- Popište možnosti JavaScriptového prvku Canvas a grafického formátu SVG a vyberte vhodnější z nich
- Vytvořte ekvivalentní úlohy v jazyce JavaScript, které nahradí původní úlohy

Seznam doporučené literatury:

[1] Flanagan, David. JavaScript: The Definitive Guide. O'Reilly Media. 6th Edition. Sebastopol, CA: O'Reilly Media, 2011. 1098 s. ISBN 978-0-596-80552-4
[2] Sharma, Kumar Chetan; Stefanov, Stoyan; Antani, Ved; Hillar, Gastón Carlos. JavaScript: Object Oriented Programming. Birmingham: Packt Publishing, 2016. 824 s. ISBN 978-1-78712-359-5

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Stanislav Vrána U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **29.04.2022** Termín odevzdání diplomové práce: **09.06.2022**

Platnost zadání diplomové práce: _____

Ing. Stanislav Vrána podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	doc. Ing. Miroslav Španiel, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

3.5.2022
Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem svou diplomovou práci na téma *Úlohy pro virtuální laboratoř v JavaScriptu* vypracoval samostatně a použil jsem podklady a literaturu uvedenou v přiloženém seznamu použitých zdrojů.

V Praze dne 7. června 2022 Podpis autora: …….

Bc. Martin Hubáček

Poděkování

Děkuji za podporu a ochotu svým rodičům a přátelům. Ing. Stanislavu Vránovi DiS, Ph.D. děkuji za odborné vedení bakalářské práce, trpělivost a cenné připomínky. Děkuji také prof. Ing. Milanovi Hofreiterovi, CSc. za přínosné konzultace.

Anotace

V diplomové práci je popsán postup přetvoření a aktualizace virtuálních úloh z fakultního webu http://vlab.fs.cvut.cz/. Původní úlohy jsou psané v jazyce Java a spouštěny v samostatných appletech, které již nemají podporu v moderních internetových prohlížečích. Z důvodu modernizace a uživatelské dostupnosti se tyto úlohy transformovaly pomocí jazyka JavaScript. Nová podoba je kompatibilní se současnými prohlížeči a nevyžaduje žádné další uživatelské zásahy, jako nastavování firewallů apod. Důraz byl kladen zejména na zachování všech funkcí původních úloh a jejich vzhledu. I přes tuto snahu se nová virtuální laboratoř v mnoha ohledech od původní odlišuje.

Klíčová slova: JavaScript, virtuální úlohy, řízení, regulace

Annotation

The diploma thesis describes the process of transforming and updating virtual tasks from the faculty website http://vlab.fs.cvut.cz/. The original tasks are written in Java and run in separate applets that are no longer supported in modern internet browsers. Due to modernization and user availability, these tasks have been transformed using JavaScript. The new look is compatible with current browsers and does not require any additional user intervention, such as setting up firewalls, etc. The emphasis was mainly on maintaining all the functions of the original tasks and their appearance. Despite this effort, the new version differs from the original in many respects.

Keywords:

JavaScript, virtual models, control, regulation

Obsah

Úvod ... 8
1 JavaScript obecně ... 9
1.1 Objektové orientované programování ... 9
1.2 Řízení událostmi ... 9
1.2.1 Charakteristika událostí ... 10
1.2.2 Kategorie událostí ... 11
1.3 Porovnání jazyka Java a JavaScript ... 12
1.3.1 Jazyky programovací ... 12
1.3.2 Jazyky skriptovací ... 12
2 Jazyk HTML a CSS ... 14
2.1 Základní prvky HTML ... 14
2.2 Využití CSS ... 15
2.3 Rozložení stránky ... 16
3 Propojení HTML, CSS a JavaScriptu na webu ... 18
3.1 Objektový model dokumentu (DOM) ... 18
3.2 Vizualizace – element canvas a SVG objekty ... 20
4 Řízení a regulátory ... 24
4.1 Používané veličiny ... 24
4.2 Regulátory ... 24
4.2.1 Regulátory typu P-I-D ... 25
4.2.2 Dvoupolohový a třípolohový regulátor ... 25
5 Původní virtuální laboratoř ... 27
6 Nová virtuální laboratoř ... 28
6.1 Struktura programu ... 28
6.2 Konkrétní skripty a jejich funkce ... 28
6.2.1 Hlavní systémové skripty ... 28
6.2.2 Skripty jednotlivých úloh ... 38
6.2.3 Použité knihovny ... 40
6.3 Rozložení stránky ... 41
6.3.1 Výstupní grafy ... 42

6.3.2 Interaktivní graf 43
6.3.3 Nastavení jazyků a rozšířené nastavení 43
6.3.4 Regulační část 45
6.3.5 Vizualizace a systémové parametry 45
6.3.6 Hlavní ovládání 46
6.3.7 Přednastavené funkce 47
6.4 Chod programu 49
6.5 Příklad využití konkrétní úlohy 50
7 Závěr 52
Použité zdroje 53
Seznam obrázků 54
Seznam výpisů zdrojových kódů 55

Úvod

Pro výuku předmětu Automatické řízení byly již před více jak deseti lety připraveny virtuální úlohy, které měly za cíl zvýšit zájem studentů a zároveň zpestřit výuku tohoto předmětu. Původní virtuální laboratoř byla napsána v jazyce Java a vizualizována v internetovém prohlížeči za použití Java appletu. Během roku 2013 začala většina nejpoužívanějších internetových prohlížečů ukončovat podporu těchto appletů a mezi roky 2015 a 2017 už nešlo tyto applety ani spouštět, s výjimkou prohlížeče Internet Explorer. Z tohoto důvodu, a také kvůli možnostem jazyka JavaScript, který je vhodný pro dynamické vizualizace, bylo rozhodnuto o přetvoření původních úloh do nové podoby právě v JavaScriptu. Nová virtuální laboratoř je vizualizována přímo na webu pomocí jazyka HTML ve spojení s kaskádovými styly, které prvkům HTML přiřazují vzhledové vlastnosti. JavaScript zajišťuje dynamické propojení všech prvků.

Virtuální laboratoř má v současné podobě 6 diskrétních a 10 spojitých úloh. Jedna spojitá úloha (Wattův roztěžník) je zároveň ovladatelná vzdáleně. Cílem diplomové práce je vytvořit novou verzi pouze spojitých úloh. Měla by však vytvořit základ pro budoucí tvorbu diskrétních úloh.

Diplomová práce je standardně členěna do dvou hlavních částí. V první části jsou shromážděny teoretické poznatky k jazyku JavaScript a vysvětlena vhodnost jeho využití jako náhrady za původní jazyk Java. Druhá část popisuje novou virtuální laboratoř a postup práce při její tvorbě.

1 JavaScript obecně

JavaScript je multiplatformní, objektově orientovaný, událostmi řízený skriptovací jazyk. Jeho syntaxe (zápis zdrojového textu) patří do rodiny jazyků C/C++/Java, ale JavaScript je od těchto jazyků zásadně odlišný sémanticky (funkčně, principiálně), jde o jiný jazyk. Slovo Java je součástí jeho názvu pouze z marketingových důvodů. [1]

1.1 Objektové orientované programování

Objektově orientované programování (OOP) je široce rozšířený princip psaní programu. Využívá se u mnoha moderních programovacích jazyků, např. C#, C++, Python, JavaScript, ale i u jazyka Java, použitého v původní virtuální laboratoři. Popisuje nejen zápis programu, ale i způsob, jakým programátor při navrhování přemýšlí. Objektově orientované programování vychází z principů, se kterými se setkáváme u objektů v běžném životě, a následně je aplikuje v programu. Reálné objekty mají mnoho znaků a funkcí společných a na tomto principu jsou založeny i objekty v OOP. [1]

S objekty úzce souvisí pojem třída. Obsahuje množinu vlastností a funkcí shodných pro konkrétní typ objektu. Třída může být také součástí jiné třídy, jiné třídy z ní mohou být odvozené na základě tzv. dědičnosti a je šablonou pro jednotlivé instance. Instance je aplikace třídy s konkrétními daty a tímto způsobem lze vytvořit mnoho rozdílných objektů, avšak se stejnou strukturou, což zajišťuje zejména lepší orientaci v programu. [1]

1.2 Řízení událostmi

Programy v JavaScriptu používají asynchronní programovací model založený zejména na událostech. Program vygeneruje událost kdykoliv se něco zajímavého stane v dokumentu, na webu nebo některém z objektů s ním spojeným. Příkladem takové události může být načtení souboru, kliknutí na tlačítko nebo i pouhý pohyb myši. Mnoho událostí je generováno přímo prohlížečem. Samotné vygenerování události ale ještě nemusí znamenat vůbec nic. Aplikace na událost zareaguje pouze v případě, že je někde ke konkrétní události v programu přiřazena tzv. obslužná metoda (event handler). Objekt vyvolávající událost běžně nemá zpětnou vazbu, zda byla událost vůbec nějakým způsobem zpracována. Tento styl programování není nijak výjimečný a všechny aplikace s grafickým rozhraním ho v určité míře používají také. [1]

1.2.1 Charakteristika událostí

V souvislosti s událostmi se rozlišuje pět základních pojmů:

- typ události,
- cíl události,
- obslužná metoda,
- objekt události,
- propagace události. [1]

Typ události se často nazývá rovněž jménem události, protože je tvořen pouze řetězcem znaků vyjadřující, jaká událost se stala. Typy událostí jsou například „pohyb myší“, „načtení“, „kliknutí myši“ či „stisk klávesy“. [1]

Cíl události je objekt, u kterého došlo k události, nebo který je s událostí spojen. Událost je definována cílem a typem. Může se jednat o kliknutí myši (typ) na tlačítko (cíl) či načtení (typ) obrázku (objekt). [1]

Obslužné metody jsou aplikací předány prohlížeči, který je následně v případě konkrétní události vyvolává. Počet obslužných metod vázaných na událost není omezen. Některé typy událostí mají v počátečním nastavení určené chování, které však lze potlačit. Konkrétní využití tohoto potlačení je popsáno v kapitole 6.3.7. [1]

Objekt události je spojen s konkrétní událostí a obsahuje detailní data o události. Běžně se předává jako parametr obslužné metodě, ale v mnoha případech to není potřeba. Každý typ události má zároveň svoje specifické parametry, např. kliknutí myši v objektu události obsahuje souřadnice místa vzniku. [1]

Propagace události je proces, kde prohlížeč rozhoduje, na kterých objektech vyvolat událost. Pokud má událost přiřazen konkrétní objekt, s kterým je svázána, není žádné propagace potřeba. V opačném případě však událost postupuje pomyslnou dokumentovou strukturou webu. Obslužná metoda může tuto propagaci zastavit a tím nedojde k vyvolání události na dalších nadřazených prvcích. Prvky webu jsou podrobněji popsány v kapitole č. 2. S propagací rovněž souvisí tzv. zachycení událostí, které v dřívějších verzích JavaScriptu nebylo podporováno a není z tohoto důvodu zatím příliš rozšířené. Obslužné metody spjaté s objektem, na kterém dochází k vyvolání události, mají možnost zastavit šíření události a zamezit tím vyvolání obslužné metody, která by v běžném případě byla běžně vyvolána na

základě vzniklé události. Tohoto zachycení je rovněž využito v praktické části při práci s body a jejich posouváním. [1]

1.2.2 Kategorie událostí

Události lze rozdělit do 4 základních kategorií:

- závislé na zařízení,
- nezávislé na zařízení,
- spojené s rozhraním,
- reagující na změnu stavu. [1, 2]

Události závislé na zařízení jsou přímo spojeny s určitým vstupním zařízením. Tyto zařízení nejčastěji zastupují klávesnice a myš. Mezi tyto události patří například stisknutí klávesy, uvolnění klávesy, pohyb myši a kliknutí myši. [1, 2]

Události nezávislé na zařízení obsahují obecnější události, které mohou být vyvolány více zařízeními a způsoby. Nejčastěji využívaná událost tohoto typu je událost kliknutí, které sice může být vyvolána kliknutím myši, ale zároveň i dotykem na dotykové obrazovce. [1, 2]

V aplikacích je často žádoucí, aby se událost vykonávala při změně hodnoty nebo když uživatel aktivuje konkrétní prvek. Tyto události se váží ke konkrétnímu rozhraní a navazují se na prvky vyšší úrovně, nejčastěji na vstupní pole pro uživatele. Mohou tak při každé změně hodnoty, například přidání či ubrání písmena ve vkládaném slovu, znovu zavolat obslužnou metodu, která z databáze vyfiltruje předměty odpovídají řetězci znaků v právě změněném vstupním poli. [1, 2]

Některé události neaktivuje sám uživatel, ale může je vyvolat změna stavu v prohlížeči nebo v síti. Mnoho událostí probíhá v pozadí a uživatel si jich ani nemusí být vědom. Tyto události bývají nejčastěji spojeny s ovládáním webového prohlížeče a vyvolávají je nejen tlačítka zpět či aktualizovat, ale i načtení celé webové stránky. Prohlížeč běžně obsahuje dvě sady události pro funkce offline a online. [1, 2]

Mezi základními událostmi by bylo vhodné uvést i časovače a obsluhu chyb, které jsou součástí asynchronního programovacího modelu JavaScriptu. Nejedná se však přímo o události, ale je užitečné je mít spojeny právě s událostmi, protože se v mnohém chovají velice podobně. Využívají se dva základní časovače a to *setTimeout* a *setInterval*. Tyto

metody jsou definovány jako globální funkce objektu *Window*, což je hlavní objekt aplikace. Funkce *setTimeout* zavolá určitý blok programu po nastaveném času v milisekundách. Funkce *setInterval* se chová jako opakovaně volaná funkce *setTimeout*. [3]

1.3 Porovnání jazyka Java a JavaScript

Java i JavaScript jsou objektově orientované jazyky. Java je jazyk programovací a JavaScript jazyk skriptovací. Ačkoliv by se mohlo zdát, že jde o prakticky stejné věci, tak se naopak v mnoha ohledech diametrální odlišují. [4]

1.3.1 Jazyky programovací

Hlavní odlišností od skriptovacích jazyků je nutnost kompilace kódu do strojového. Programovací jazyk se v podstatě používá k transformaci dat. Ze zdrojového kódu se při kompilaci vytvoří binární instrukce, se kterými následně pracuje procesor konkrétního zařízení. Zástupci programovacích jazyků jsou C, C++ a Java. Tyto jazyky obecně tvoří základ aplikace. Velice účelné je však propojit jádro aplikace psané v těchto jazycích s určitými prvky jazyků skriptovacích a využít tak výhod obou skupin. [4]

Jazyk Java je stále hojně využíván v různých aplikacích. Kód se spouští v takzvaném „Java virtual machine“, který vykonává bytový kód a překládá jej do nativních procesorových instrukcí podle konkrétního procesoru. Samotný bytový kód je na hardwaru nezávislý. Důvod rozboru jazyka Java v této práci jsou tzv. Javové applety. Tyto applety jsou malé aplikace, které rozšiřují možnosti webu a většinou se spouštějí v samostatném okně. Dřívější applety byly často mnohokrát rychlejší než aplikace v JavaScriptu, ale po čase se projevily zásadní nedostatky těchto appletů. I z toho důvodu se postupně ukončila jejich podpora ve všech rozšířenějších webových prohlížečích. [5]

1.3.2 Jazyky skriptovací

Aplikace, webové stránky v prohlížeči a interpret příkazů (Shell) operačního systému mohou být modifikovány pomocí skriptovacích jazyků. Mezi rozšířené skriptovací jazyky patří JavaScript, Perl a VBScript. Tyto jazyky nepotřebují kompilátor a mají menší přístup k funkcím operačního systému počítače, protože fungují jako nadstavba programovacích

jazyků. Dnešní prohlížeče si i přesto například JavaScript kompilují, avšak nekompilují jej vždy, ale pouze ty části, jejichž spouštění se opakuje. [4]

JavaScript se ve většině případů vyskytuje ve spojení s HTML souborem a vkládá do něj dynamické chování. Může se používat jako objektově orientovaný i jako procedurální jazyk. Na rozdíl od Java appletů má mnohem širší spektrum možností dynamizace webu. Patří mezi nejpoužívanější jazyky na světě, a proto lze nalézt mnoho externích knihoven s předpřipravenými funkcemi i kvalitní dokumentací. S těmito možnostmi se zároveň pojí nutnost většího rozhodování a zvažování vhodnosti použití konkrétní knihovny. Zásadním rozhodovacím faktorem bývá aktuálnost a rozšířenost. [6]

2 Jazyk HTML a CSS

Běžná webová stránka je tvořena kódem HTML, zajišťující základní strukturu, doplněném o část v jazyce CSS (Cascading style sheet), která upravuje vzhled jednotlivých HTML objektů. Většina stránek také obsahuje externí obsah jako obrázky, videa, animace či zvukové záznamy. Tyto externí prvky jsou vnořeny do prvků HTML k tomu určených. [7]

Prvky se v HTML zapisují počáteční a koncovou značkou (tag). Tagy se chovají jako obálky a zapouzdřují veškerý obsah mezi nimi. Příklady tagů mohou být počáteční tag a koncový tag . Koncový tag se od počátečního odlišuje pouze znakem „/“. [7]

2.1 Základní prvky HTML

Jazyk HTML obsahuje mnoho prvků, ale existuje několik z nich, které se vyskytují v téměř každé aplikaci:

- <html> - nejvyšší tag zapouzdřující celou stránku,
- <head> - obsahuje údaje o stránce, které nebývají vidět v hlavní části,
- <body> - hlavní tag, ve kterém se nachází většina prvků,
- - tag umožňující vytvořit podskupinu tagů, na rozdíl od předchozích není jedinečný,
- - tag s podobnými vlastnostmi jako div používaný většinou pouze na jeden řádek,
- <script> - reprezentace nahraného skriptu v html souboru, obsahuje adresu umístění skriptu,
- <button> - prvek tlačítka,
- <img> - prvek obrázku,
- <input> - vstupně/výstupní okno pro interakci s uživatelem,
- <h1, h2, …, h6> - prvky nadpisů podle úrovně. [7]

Tag může mít atribut, který přidává další informace o vlastnostech a chování prvku. Atribut je tvořen jménem a hodnotou. Různé tagy používají různé atributy, ale některé jsou použitelné všude, např. atribut ID přidělující prvku jméno pro dynamizaci. Atributů existuje veliké množství a vývojová prostředí bývají uzpůsobená tak, aby programátorovi nabízela

pouze ty, které jsou v konkrétní situaci použitelné. Atributy použité v praktické části jsou popsány přímo v použitém kontextu. [7]

Tagy jsou strukturovány hierarchicky a mezi atributy se vyskytují určitě formy dědičnosti. Dědičnost u HTML tagů funguje však jinak než dědičnost u tříd v kontextu objektově orientovaného programování. Pokud je tag uvnitř jiného, označuje se jako potomek a když jsou dva tagy v hierarchii přímo u sebe tak jde o specifický případ označovaný jako dítě. Neznamená to ale, že se automaticky všechny atributy z nadřazeného tagu přebírají se stejnými vlastnostmi. Tato hierarchie je vytvořena zejména pro efektivní adresování a rozdělení dokumentu. [7]

2.2 Využití CSS

CSS umožňuje snadné formátování HTML elementů a tím snadné formátování webových stránek. CSS spojuje styl s elementem HTML. Pravidlo CSS je tvořeno dvěma částmi: selektorem a deklarací. Selektor určuje, jaké elementy toto pravidlo ovlivňuje a deklarace určuje, jak by se zvolené elementy měly stylizovat. Samotná deklarace má stejnou strukturu jako atribut a v konečném důsledku jde o přepsání obecného atributu, který však nemusí být pouze pro jediný prvek. Příkladem CSS může být například: `button {font-family:Arial;}`. V tomto pravidle je selektorem první slovo button, což znamená, že se pravidlo vztahuje na všechna tlačítka v dokumentu. Deklarací se určuje všem tlačítkům použitý styl písma na Arial. Atribut přímo u elementu má větší váhu než obecně psané pravidlo. Při použití tagu `<button style="font-family:Tahoma">` bude toto tlačítko používat styl písma Tahoma, protože je styl určen přímo a takové přidělení má větší prioritu než obecné pravidlo. [7]

Existuje několik druhů selektorů umožňujících lepší volbu a interakci s prvky HTML:

- univerzální,
- typový,
- třídový (class),
- podle ID,
- podle potomka. [7]

Univerzální selektor ovlivní všechny prvky v dokumentu a typový pouze konkrétní typ elementu (img, button). Třídový selektor funguje stejně jako podle ID a bývá zvykem, že

každý prvek má unikátní ID, ale několik prvků může mít stejnou třídu. Selektor podle potomka umožňuje strukturovat vlastnosti pro prvky vnořené do konkrétního prvku. Příkladem může být zápis `span h1 {color: rgb(255,0,0)}` znamenající, že všechny nadpisy první úrovně v prvku span budou mít červenou barvu. Speciálním případem selektoru podle potomka je selektor ovlivňující pouze přímého potomka (child). Konkrétní použití je uvedeno ve výpisu č. 1 a stejný kód je zároveň použit v praktické části pro hlavní strukturu rozložení stránky. [7]

```
.display-table {
            display: table;
        }
            .display-table > div {
                display: table-row;
            }
                .display-table > div > div {
                    display: table-cell;
```

Výpis č. 1: Strukturování oddílu do tabulkového zobrazení

Se selektory souvisí i dědičnost CSS vlastností. Některé vlastnosti se automaticky propisují do vnořených prvků. Většinou se však vlastnosti přímo nedědí, ale lze toho docílit přidáním klíčového slova *inherit* namísto konkrétní hodnoty vlastnosti. [7]

2.3 Rozložení stránky

Při návrhu stránky je v počátku zásadní zvolit rozložení prvků, jejich velikost a s velikostí související reakci na případnou změnu rozlišení zobrazovacího zařízení či okna prohlížeče. Atribut pro pozici má název position. Používají se tři základní pozicovací schémata:

- normální,
- relativní,
- absolutní. [7]

Počáteční nastavení pozicování všech html souborů bývá normální. Prvky se při takovém nastavení řadí pod sebe. Relativní pozicování umožňuje umístění prvků odvozené od jejich pozice v normálním nastavení. Nejčastěji se používá posunutí prvku o určitý počet pixelů do strany. Příkladem syntaxe je CSS pravidlo `{position: relative; top: 10px; left: 100px;}`, značící přidělení relativního pozicování, posunutí o 10 pixelů od horního okraje

a o 100 pixelů od levého okraje ohraničujícího prvku. V této souvislosti je důležitá skutečnost, že atribut pozice se vztahuje vždy k nadřazenému prvku, a nikoliv celé stránce. Lze tak docílit relativního pozicování vnořeného prvku v prvku, který má vzhledem k celé stránce stále pozici v normálním režimu. [7]

Absolutní nastavení pozice zcela vyjme prvek z normálního toku stránky. V tuto chvíli přestane ovlivňovat ostatní prvky, jako kdyby se na stránce vůbec nevyskytoval. Jeho parametry (top, bottom, right, left) v pixelech ovlivňují odsazení od nadřazeného prvku. V případě vyjmutí prvku z normálního toku se mohou prvky překrývat. Vlastnost z-index umožňuje volit, v jakém pořadí se mají prvky zobrazovat na sobě. Čím větší hodnota této vlastnosti je, tím výše se prvek zobrazí. [7]

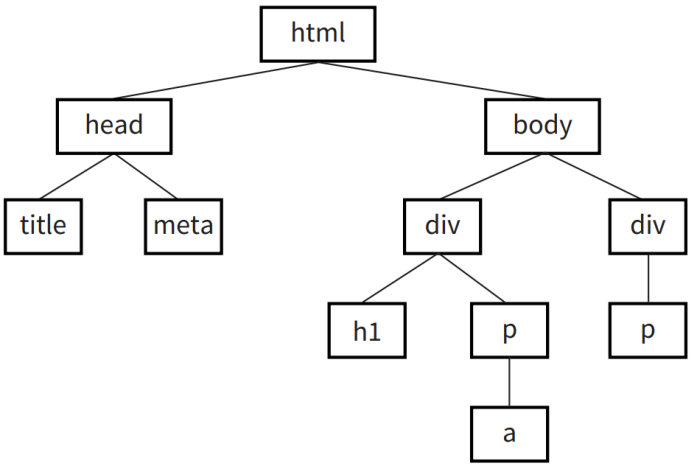
3 Propojení HTML, CSS a JavaScriptu na webu

Předchozí kapitola se věnovala pouze propojení HTML a CSS. Hlavním přínosem jazyka JavaScript je možnost dynamizace HTML objektů a CSS vlastností. Nejčastějšími případy bývají změna barvy, textu či viditelnosti konkrétního prvku, např. tlačítka. [8]

Existují dvě možnosti vložení skriptu do webové stránky. Kód lze vložit přímo mezi tag `<script> Kód v JavaScriptu </script>` nebo jako odkaz na externí soubor s koncovkou .js ve formátu `<script src="kod_v_javscriptu.js"></script>`. Skriptové tagy se ve většině případů umisťují na úplný konec prvku body. V některých aplikacích může být však žádoucí umístit skript na začátek stránky, aby se kód začal vykonávat ještě před načtením celé stránky. [8]

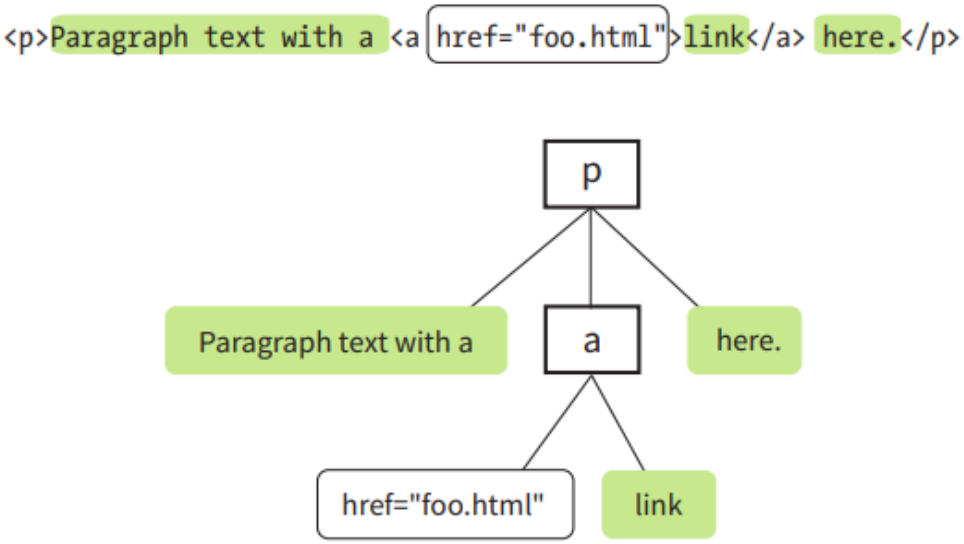
3.1 Objektový model dokumentu (DOM)

DOM poskytuje způsob, jak přistupovat k obsahu a manipulovat s ním. DOM je programovací rozhraní (API) pro stránky HTML. Poskytuje strukturovanou mapu dokumentu a také sadu metod pro rozhraní s prvky v něm obsaženým. Účinně překládá značení do formátu, kterému JavaScript (a další jazyky) rozumí. Je možné ho použít k vyhledání prvků podle jejich názvů nebo atributů a poté k přidání či úpravě vlastností, případně k odstranění prvků. Prvek na stránce se nazývá uzel. DOM má strukturu stromu, každý uzel je samostatná větev, která může obsahovat další větve. DOM však umožňuje hlubší přístup k obsahu než samotné CSS, protože zachází také s obsahem prvku jako s uzlem (viz Obrázek č. 1). [8]



Obrázek č. 1: Struktura z pohledu HTML [8]

Obrázek č. 2 ukazuje strukturu elementu <p>. Každý prvek, jeho atributy a jeho obsah představuje jeden uzel ve stromu uzlů modelu DOM. [8]



Obrázek č. 2: Struktura z pohledu DOM [8]

Objekt s názvem document v DOM identifikuje samotnou stránku a používá se jako výchozí bod pro procházení DOM. Objekt dokumentu obsahuje řadu standardních vlastností a metod pro přístup k prvkům. Samotná adresace konkrétního prvku funguje na základní standardní tečkové notace známé z běžných objektových struktur. Následující příklad ukazuje, jak v modelu DOM označit atribut barvy pozadí prvku s konkrétním ID a změnit tuto barvu na červenou: `document.getElementById("unikatni_ID").style.backgroundColor = „red“;`. Metod pro označení konkrétních prvků existuje mnoho a za zmínku ještě stojí metoda `document.querySelector()`, která vrátí odkaz na první objekt odpovídající selektoru vloženému jako parametr, což může být také ID. [8]

Kromě možnosti ovládat prvky na webové stránce poskytuje JavaScript přístup k částem okna prohlížeče a možnost manipulovat s nimi. Okno prohlížeče obsahuje veškerá data o stránce, a navíc i metody a vlastnosti prohlížeče. [8]

3.2 Vizualizace – element canvas a SVG objekty

Existuje mnoho způsobů, jak vizualizovat určitý obsah na webové stránce. Vedle statického obrázku, elementu pro externí html zdroj (<iframe>) či elementu videa, lze využit také element canvas a objekty SVG, umožňující dynamické vykreslování. [8]

Canvas

Prvek canvas vytváří oblast na webové stránce pro dvojrozměrné vykreslování se sadou funkcí JavaScriptu pro tvorbu čar, tvarů, výplní, textů, animací a tak dále. Lze v něm zobrazit i statický obrázek, ale hlavním potenciálem prvku canvas je, že je jeho obsah generovaný skriptováním. To znamená, že je dynamický a dokáže věci kreslit za běhu a reagovat i na uživatelský vstup. Díky tomu je možné ho použít pro vytváření animací, her i celých aplikací. Objekt plátna se v HTML kódu vytvoří takto: `<canvas width="600" height="400" id="prvni_canvas"></canvas>`. Mezi počátečním a koncovým tagem není nic, protože jde pouze o určení prostoru, kde se bude následně vykreslovat obsah. Plocha pro kreslení plátna je tvořena mřížkou pixelů. Tím se odlišuje od objektů SVG, který používá vektorové tvary a cesty, které jsou definovány pomocí bodů a matematických vztahů. [8]

SVG

Při vykreslení v okně prohlížeče může grafika SVG vypadat jako obrázek v jakémkoli jiném formátu. Jde o vektorový formát, tedy tvary v rámci SVG jsou definovány souřadnicemi a čarami, nikoli mřížkami pixelů. Tato vlastnost zajišťuje škálovatelnost – mohou nekonečně měnit velikost bez ztráty kvality. Příkladem SVG objektu může být `<svg width="150" height="200"> <circle cx="75" cy="75" r="40"/> </svg>`, který by se zobrazil jako kruh se středem na souřadnicích [75, 75] a o poloměru velikosti 40. Zastřešující objekt SVG má konkrétní pozici na stránce a vnitřní element kruhu následně používá relativní souřadnicový systém vzhledem k hranicím prvku SVG. [8]

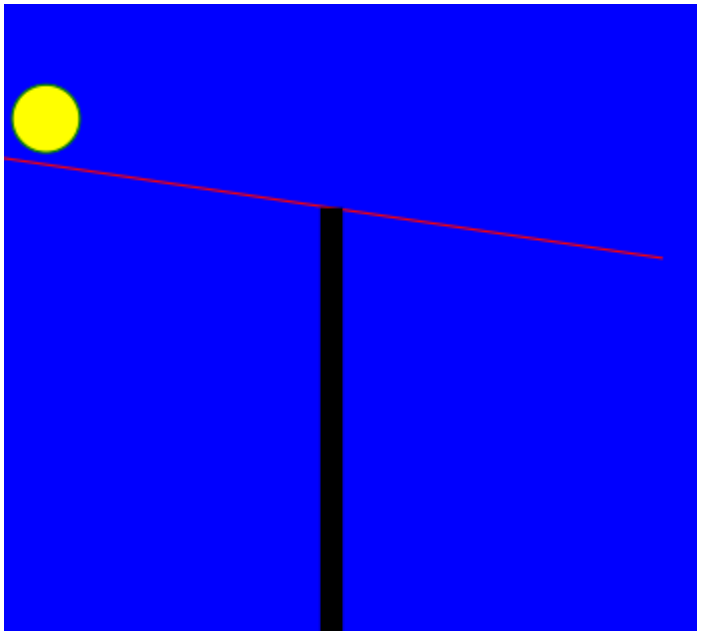
Porovnání elementu canvas a SVG

Obrázek SVG je vytvořen pomocí obecného značkovacího jazyka (XML) a element canvas se vykresluje příkazy JavaScriptu. Jazyk XML není přímo jazyk, ale spíše rozsáhlý soubor pravidel pro tvorbu jiných značkovacích jazyků. Oba elementy mohou obsahovat obrázky, videa, animace a dynamicky se aktualizovat v reálném čase. Canvas je lepší pro rychlé překreslování za běhu, takže se lépe hodí pro navrhování her a animace. SVG nabízí výhody

jednoduchostí skriptování a animací. Komplikované objekty SVG však vyžadují více výpočetního výkonu než canvas element. [8]

Při rozhodování o vhodnosti využití prvku canvas a svg bylo nutno brát v potaz dva faktory. Nejdůležitějším z nich je proveditelnost, tedy zjistit, zda je možné vizualizovat všechny úlohy pomocí jednoho prvku. Po hlubším zkoumání jsem došel k závěru, že lze každou úlohu vizualizovat jak pomocí SVG, tak i prvku canvas, ale s velice odlišnou náročností. S náročností souvisí i druhý faktor, a to možnost využití původního kódu, který se velice podobal struktuře kódu pro element canvas.

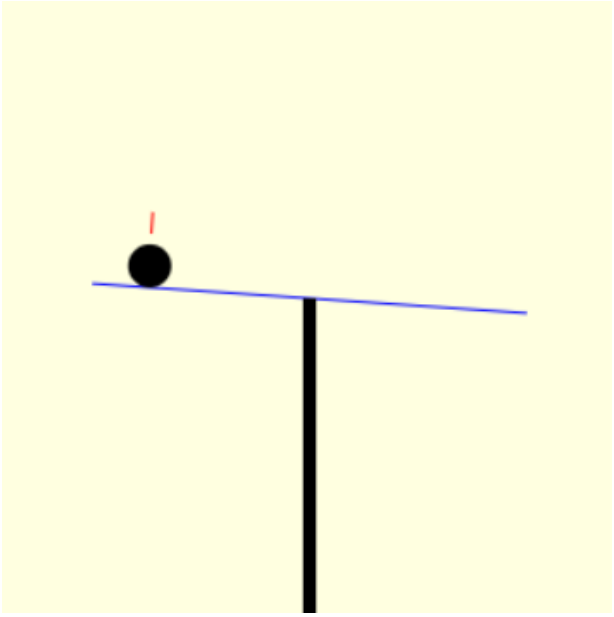
Pro vysvětlení konečného rozhodnutí jsem vybral dva příklady: úlohu s kuličkou na tyči a s vodní nádrží. V obou případech se canvas projevil jako vhodnější prvek. Při animování při využití SVG dochází ke špatnému pohybu kuličky po podstavě. Kulička se v některých chvílích od podstavy zcela odpoutá a směřuje k cílové pozici po jiné trajektorii, než by ve skutečnosti měla (viz Obrázek č. 3).



Obrázek č. 3: Vizualizace pomocí SVG

Prvek canvas se překresluje podle aktuálních hodnot v každém průběhu cyklu a kulička je proto vždy ve správné poloze (viz Obrázek č. 4), ale zároveň je pozorovatelná mírná nerovnoměrnost v průběhu, protože cyklus trvá téměř sto milisekund a lidské oko je při takové frekvenci schopné vnímat obnovování obrazu. Obrázky mají odlišné barvy z důvodu zvýšení kontrastu. Animace pomocí SVG je celkově plynulejší, což je zajištěno chováním

externí knihovny pro animaci těchto prvků. U kuličky na tyči je tato skutečnost dobře pozorovatelná, protože jde o rychle se pohybující systém.



Obrázek č. 4: Vizualizace v elementu canvas

Kód pro animování pomocí SVG se nachází ve výpisu č. 2. Pomocí externí knihovny anime se na základě vložených parametrů natáčí podstava a kulička posouvá. Aby se podstava neotáčela kolem krajního bodu je nutné posunout střed rotace do středu podstavy funkcí *transformOrigin*.

```
const podstava = document.querySelector("#podstava");
const kulicka = document.querySelector("#kulicka");

function aktualizovatPodstavu(uhel) {

    anime({
        targets: podstava,
        rotate: uhel,
        transformOrigin: ['50% 30%', '50% 30%']
    });
}
function aktualizovatKulicku(x, y) {

    anime({
        targets: kulicka,
        translateX: x - 250,
        translateY: y - 84,
    });
}
```

Výpis č. 2: Animování SVG

Úloha s vodní nádrží je vizualizována statickým obrázkem a obdélníkem znázorňujícím vodu v nádrži. S využitím prvku canvas se volá pouze jedna metoda kódu vykreslující obdélník a to: `c2.fillRect(112, 232 - ypos, 84, ypos);`. V případě použití SVG je kód složitější, protože se upravují vlastnosti již vytvořeného obdélníku. Pokud by bylo možné přímo měnit rozměry obdélníku, tak by toto řešení bylo srovnatelné s prvkem canvas, ale pro změnu rozměru u SVG je nutné použít metodu *scale*, který objekt zvětší či zmenší v daném měřítku. Metoda *scale* přijímá dva parametry. První parametr je osa (x, y), v jakém směru se má změna rozměru vykonat a druhý parametr je číselná hodnota udávající násobek původní velikosti. Se změnou velikosti rovněž souvisí středový bod, vůči kterému se úprava rozměrů vztahuje. Pro konkrétní příklad s nádrží je nutné nejdříve posunout počáteční bod na spodní okraj obdélníku, zavolat metodu *scale* s parametrem osy y, aby se obdélník nezúžil či nerozšířil, a následně vypočítat jaký násobek je potřeba vložit jako číselný parametr. Tuto hodnotu lze vztahovat k referenční hodnotě, např. maximálnímu rozsahu obdélníku, ale i v takovém případě se jedná o velice složité řešení v porovnání s jedním řádkem kódu při využití elementu canvas. I z tohoto důvodu jsem se rozhodl ve všech úlohách využít pouze element canvas.

4 Řízení a regulátory

Pojem řízení představuje působení na konkrétní objekt tak, aby vykonával požadovanou činnost. Ovládaný objekt se nazývá regulovaná soustava či regulovaný systém a řídící člen regulátor. Soustava tvoří společně s regulátorem tzv. regulační obvod. [9]

4.1 Používané veličiny

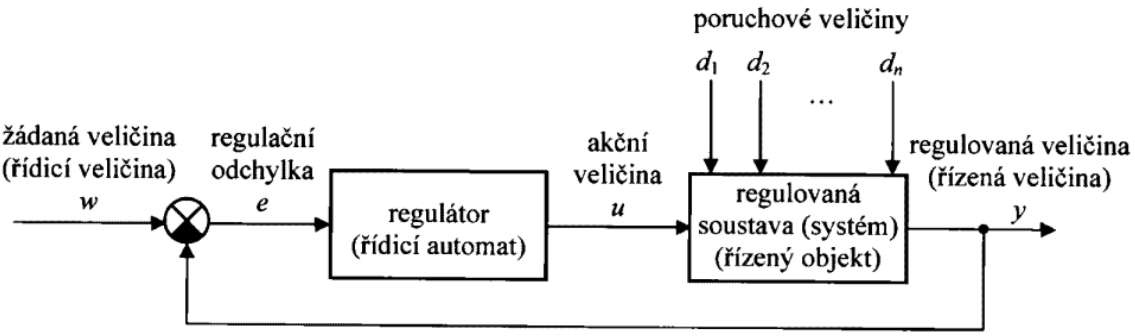
V teorii automatického řízení se používají 4 veličiny, pomocí kterých je popsán vztah mezi vstupními a výstupními parametry celého regulačního obvodu. Tyto veličiny se nazývají:

- regulovaná veličina – y,
- žádaná hodnota regulované veličiny – *w*,
- akční zásah – *u*,
- poruchová veličina – *d*.

Důležitým pojmem pro řízení je regulační odchylka, která představuje rozdíl žádané a aktuální hodnoty regulované veličiny. Obvykle ji značíme písmenem *e*. [9]

4.2 Regulátory

Regulátor je složen ze tří častí: měřícího, ústředního a akčního členu. [9] Měřící člen na základě žádané veličiny a aktuální hodnoty výstupní veličiny vypočítá regulační odchylku, která se následně předá do ústředního členu. Ústřední člen v závislosti na regulační odchylce a následně regulátor určuje akční zásah pro akční člen. Akční člen provede zásah do soustavy úměrný vstupnímu akčnímu zásahu. Tento zásah může být například pootevření či přivření ventilu ovládající přítok do nádrže. Ve většině případů je na regulátor nahlíženo v užším smyslu. To znamená, že se regulátorem myslí pouze jeho ústřední člen a zbývající dva členy jsou součástí soustavy. Schéma regulačního obvodu s regulátorem v užším smyslu je na obrázku č. 5. [9]



Obrázek č. 5: Regulační obvod [9]

4.2.1 Regulátory typu P-I-D

Regulátory typu P-I-D jsou regulátory lineární a spojité. Tvoří je 3 složky: proporcionální (P), integrační (I) a derivační (D). Podle kombinací složek použitých pro regulaci rozlišujeme 5 druhů těchto regulátorů:

- P-regulátor,
- I-regulátor,
- PI-regulátor,
- PD-regulátor,
- PID-regulátor. [9]

Proporcionální složka nastavuje akční veličinu úměrně regulační odchylce. Integrační složka má podobné chování, ale zároveň zohledňuje předchozí regulační odchylky a nastavuje akční veličinu úměrnou integrálu regulační odchylky. Derivační složka reaguje na velikost změny regulační odchylky. Nemůže být použita samostatně, protože v případě ustálené regulační odchylky by akční zásah derivační složky byl nulový. [9]

4.2.2 Dvoupolohový a třípolohový regulátor

Dvoupolohový a třípolohový regulátor patří mezi nelineární a nespojité regulátory. Nespojité regulátory se vyznačují tím, že akční veličina může nabývat pouze určitého počtu hodnot. Používají se v aplikacích s nižšími nároky na kvalitu řízení. [9]

Dvoupolohový regulátor

Dvoupolohový regulátor udržuje hodnotu regulované veličiny mezi dolní a horní mezní hodnotou, nazývanými také spínací vypínací úrovní. Rozdíl těchto úrovní se nazývá hystereze a jejím účelem je omezení kmitání akčního členu. Nejjednodušším dvoupolohovým regulátorem je bimetal, který k regulaci nepotřebuje ani pomocnou energii. [9]

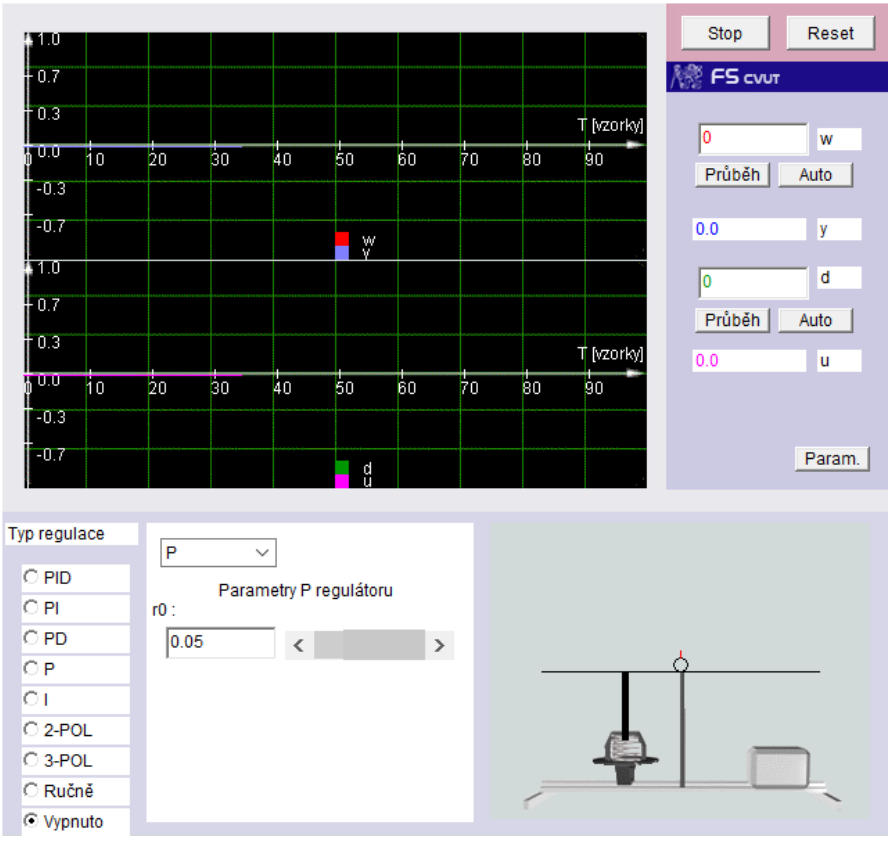
Třípolohový regulátor

U třípolohového regulátoru je na rozdíl od dvoupolohového mezi dvěma krajními polohami vřazena ještě třetí poloha s jinou úrovní akční veličiny. Obvykle se nazývá střední hodnotou a velikost akčního zásahu v této poloze bývá nulový. Rozmezí pro tuto polohu je tzv. pásmo necitlivosti. [10]

5 Původní virtuální laboratoř

Virtuální laboratoř byla již v minulosti několikrát upravována. Úlohy v ní jsou psané v jazyce Java a spouštěny v samostatných appletech. Jediný prohlížeč, ve kterém lze původní virtuální laboratoř v současné době spustit je Internet Explorer s nutností nainstalovat doplněk pro jazyk Java. Všechny úlohy mají jednotný vzhled, který se liší pouze v konkrétní části pro vizualizaci.

Na obrázku č. 6 je úloha kuličky na tyči. Největší plochu appletu zabírají dva grafy s černým pozadím. V horním grafu se vykresluje žádaná hodnota výstupu (w) a aktuální hodnota výstupu (y), ve spodním grafu akční zásah regulátoru (u) a externí poruchová veličina (d). Vedle grafů je hlavní ovládací panel celé úlohy umožňující spouštění a resetování celé úlohy včetně nastavování žádané hodnoty a poruchy. V dolní části úlohy se nachází nastavení regulátorů a samotná vizualizace úlohy, která vykresluje aktuální výstup (poloha kuličky) a žádanou hodnotu (cílová pozice). Většina funkcionalit původní virtuální laboratoře zůstala zachována i v nové. Z toho důvodu bude detailnější popis jednotlivých částí rozepsán v další kapitole.



Obrázek č. 6: Původní virtuální laboratoř

6 Nová virtuální laboratoř

V praktické části je popsán postup práce při vytváření nové virtuální laboratoře. Jsou zde uvedeny použité knihovny a jejich využití, zdůvodněna konkrétní rozhodnutí pro funkčnost i návrh celkového vzhledu.

6.1 Struktura programu

Celý program byl zpracován ve vývojovém prostředí Visual Studio od společnosti Microsoft a je tvořen 3 složkami:

- složka s html soubory,
- složka s obrázky,
- složka se skripty.

Samotná složka se skripty se zejména pro přehlednost a zjednodušení budoucích úprav dělí na další tři podsložky:

- složka s hlavními skripty,
- složka pro skripty jednotlivých úloh,
- složka s externími knihovnami.

Složky s hlavními skripty a s externími knihovnami jsou pro všechny úlohy stejné. Konkrétní úlohy se liší pouze ve vlastním skriptu a HTML souboru.

6.2 Konkrétní skripty a jejich funkce

Na základě rozdělení ve složkách jsou dále podrobněji popsány konkrétní skripty a jejich funkčnost.

6.2.1 Hlavní systémové skripty

Hlavních skriptů je v rámci projektu celkově devět. U třech z nich jsem vycházel z původní virtuální laboratoře a některé části jsou pouze přepsané z Javy do JavaScriptu. V celkovém objemu kódu tvoří přibližně 10 %, což značí, že se veliká část původního kódu ani struktury nezachovala a mnoho funkcionalit přibylo. Následuje podrobnější popis těchto souborů s tím, že nejprve budou zmíněny právě ty, které se zachovaly.

AbstractSystem

Tento skript obsahuje jednu třídu s názvem AbstractSystem, která je následně využitá ve všech skriptech úloh jako šablona konkrétních systémů. Třída obsahuje všechny základní parametry pro matematický model, citlivosti na poruchu i akční zásah, včetně maxim, minim a aktuálních hodnot (*w, y, u, d*). Dále obsahuje základní metody na nastavení (settery) či načtení (gettery) zmíněných vlastností a hlavní metodu *UpdateSystem*, která tvoří jádro celého projektu a je uvedena ve výpisu č. 3. Jejím účelem je výpočet aktuálního výstupu na základě předchozích vstupů, výstupů, poruchy a parametrů modelu.

```
UpdateSystem() {
        for (let i = 3; i > 0; i--)
        this.m_aLast[i] = this.m_aLast[i - 1];
        this.m_aLast[0] = this.m_fY;
        let cast1 = this.m_fdT * (this.m_fD1 * this.m_fSd1 + this.m_fD2 *
this.m_fSd2 + this.m_fSu * this.m_fU);
        let cast2 = this.m_fdT * (this.m_fA1 * this.m_aLast[0] + cast1);
        let cast3 = this.m_fdT * (this.m_fA2 * (2.0 * this.m_aLast[0] -
this.m_aLast[1]) + cast2);
        let cast4 = this.m_fdT * (this.m_fA3 * (3.0 * this.m_aLast[0] - 3.0 *
this.m_aLast[1] + this.m_aLast[2]) + cast3);
        let citatel = cast4 - this.m_fA4 * (this.m_aLast[0] - 4.0 *
this.m_aLast[1] + 3.0 * (2.0 * this.m_aLast[2] + this.m_aLast[3]));
        let jmenovatel = this.m_fdT * (this.m_fA3 + this.m_fdT * (this.m_fA2 +
this.m_fdT * (this.m_fA1 + this.m_fdT * this.m_fA0)));
        if (citatel == 0.0) {
            this.m_fY = 0.0;
        } else if (jmenovatel == 0.0) {
            this.m_fY = 0.0;
        } else {
            this.m_fY = citatel / jmenovatel;
        }
        this.limitY();
    }
```

Výpis č. 3: Metoda UpdateSystem

AbstractController

Stejně jako AbstractSystem obsahuje tento skript pouze jednu třídu s názvem AbstractController, která se používá ve třídě Controllers jako šablona pro jednotlivé regulátory. Tato třída má pouze 3 vlastnosti – žádanou hodnotu *w*, výstupní hodnotu *y* a vzorkovací periodu *dT*. Tyto vlastnosti mají všechny použité regulátory ve virtuální laboratoři a každý z nich ještě nějaké navíc. Dále obsahuje nejdůležitější metodu pro výpočet akčního zásahu *u*, která se však pro každý regulátor liší.

Controllers

S využitím principů objektově orientovaného programování a schopností jazyka JavaScript tyto principy aplikovat jsem vytvořil sedm tříd regulátorů dědících vlastnosti třídy AbstractController. Skript Controllers dále obsahuje dvě třídy pseudo-regulátorů, které se nechovají přímo jako regulátory, ale z důvodu obecnosti kódu bylo vhodné je vytvořit. Prvním ze zmíněných je manuální regulátor a druhým regulátor pro vypnutý stav. Hlavním úkolem těchto regulátorů je vyhnutí se vzniku prázdné reference na objekt a snadnější orientace v kódu i při následném pozorování chování programu přímo za chodu.

Engine

Hlavní chod programu probíhá ve třídě Engine obsahující referenci na systém a regulátor. Dále obsahuje metody:

- doCycle,
- run,
- updateCharts.

Metoda *doCycle* nejdříve načte hodnotu aktuálního výstupu systému metodou *getY*, vloží ji jako parametr metodě regulátoru *getU* počítající akční zásah a návratovou hodnotu této metody následně vloží jako parametr metodě systému *setU*. K aktualizaci systému dochází při volání metody *getY* zavoláním již zmíněné metody *updateSystem.* Tento postup je pro každou úlohu stejný a nezávisí na žádném externím parametru. Metoda *run* je jediná asynchronní metoda v celém projektu a má jedinou funkci – volat metodu *doCycle* v nekonečné smyčce, pokud je splněna podmínka zapnutí. Poslední zmíněná metoda *updateCharts* v každém cyklu smyčky aktualizuje výstupní grafy s hodnotami *w, y, u, d*.

Setup

Ve skriptu setup se nevyskytuje žádná třída, ale tvoří základ celého projektu. Obsahuje více než tisíc řádků kódů, což je přibližně o tři sta řádků více než všechny již zmíněné skripty dohromady. Celkově je tvořen 26 funkcemi, 72 obslužnými metodami a objekty s odkazy na jednotlivé elementy stránky. Tyto objekty jsou zejména tlačítka a vstupně/výstupní pole pro interakci s uživatelem. Pro získání reference a následné uložení do proměnné je využito objektového modelu dokumentu. Příklad je uveden ve výpisu č. 4. V prvním řádku je

uložena reference na objekt tlačítka pro zastavení do proměnné *stopButton*. Následně je k tomuto tlačítku přiřazena obslužná metoda, které při kliknutí na tlačítko zavolá metodu *stop*.

```
let stopButton = document.querySelector("#stop-button");
stopButton.addEventListener('click', () => {
    stop();
});
```

Výpis č. 4: Příklad přiřazení obslužné metody k tlačítku

Charts

Ve skriptu Charts se nacházejí dva grafy vykreslující zmíněné veličiny (*y, w, u, d*). Tyto objekty jsou instancemi třídy Chart, která je definována v externí knihovně Chart.js popsané v další kapitole. Konstruktor této třídy má 4 parametry:

- referenci na objekt canvas, ve kterém by se měl graf vykreslovat,
- typ grafu,
- data,
- nastavení a vlastnosti grafu.

Reference na objekt canvas je provedena příkazem `const canvas3 = document.getElementById('chart11').getContext('2d'),` ze kterého lze poznat, že odkaz necílí přímo na objekt, ale pouze na jeho vlastnost. Typ grafu je zvolen tzv. scatter (rozptýlený) a je odvozen od grafu spojnicového. Formát dat pro tento typ grafu odpovídá výstupním hodnotám veličin v úloze a jevil se proto jako nejvhodnější pro tuto aplikaci. Data se do něj zadávají ve formátu souřadnic, konkrétně `[{x:1,y:5}, {x:2,y:7}]`. Za souřadnici x se dosazuje uplynulý čas zapnuté úlohy, konkrétně počet vzorků a souřadnice y reprezentuje odpovídající výstupní hodnoty. Objekt typu Chart obsahuje veliké množství možností a nastavení, z nichž se v tomto konkrétním případě využívá pouze několik. Některé nastavení je rovněž prováděno v objektu dat. Toto nastavení souvisí přímo s daty, ovládá např. zda mají body konkrétního souboru dat být spojeny čárou, jakou barvou mají být vykresleny nebo pomocí jeho lze nastavit legendu související s konkrétním datovým souborem, kterých může graf obsahovat několik (viz Výpis č.5).

```
label: 'Y',
            data: [{ x: 0, y: 0 }],
            showLine: true,
            fill: false,
            borderColor: 'rgba(0, 200, 0, 1)',
            pointRadius: 0.1
```

Výpis č. 5: Nastavení datového souboru výstupní veličiny v grafu

V samostatném bloku nastavení lze přidávat do grafu dodatečné funkcionality (pluginy), aktivovat nebo deaktivovat globální vlastnosti, volit rozsah os apod. Ve výpisu č. 6 je zobrazena aktivace pluginu zoom a pan. Tímto je umožněno přibližovat a oddalovat graf kolečkem myši a posouvat ho. Mód interakce je nastaven na „xy“, což znamená, že pohyb není omezen jen na některou z os. Pro posuv v grafu je nutné na něj kliknout a pohybovat kurzorem při současném držení tlačítka shift. Toto tlačítko by mohlo být i jiné a nastavuje se ve vlastnosti modifierKey.

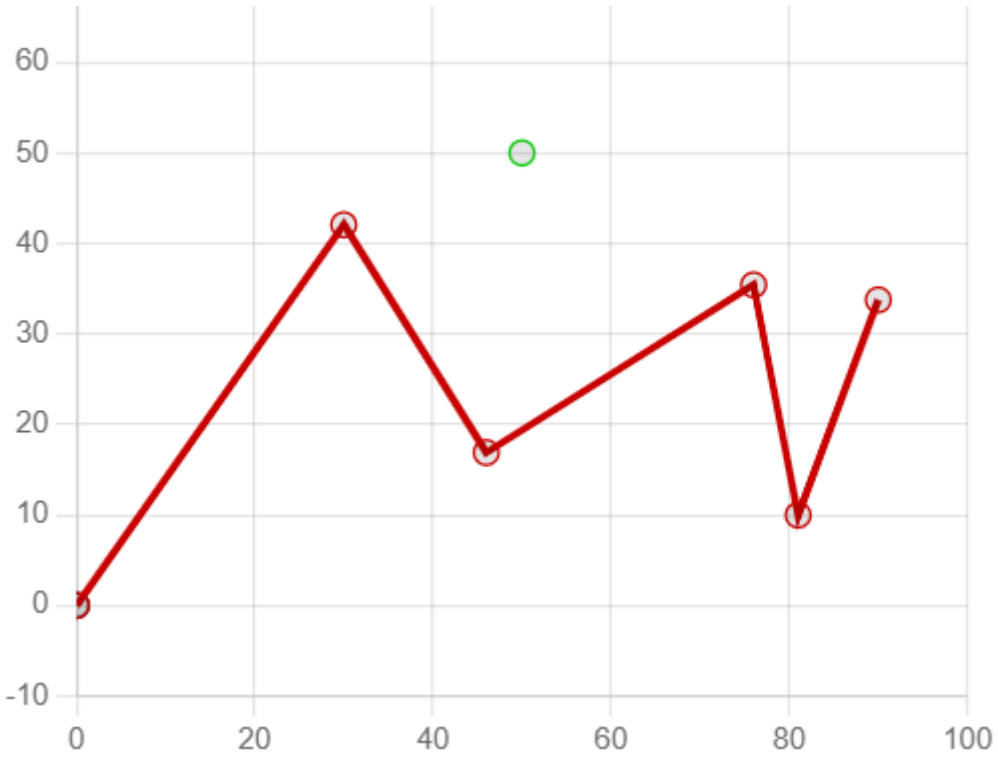
```
plugins: {

            zoom: {
                zoom: {
                    wheel: {
                        enabled: true,
                    },
                    pinch: {
                        enabled: true
                    },
                    mode: 'xy',
                },
                limits: {
                    y: { min: -500, max: 500 },
                    x: { min: 0, max: 50000 }
                },
                pan: {
                    enabled: true,
                    mode: 'xy',
                    modifierKey: 'shift'
                }
            }
        }
```

Výpis č. 6: Ukázka přidání pluginů ke grafu

InputGrid

Skript s názvem InputGrid obsahuje funkcionality pro interaktivní graf, včetně 4 grafů stejného typu, jako grafy ve skriptu Charts. Většina zmíněných principů u použitých grafů v předchozím odstavci se využívá i v tomto případě. Každý graf znázorňuje průběh jedné veličiny a umožňuje nastavení žádaného průběhu její hodnoty. Z důvodu většího počtu grafů

a jejich podobných vlastností jsem vytvořil třídu InputGrid. Každá instance této třídy obsahuje referenci na konkrétní graf a doplňující informace pro konkrétní hodnotu. Důležitou vlastností je interní čítač, který je aktivní pouze při aktivaci dané instance. Všechny parametry jsou interní a využívané pouze konkrétní instancí, tím je zajištěna nezávislost nastavování všech veličin a při stejném nastavení lze dosáhnout jiných celkových průběhů aktivací daných instancí v rozdílný čas. Ukázka interaktivního grafu je na obrázku č. 7.



Obrázek č. 7: Interaktivní graf

Tato třída obsahuje 13 funkcí a 18 atributů. Použité funkce umožňují uživateli vkládat body kliknutím, posouvat je tahem, odstraňovat je, upravovat jejich souřadnice a vložit přednastavenou množinu bodů podle nastavený parametrů. Vnitřní proměnné jsou nutné pro lepší interakci s uživatelem. Je mezi nimi například naposled vložený a odstraněný bod, právě upravovaný bod, předchozí a následující bod pro případ, kdy již probíhá vyčítání hodnoty z grafu a dále informace o tom, zda je aktivní manuální či automatický mód.

Mezi důležité metody patří:

- addPoint,
- findNextPoint,
- autoModeCycle.

Metoda *addPoint* má dva parametry, souřadnici x a y. Tato metoda, která vkládá bod do grafu, nejprve hledá v datovém souboru konkrétního grafu, zda již neexistuje bod se stejnou souřadnicí x rovnou zadanému parametru. Pokud takový bod existuje, je odstraněn a nahrazen novým bodem, který může mít odlišnou souřadnici y. V opačném případě se bod přidá do datového souboru přímo. Na konci metody jsou všechny body ještě setříděny podle souřadnice x a graf překreslen podle aktuálních hodnot. Po vložení bodu se nastaví hodnota proměnné vyjadřující vložení bodu na logickou true. Tuto proměnnou využívají následně funkce *autoModeCycle*.

Metoda *findNextPoint* je zavolána v metodě *autoModeCycle* v případě, že byl přidán či odebrán bod. Pokud je bod vložen mimo dva body, mezi kterými se v současné době nachází hodnota interního čítače, nestane se nic, v opačném případě proběhne kód ve výpisu č. 7. Pokud je hodnota čítače větší než souřadnice x naposledy přidaného bodu, změní se odkaz na následující bod z původního na nový. Analogicky to platí pro případ, že je hodnota čítače nižší. Pokud by se tato metoda nepoužila, tak by při přidání bodu graf adekvátně nezareagoval a vyčítala by se z něj jiná hodnota, než která odpovídá aktuálnímu průběhu.

```
if (this.lastAddedPoint.x < this.nextPoint.x && this.lastAddedPoint.x >
this.lastPoint.x) {
                if (this.counterGlobal >= this.lastAddedPoint.x) {
                    this.lastPoint = this.lastAddedPoint;
                }
                else if (this.counterGlobal < this.lastAddedPoint.x) {
                    this.nextPoint = this.lastAddedPoint;
                }
```

Výpis č. 7: Přidání nového bodu při aktivním vyčítání hodnoty

Metoda *autoModeCycle* je volána pro každou veličinu zvlášť a pouze v případě, že není aktivní manuální vkládání hodnoty a probíhá hlavní programový cyklus. V metodě *autoModeCycle* se mimo volání metody *findNextPoint* řídí interní časovač a získává hodnota z průběhu, která je zároveň i návratovou hodnotou této metody. Čítač mění svou hodnotu v každém průběhu cyklu s inkrementem rovné jedné. Pokud dosáhne maxima, což je souřadnice x posledního bodu v datovém souboru, resetuje se jeho hodnota na nulu. Vyčítání hodnoty z grafu se provádí za použití lineární interpolace. Vypočítá se rozdíl souřadnic sousedních bodů. Rozdíl souřadnic y sousedních bodů je podělen rozdílem souřadnic x stejných bodů, čímž se získá směrnice přímky spojující tyto body. Výsledná hodnota je rovna

součtu souřadnice y bodu, který má nižší souřadnici x, a směrnici přímky násobené počtem časových kroků od tohoto bodu.

Relativně složitý problém představuje nastavení posouvání bodů v grafu. Každý bod má unikátní souřadnici x a je žádoucí, aby uživatel nemohl přetáhnout bod tak, aby se bod s nižší souřadnicí x, než bod dále na ose x dostal za něj a opačně i v případě snižování souřadnice x. Tímto je docíleno snazšího ovládání a je možné posunout body tak, že jsou od sebe vzdáleny přesně o jeden souřadnicový bod x, což odpovídá jednomu časovému kroku a průběhu hlavního cyklu programu.

Ve výpisu č. 8 je kód, který se vykonává v případě, že uživatel pohybuje s bodem. Podmínky nejdříve ověřují, zda se nemanipuluje s krajním bodem datového souboru. Následně se ověřuje souřadnice x. Pokud dojde k překročení souřadnice některého ze sousedních bodů, uloží se tato souřadnice jako limitní. Ačkoliv uživatel může pohybovat kurzorem i za touto hranicí, tak se bod již nehýbe a zůstává na místě, kde došlo k překročení limitu. Po uvolnění bodu dojde k uložení bodu s novými souřadnicemi. V případě, že se uživatel kurzorem vrátí zpět mezi dva sousední body pokračuje program dále a s bodem lze pohybovat jako v počátku. Samotné uložení bodu v následujícím kódu není a je prováděno po vyvolání události ukončení tažení bodu.

```
onDrag: function (e, datasetIndex, index, value) {
                e.target.style.cursor = 'grabbing';
if (index != parseInt(W_input.chart.data.datasets[datasetIndex].data.length)-1 )
{
                    if (W_input.chart.data.datasets[datasetIndex].data[index].x >
W_input.chart.data.datasets[datasetIndex].data[parseInt(index) + 1].x) {
                        dragDenied = true;
                        if (!limitMaxSet) {
                            limitMax =
parseInt(W_input.chart.data.datasets[datasetIndex].data[index + 1].x)-1;
                            limitMaxSet = true;
                        }
                        return false;
                    }
                    else {
                        dragDenied = false;
                        limitMaxSet = false;
                    }
                }
                if (index != 0) {
                    if (W_input.chart.data.datasets[datasetIndex].data[index].x <
W_input.chart.data.datasets[datasetIndex].data[index - 1].x) {
                        dragDenied = true;
                        if (!limitMinSet) {
                            limitMin =
parseInt(W_input.chart.data.datasets[datasetIndex].data[index -1].x)+1;
                            limitMinSet = true;
                        }
                        return false;
                    }
                    else {
                        dragDenied = false;
                        limitMinSet = false;
                    }
                }
```

Výpis č. 8: Funkce pro tažení bodu v grafu

Languages

Skript Languages obsahuje 4 metody pro změnu jazyků, které jsou volány po kliknutí na příslušnou státní vlajku. Dostupnými jazyky jsou čeština, angličtina, němčina a ruština. Metody jsou velice podobné, kód v nich se liší jen velmi málo. Na každém řádku kódu je pomocí zmíněného objektového modelu dokumentu označen jeden prvek stránky, který obsahuje text a je mu přiřazena hodnota v odpovídajícím jazyce. Příkladem může být přepsání písmena y na odpovídající znak azbuky ы ve výpisu č. 9.

```
document.querySelector("#y-val").innerHTML = "Ы = " +
mainEngine.m_System.m_fY.toFixed(2);
```

Výpis č. 9: Ukázka změny jazyka

V tomto případě je úryvek kódu z části vypisující hodnotu *y* a z toho důvodu je ke znaku v azbuce ještě přidán znak rovnítka a aktuální hodnota výstupu *y* ze systému. Pro přidání dalšího jazyka stačí zkopírovat metodu pro některý z již použitých jazyků (např. *changeLangEng*) a přepsat pravé strany řádků kódu.

DownloadUpload

Posledním skriptem společným pro všechny úlohy je DownloadUpload. Obsahuje funkce pro ukládání dat a jejich následné načtení zpět do úlohy. Hlavním důvodem pro jeho vytvoření bylo zjednodušení výuky včetně výuky ve vzdáleném režimu. Uživatel má možnost uložit nastavení úlohy a výstupní hodnoty veličin. Která data uložit lze zvolit zaškrtnutím checkboxů, takže lze například uložit pouze nastavení regulátorů. Po zavolání metody stažení dat je vytvořen soubor formátu .csv, který je vhodný pro práci s daty v tabulkových kalkulátorech. Struktura tohoto souboru je pro každou úlohu i variantu ukládaných dat stejná, ale data se ukládají vždy jen do příslušných buněk odpovídajících uživatelskému nastavení ukládání zvolenému zaškrtnutím patřičných checkboxů. Celá struktura je zobrazena ve výpisu č. 10. Celkově se může ukládat 5 typů dat: parametry regulátoru, výstupní data, souřadnice bodů z interaktivního grafu, aktuální hodnoty a systémové parametry.

```
    var csvFileData = [
        ['Regulator parameters'],  // 0
        ['P-reg', 'P', 'P-val'], // 1
        ['I-reg', 'I', 'I-val'],// 2
        ['PI-reg', 'P', 'P-val', 'I', 'I-val'],// 3
        ['PD-reg', 'P', 'P-val', 'D', 'D-val'],// 4
        ['PID-reg', 'P', 'P-val', 'I', 'I-val', 'D', 'D-val'],// 5
 ['2-pol-reg', 'Hyst', 'Hyst-val', 'U-min', 'U-min-val', 'U-max', 'U-max-
val'],// 6
['3-pol-reg', 'Hyst', 'Hyst-val', 'U-min', 'U-min-val', 'U-str', 'U-str-   val',
'U-max', 'U-max-val', 'Necit', 'Necit-val'],// 7
        [],// 8
        ['Graph outputs', 'counter', 'counter-val'],// 9
        ['Y-x', 'Y-x-values'],// 10
        ['Y-y', 'Y-y-values'],// 11
        ['W-x', 'W-x-values'],// 12
        ['W-y', 'W-y-values'],// 13
        ['U-x', 'U-x-values'],// 14
        ['U-y', 'U-y-values'],// 15
        ['D1-x', 'D1-x-values'],// 16
        ['D1-y', 'D1-y-values'],// 17
        ['D2-x', 'D2-x-values'],// 18
        ['D2-y', 'D2-y-values'],// 19
        [],// 20
        ['Graph inputs'],//21
        ['W-x', 'W-x-inp'],// 22
        ['W-y', 'W-y-inp'],//23
        ['U-x', 'U-x-inp'],// 24
        ['U-y', 'U-y-inp'],//25
        ['D1-x', 'D1-x-inp'],// 26
        ['D1-y', 'D1-y-inp'],// 27
        ['D2-x', 'D2-x-inp'],// 28
        ['D2-y', 'D2-y-inp'],//29
        [],// 30
 ['Actual values', 'Y', 'Y-val', 'W', 'W-val', 'U', 'U-val', 'D1', 'D1-val',
'D2', 'D2-val', 'm_aLast1', 'm_aLast2', 'm_aLast3', 'm_aLast4', 'actCont'],//31
        [],//32
 ['System parameters', 'a1', 'a1-val', 'a2', 'a2-val', 'a3', 'a3-val', 'a4',
'a4-val', 'Su', 'Su-val', 'Sd1', 'Sd1-val', 'Sd2', 'Sd2-val' ],// 33
        ['Developed by Martin Hubacek, 721 265 725'] //34
    ];
```

Výpis č. 10: Struktura ukládaného souboru

Načítání dat probíhá analogicky k ukládání a využívá znalosti struktury načítaného souboru. Uživatel má i při načítání dat možnost zvolit jaká data do úlohy nahrát. Funkce zároveň ověřuje, zda jsou v souboru kompatibilní data. Pokud by došlo ke změně struktury načítaného souboru, mohlo by dojít k načtení dat do nesprávných proměnných. I z tohoto důvodu byla zvolena pevná struktura souboru umožňující snadnou kontrolu.

6.2.2 Skripty jednotlivých úloh

Každá úloha virtuální laboratoře má vlastní html soubor a skript. Struktura skriptů úloh je následující:

- původní hodnoty parametrů,
- aktuální parametry systému,
- aktuální parametry regulátorů,
- vizualizační část,
- třída konkrétní úlohy,
- nastavení úlohy.

V každém skriptu se nachází v horní části kód s původními parametry převzatými z původní virtuální laboratoře. Tyto parametry se nevyužívají, ale hodnotově se ve většině případů shodují s aktuálně používanými parametry. Parametry matematického modelu zůstaly stejné, ale bylo nutné změnit hodnoty omezení akčního zásahu, žádané veličiny, výstupní veličiny i poruchy z důvodu kompatibility s vizualizační částí programu. Parametry regulátoru jsou zapsány ve formě objektu v následujícím tvaru: `par_PIR0 = {val: 0.05, min: 0, max: 1};` První hodnota je aktuální hodnota nahraná jako počáteční nastavení regulátoru při restartování úlohy a další dva členy představují minimum a maximum možných hodnot, v tomto případě pro proporcionální složku PI regulátoru.

Vizualizační část je pro všechny úlohy téměř stejná, ale liší se odkazem na obrázek konkrétní úlohy. Obsahuje tři objekty typu canvas, které se zobrazují v různých vrstvách za pomoci atributu z-index, zmíněného v teoretické části. V nejnižší vrstvě se nachází obrázek pozadí či pouze výplň specifické barvy, uprostřed průhledný obrázek úlohy a v nejvyšší vrstvě se provádí vykreslování.

Třída úlohy se skládá se dvou částí. První tvoří konstruktor, ve kterém se provádí přiřazení parametrů systému. Ve druhé části je metoda vykreslení. Ta je pro každou úlohu zcela unikátní. V některých úlohách, například u batyskafu, může tato metoda obsahovat pouhých osm řádků kódu, ale v nejsložitější úloze pro wattův roztěžník i více než sto.

Při načtení úlohy se provede úvodní inicializace systému následujícím způsobem. Vytvoří se instance třídy konkrétní úlohy a proměnná typu Engine, do které se následně uloží reference na vytvořený objekt systému a ve které se provádí hlavní chod programu. Jako poslední se provede prvotní vykreslení úlohy ve vizualizační části (viz Výpis č.11).

```
let mainEngine = new Engine();
let system = new WattuvRozteznik();
mainEngine.SetSystem(system);
system.draw();
```

Výpis č. 11: Vytvoření systému pro konkrétní úlohu

6.2.3 Použité knihovny

Celkově bylo v práci použito šest externích knihoven. Byly vybírány volně dostupné knihovny s kvalitní dokumentací a perspektivou fungování mnoha let, aby byla zajištěna možnost snadné úpravy virtuální laboratoře i v budoucnosti. Na jednu požadovanou funkčnost existuje mnoho knihoven, ale nebylo by vhodné volit knihovny s malou komunitou uživatelů i vývojářů, což v konečném důsledku zvyšuje pravděpodobnost ukončení podpory v blízké budoucnosti. Menší počet vývojářů zároveň často znamená i nižší kvalitu dokumentace.

Chart.js

Tato knihovna je pravděpodobně nejdůležitější ze všech použitých. Umožňuje vykreslování grafů a následnou práci s nimi. Společně se základní verzí byly použity ještě dvě doplňující knihovny: Chart-zoom a Chart-drag. Tyto knihovny dodávají možnost přibližování obrazu a přesouvání bodů v grafu. Chart-drag však potřebuje ještě zjišťovat, jak uživatel interaguje s grafem a následně na tyto události reaguje. Pomyslný přechod mezi uživatelem a knihovnou Chart-drag představuje knihovna Hammer.js. [11]

Hammer.js

Knihovna Hammer.js byla vytvořena zejména pro dynamizaci vizualizace a vyhodnocování uživatelských zásahů nejen myší, ale v případě dotykových obrazovek i dotyků prstů. Hlavním důvodem použití této knihovny ve virtuální laboratoři byla možnost lepší interakce s grafy, konkrétně možnost využití knihovny Chart-drag, která umožňuje pohybovat s body v grafu. Hammer.js zpracuje událost, ve většině případů kliknutí na bod, přidržení myši a následný pohyb po grafu a předává nutné informace knihovně Chart-drag, která pohybuje body. [12]

Papaparse.js

Knihovna Papaparse, konkrétně její příkaz *parse* (viz Výpis č.12), se ve skriptu DownloadUpload používá při načtení dat ze souboru na objekt načtené složky. Tato metoda následně vrátí dvoudimenzionální pole, ze kterého je již relativně jednoduché získat potřebná data a nahrát je do nastavení úlohy. [13]

```
Papa.parse(document.getElementById("file-input").files[0], {
        complete: function (results) {file = results.data; … }
```

Výpis č. 12: Načítání dat ze souboru

Bootstrap.css

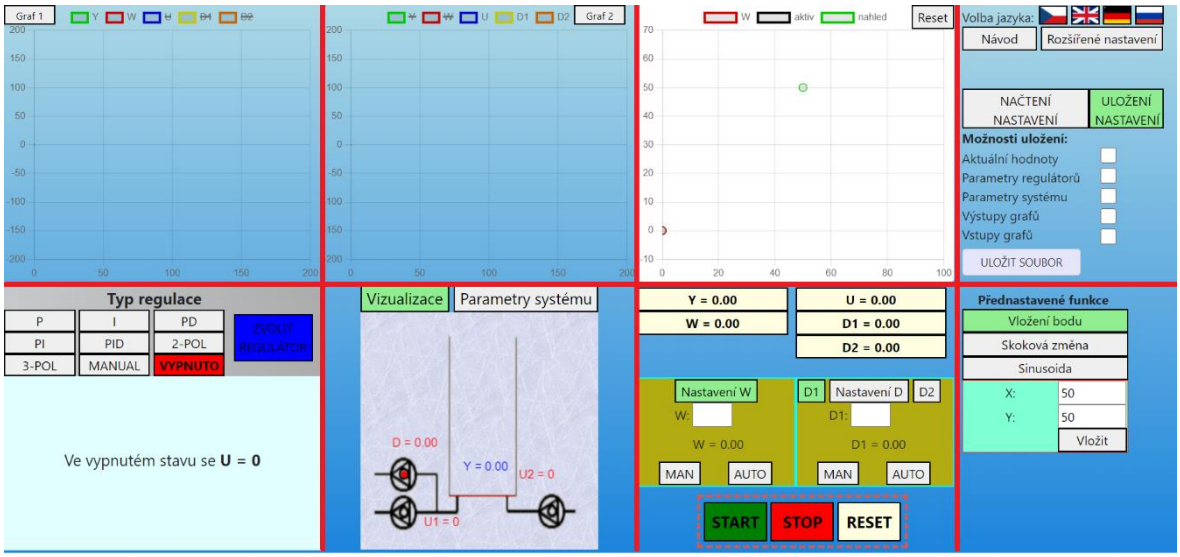
Knihovna Bootstrap je externí seznam pravidel CSS, které obsahuje základní nastavení některých prvků na stránce. Dále lze využít i přednastavené třídy pro jednoduší ovládání vlastností elementů. Toto řešení má však nevýhodu, protože je nutné z dokumentace zjistit, jaká pravidla konkrétní třídy obsahují. Zároveň to ztěžuje případné předávání kódu jinému uživateli, který nemusí mít o této knihovně povědomí. Z toho důvodu byla tato knihovna využita pouze pro základní nastavení elementů, ale všechna ostatní CSS pravidla jsou psaná pro přehlednost přímo v kódu. [14]

Nepoužité knihovny

V souvislosti s využitím objektů SVG pro vizualizaci místo objektů canvas byla vyzkoušena knihovna Anime.js, Kute.js a GSAP.js. Knihovna GSAP.js se zpočátku jevila jako vhodná, ale po hlubším zkoumání se projevila nutnost zaplacení licence pro využití určitých prvků, konkrétně dynamickou změnu velikosti SVG objektů. V porovnání Anime.js a Kute.js nakonec zvítězila knihovna Anime.js z důvodu jednodušší syntaxe kódu. Knihovna Anime.js zůstává jako součást projektu a je možné ji využít v případě záměru vizualizace úloh pomocí SVG objektů. [15, 16, 17]

6.3 Rozložení stránky

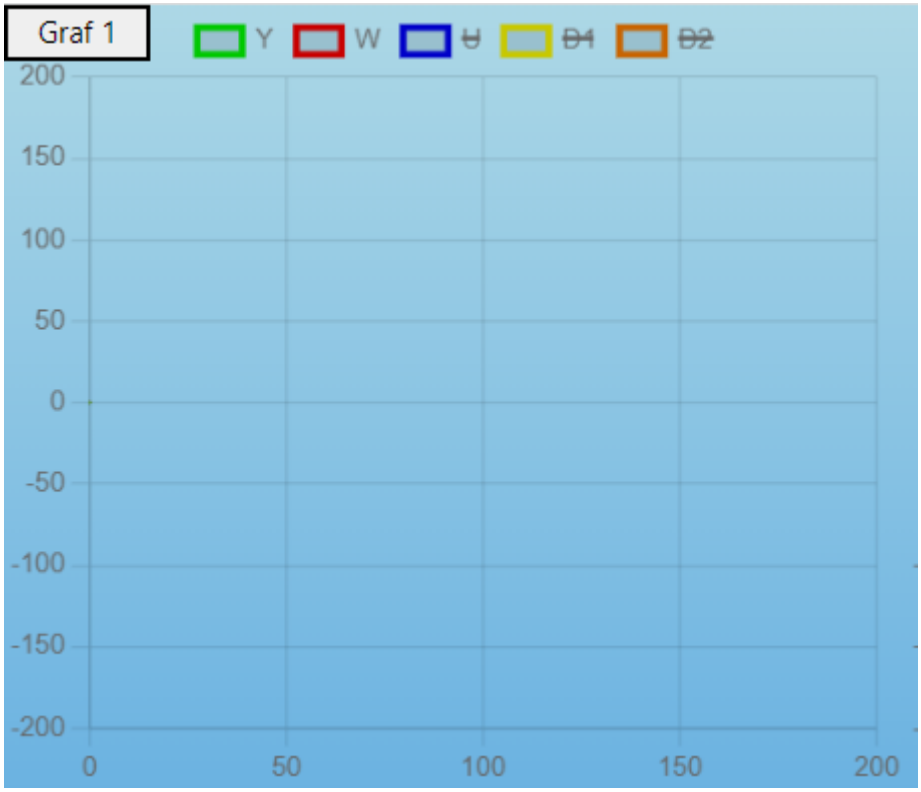
Zachování podobnosti s původní virtuální laboratoří nebylo nutné, ale v některých případech se původní řešení jevilo jako nejvhodnější. Všechny úlohy mají jednotný vzhled a odlišují se pouze v konkrétních hodnotách modelu a vizualizační částí. Základní rozložení stránky je ve formátu tabulky s dvěma řádky a čtyřmi sloupci a je zobrazeno na obrázku č. 8. Jednotlivé buňky pomyslné tabulky jsou na obrázku odděleny červenou čarou. Toto rozložení umožnilo vytvořit přehledný kód.



Obrázek č. 8: Rozložení stránky

6.3.1 Výstupní grafy

První dvě buňky pomyslné tabulky v levém horním rohu obsahují grafy s výstupními veličinami (viz Obrázek č. 9). Každá z nich obsahuje tlačítko, které umožňuje zvětšení příslušného grafu dané buňky na plochu obou buněk. Tyto grafy využívají externí knihovnu Chart.js a jsou konfigurovány v nově vytvořeném skriptu Charts.js. U grafů lze kliknutím na konkrétní značky v legendě zvolit, jaké veličiny by se na nich měly zobrazovat. Oba grafy obsahují stejná data a nastavení s výjimkou toho, jaké veličiny se na nich po načtení úlohy zobrazí. První graf zobrazuje žádanou hodnotu *w* a výstupní hodnotu *y*, druhý graf akční zásah *u* a poruchu *d*. Je tedy například možné zobrazit všechny hodnoty na jednom grafu a zvětšit ho, nebo naopak zobrazit akční zásah vedle žádané hodnoty.



Obrázek č. 9: Výstupní graf

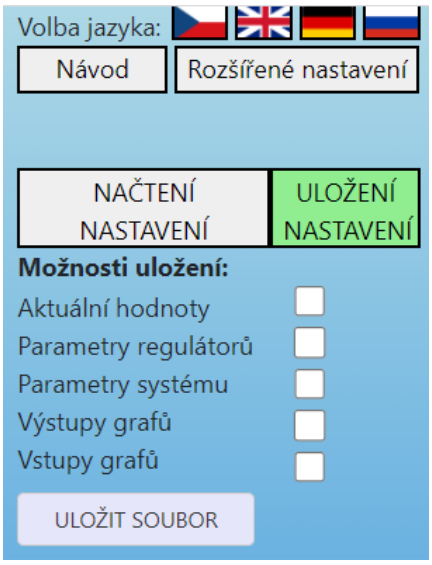
6.3.2 Interaktivní graf

Vedle výstupních grafů se nachází interaktivní graf pro vkládání bodů a získání konkrétního průběhu některé z veličin. Využívá zejména skript InputGrid popsaný v kapitole 6.2.1. Společně s tímto skriptem zde ještě zasahují přednastavené funkce podrobněji popsané v kapitole 6.3.7. Zároveň však umožňuje uživatelům zkoušet mnoho kombinací nastavení systému, které by se s pouhým manuálním řízením nedaly realizovat už z důvodu nutnosti nastavování několika veličin ve stejnou dobu.

6.3.3 Nastavení jazyků a rozšířené nastavení

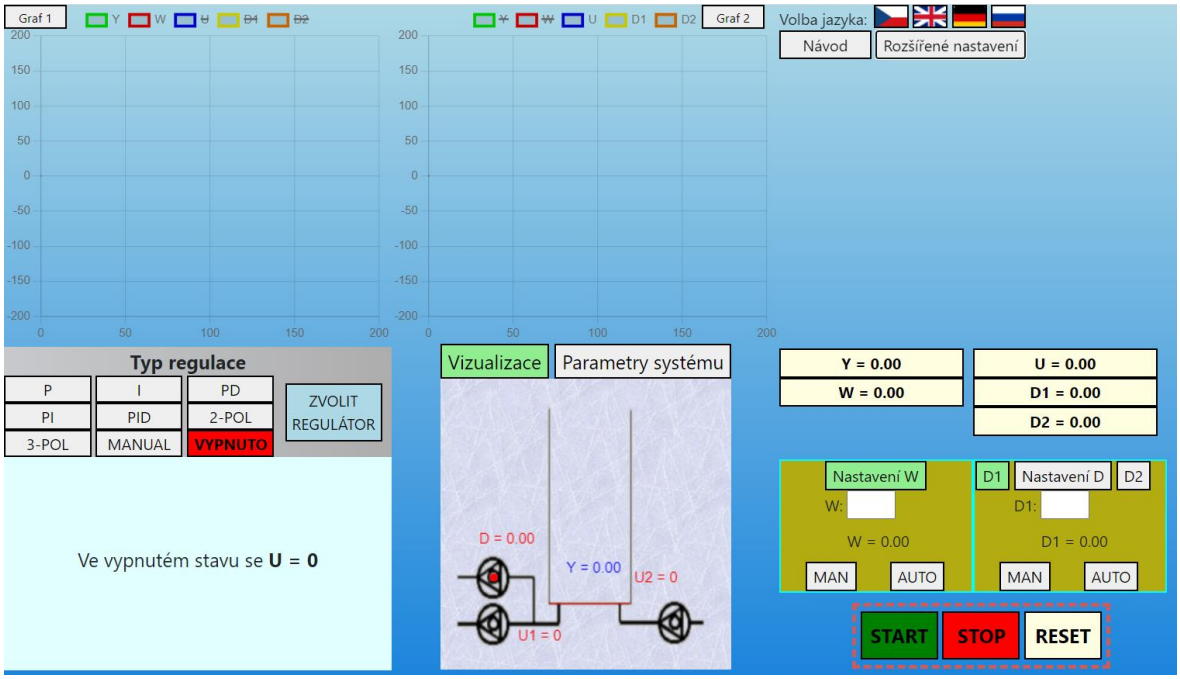
Poslední pole v prvním řádku pomyslné tabulky (viz Obrázek č. 10) obsahuje možnost volby jazyka, návod, tlačítko pro zobrazení rozšířeného nastavení a oddíl ukládání a načítání dat. Návod se otevírá jako nové pop-up okno prohlížeče a lze s ním manipulovat kdykoliv během využívání aplikace. Stránka návodu obsahuje popis základního ovládání úlohy a zejména seznam speciálních klávesových zkratek, které je nutné znát pro využití určitých funkcí úlohy. Volba jazyků je znázorněna čtyřmi vlajkami. Vlajky jsou vloženy jako obrázek na pozadí do elementů tlačítek a po kliknutí na příslušnou vlajku je vyvolána událost tlačítka, na kterou zareaguje příslušná obslužná metoda popsaná ve skriptu Languages. Tato metoda

změní jazykovou lokalizaci aplikace. Tlačítkem pro rozšířené nastavení se ovládá viditelnost 3 sekcí – interaktivního grafu, oddílu ukládání a načítání dat a části s přednastavenými funkcemi.



Obrázek č. 10: Oddíl ukládání dat a volby jazyka

Na již zmíněném obrázku č. 8 je obrazovka s viditelným rozšířeným nastavení a na obrázku č. 11 naopak bez něj.

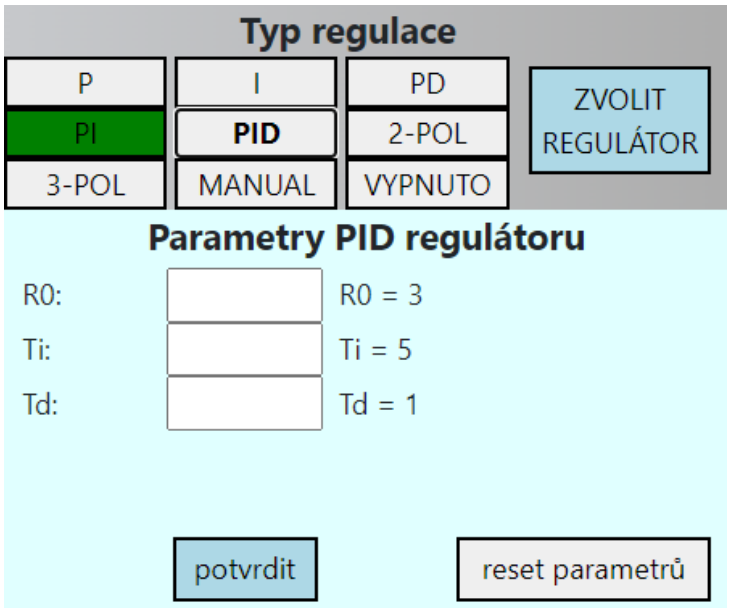


Obrázek č. 11: Úloha se skrytým rozšířeným nastavením

6.3.4 Regulační část

V druhém řádku je na prvním místě regulační část (viz Obrázek č. 12), kde uživatel přiřazuje soustavě regulátor a zároveň může upravovat parametry jednotlivých regulátorů. Po kliknutí na konkrétní tlačítko se zobrazí parametry regulátoru, ale regulátor soustavy se nezmění. Přiřazení regulátoru soustavě se musí potvrdit dalším tlačítkem pro zvolení regulátoru. Pokud uživatel mění parametry regulátoru, který není k systému přiřazen, tlačítko pro zvolení regulátoru mění barvu každou vteřinu, aby byl uživatel upozorněn, že se úpravy parametrů projeví až po přiřazení změněného regulátoru k systému.

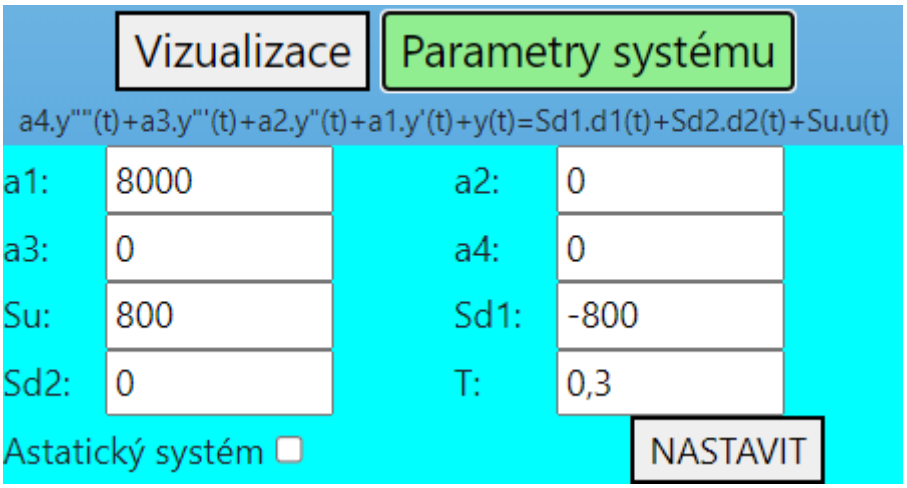
Tlačítko regulátoru přiřazeného soustavě má zelenou výplň a tlačítko regulátoru, kterého parametry jsou právě viditelné, obsahuje písmo tučné. Nastavení parametru je potřeba potvrdit tlačítkem.



Obrázek č. 12: Regulační část

6.3.5 Vizualizace a systémové parametry

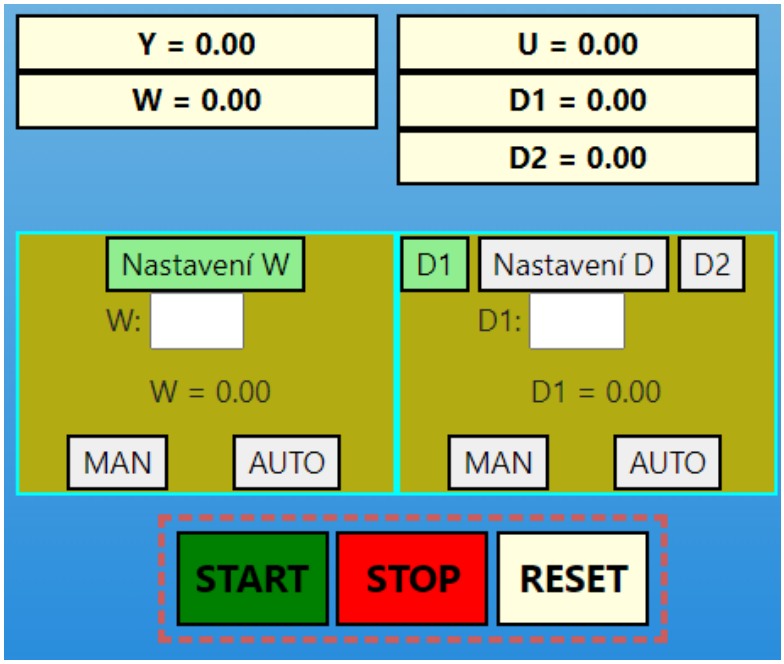
Napravo od regulační části se nachází část vizualizační společně s parametry systému (viz Obrázek č. 13). Vizualizace se nachází v samostatném oddílu ( ), stejně jako parametry systému. Je tvořena elementem canvas a obrázkem či několik obrázky s odlišnými atributy z-index pro zobrazování ve vrstvách. Uživatel má možnost tlačítky ovládat viditelnost oddílů vizualizace i systémových parametrů. Vždy je vidět pouze jeden z nich.



Obrázek č. 13: Parametry systému

6.3.6 Hlavní ovládání

V pravém dolním rohu na obrázku č. 11 je umístěné hlavní ovládání celé úlohy (viz Obrázek č. 14). V horní části této buňky jsou všechny výstupní veličiny vypisovány ve formátu zaokrouhleného čísla na dvě desetinná místa. Dále tu lze nastavovat žádanou hodnotu *w* hodnotu poruchy. V případě úlohy auta na nakloněné rovině je zde možné nastavovat poruchy dvě. Ve spodní části buňky jsou tři tlačítka, která spouští, zastavují a resetují celou úlohu.



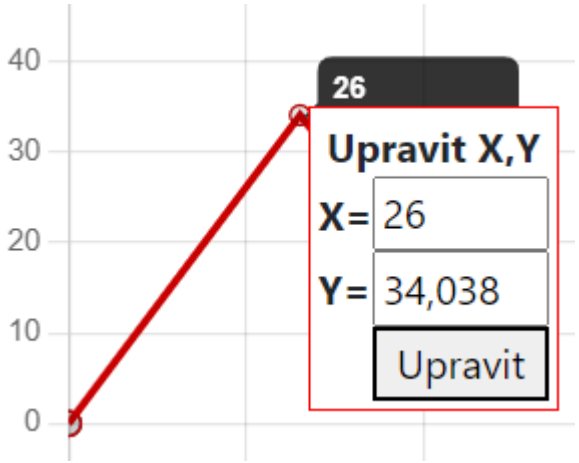
Obrázek č. 14: Hlavní ovládací panel

6.3.7 Přednastavené funkce

Poslední část grafického návrhu v pravém dolním rohu obsahuje přednastavené funkce. V současném návrhu jsou tyto tři funkce:

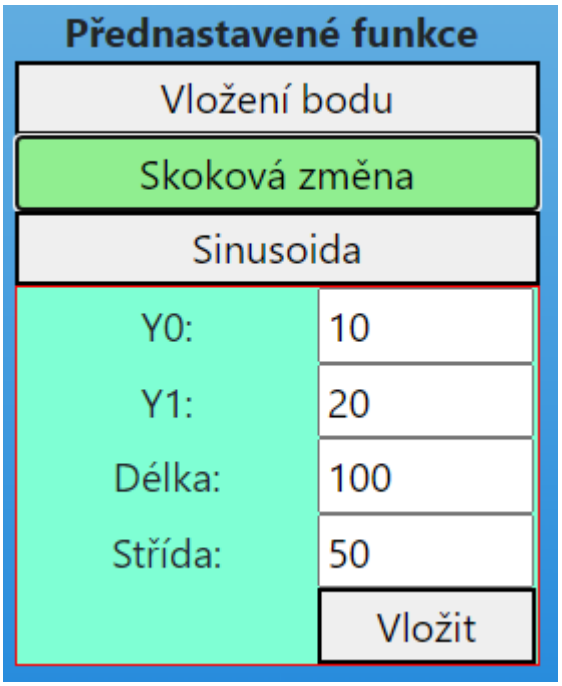
- vložení bodu,
- vložení skokové změny,
- vložení harmonického signálu.

Funkce pro vložení bodu vykonává stejný kód, který se spouští po kliknutí do interaktivního grafu. Parametry funkce jsou souřadnice x a y, které jsou v případě kliknutí myši obecně reálná čísla, ale při použití přednastavené funkce lze bod vložit s přesnými souřadnicemi. S vkládáním bodů rovněž souvisí možnost jejich úpravy. Uživatel může na bod kliknout levým tlačítkem myši a do doby, než ho uvolní se bod pohybuje s kurzorem. V případě kliknutí na bod pravým tlačítkem místo levého se otevře malé kontextové menu (viz Obrázek č. 15) umožňující úpravu souřadnic bodu. Tato funkce využívá potlačení vyvolání události. Pro odstranění bodu musí uživatel držet stlačenou klávesu ctrl a kliknout na bod levým tlačítkem.



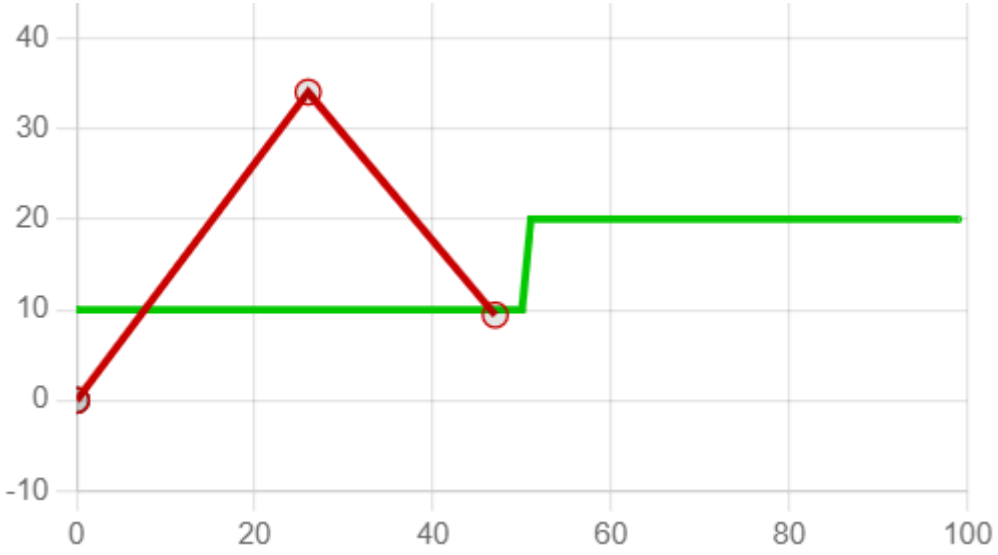
Obrázek č. 15: Kontextové menu pro editaci bodů

Druhou přednastavenou funkcí je vložení skokové změny. Touto cestou je možné získat odezvu systému na jednotkový skok či obdélníkový signál. Uživatel volí čtyři parametry – délku, počáteční a koncovou hodnotu a střídu, která udává, jak dlouho bude hodnota na počáteční a jak dlouho na koncové hodnotě. Nastavení skokové změny je na obrázku č. 16.



Obrázek č. 16: Blok přednastavených funkcí

Každý parametr u všech přednastavených funkcí vyvolává událost při změně jeho hodnoty. Tím dochází k aktualizaci náhledu vytvářeného průběhu při každém zásahu uživatele bez nutnosti dalšího potvrzování či opouštění vstupního pole. Příklad náhledu vytvářeného průběhu skokové změna je na obrázku č. 17. Zelený průběh zároveň odpovídá hodnotám z obrázku č. 16 a červeně je v grafu zobrazen aktivní průběh hodnoty. V případě potvrzení vytvoření žádaného průběhu pomocí přednastavené funkce dochází k přepsání dat a z náhledu se stane aktivní průběh.



Obrázek č. 17: Ukázka využití přednastavené funkce skokové změny

6.4 Chod programu

Po zapnutí úlohy se načtou základní parametry konkrétní úlohy a provede se inicializační funkce. Uživatel může následně upravit parametry regulátoru, přiřadit ho soustavě, nastavit žádané průběhy některých veličin a u každé z nich nastavit manuální či automatický režim. Úloha samotná se spustí tlačítkem start. Po zapnutí úlohy se začne vykonávat jediná asynchronní metoda *run* (viz Výpis č. 13), ve které probíhá cyklus, který se opakuje tak dlouho, než uživatel klikne na tlačítko stop.

```
async run() {
       while (!this.m_bPause) {
           this.doCycle();
           await pause(85);
        }
    }
```

Výpis č. 13: Hlavní cyklus úlohy

V metodě *run* je volána funkce *doCycle*, ve které se vykonává celý program kromě obslužných metod reagujících na události. Metoda nejdříve získá hodnotu aktuálního výstupu *y* soustavy a předá ji jako parametr metodě regulátoru, která následně vrátí vypočítaný akční zásah *u*. Ve zbývající části metody se nastavují žádané průběhy veličin s ohledem na to, zda jsou nastavovány v manuálním či automatickém režimu. Výjimku představuje zadávání žádaného akčního zásahu, které je na rozdíl od ostatních veličin ještě vnořeno do další podmínky. Tato podmínka ověřuje, zda je aktuálně k soustavě přiřazený manuální regulátor na základě názvu konstruktoru regulátoru (viz Výpis č. 14).

```
if (this.m_Controller.constructor.name == "manualController") {
        if (U_input.manMode == true && U_input.autoMode == false) {
            let userVal = parseFloat(document.getElementById("U-user").value);
            if (!isNaN(userVal)) {
                userVal = limit(userVal, par_UMin_val, par_UMax_val);
                this.m_Controller.m_fU = parseFloat(userVal);
                document.getElementById("U-val").innerHTML = userVal;
}
}
}
```

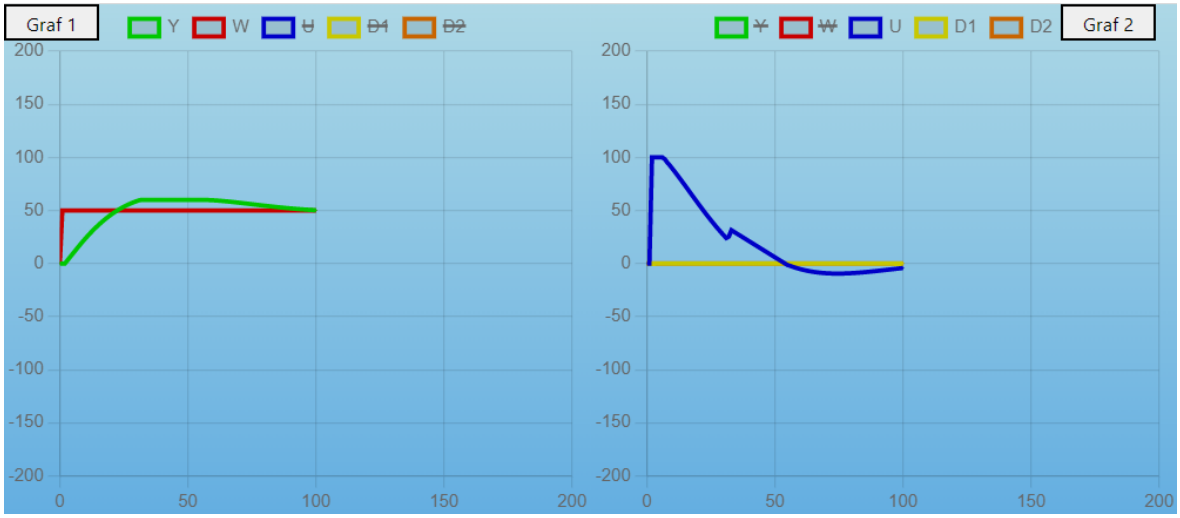
Výpis č. 14: Načítání akčního zásahu zadaného uživatelem

Na konci metody *doCycle* se aktualizují výstupní grafy. V metodě *run* je ještě provedeno zastavení programu na 85 milisekund, čímž je docíleno dobré shody reálného času s časem

programu, protože jeden průběh hlavního cyklu poté trvá téměř desetinu vteřiny. Tato hodnota byla určena empiricky.

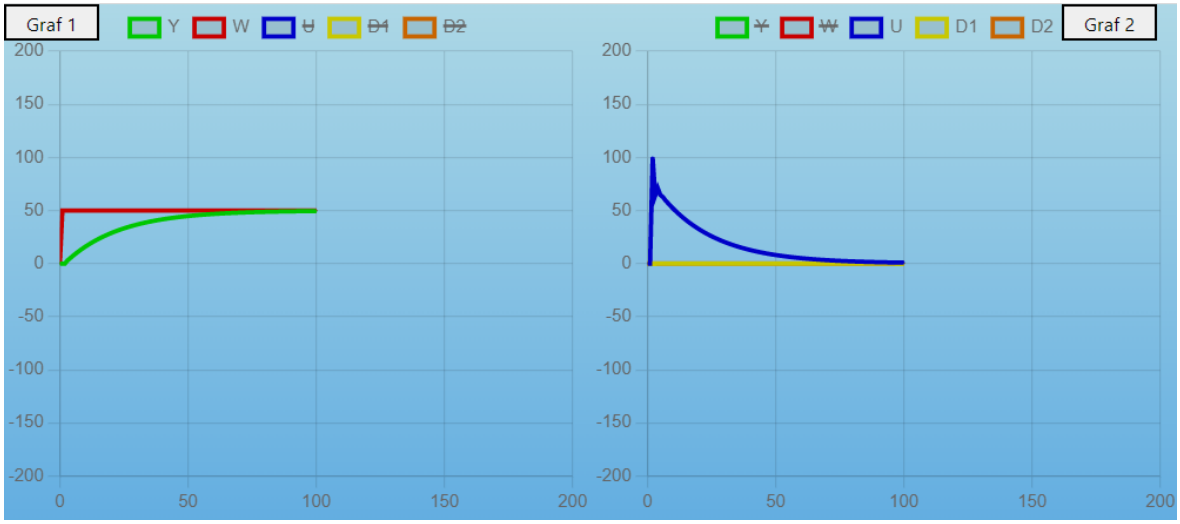
6.5 Příklad využití konkrétní úlohy

Použití konkrétní úlohy jsem se rozhodl ukázat na soustavě vodní nádrže s regulovaným přítokem i odtokem a poruchovou veličinou představující neregulovaný odtok. Cílů v každé úloze může být mnoho, ale pro příklad uvádím regulaci na konkrétní úroveň hladiny s požadavkem regulace bez překročení žádané hladiny. Žádaná hodnota hladiny v tomto případě byla 50. K soustavě jsem nejprve připojil PID regulátor s koeficienty P = 3, Ti = 20s a Td = 5s. Na obrázku č. 18 je vidět průběh výstupu společně s úrovní žádané hladiny. Výstup po 25 časových vzorcích překročil žádanou hodnotu a dalších 75 vzorků trvalo navrácení výstupu na úroveň žádané hodnoty. Tento průběh nesplnil základní požadavek pro nepřekročení žádané hladiny a doba regulace je navíc velice dlouhá.



Obrázek č. 18: Regulování hladiny PID regulátorem

Jako další jsem zvolil PD regulátor s parametry P = 2 a Td = 5s. Průběh výstupu po zapojení tohoto PD regulátoru je na obrázku č. 19. Tento průběh splňuje požadavek na nepřekročení regulované hladiny, ale je z něj také patrné, že se snižující regulační odchylkou klesá i akční zásah z důvodu chybějící integrační složky. Ta však může způsobit překročení regulované hladiny, což je vidět v předchozím průběhu s připojeným PID regulátorem.



Obrázek č. 19: Regulování hladiny PD regulátorem

7 Závěr

Nová virtuální laboratoř není pouze původní virtuální laboratoř přepsaná do jiného jazyka. V průběhu vývoje se objevilo mnoho faktorů mající za následek dodatečné úpravy a často i odklon od původní virtuální laboratoře. Tyto úpravy však většinou nebyly kompromisy, ale spíše vylepšení a umožnily zavedení nových prvků. Některé knihovny a možnosti jazyka JavaScript nabízí funkce, které původní laboratoř neobsahovala, a proto se s nimi ani v počátku nepočítalo. Celkově byl kladen důraz na zpříjemnění a zatraktivnění výuky základů automatického řízení pro studenty, kteří se s řízením dosud nesetkali nebo pouze omezeně.

Cílem exportu parametrů regulátoru a souřadnic grafu není získání konkrétních dat, ale zejména umožnit efektivnější výuku i v případě nutnosti distanční výuky. Vyučující má možnost studentům zaslat pouze grafický výstup a zadat práci v podobě úkolu naladit regulátor a získat stejnou či podobnou odezvu soustavy. Zároveň umožňuje si práci uložit a pokračovat v činnosti později, a to i na jiném zařízení v případě přenesení exportovaného souboru.

Hlavní motivací pro vytvoření virtuální laboratoře byla možnost seznámit studenty s řízením soustav zábavnou a interaktivní formou. Podobné úkoly může učitel zadat studentům ještě před výukou v reálné laboratoři. Studenti si vyzkouší chování soustavy s různými nastaveními regulátorů a nemusí následně v laboratoři trávit čas zkoumáním základních typů regulátorů a jejich vlastností.

I přes naplnění všech stanovených cílů se v průběhu tvorby objevilo několik dalších možností na celkové zlepšení, které by však vyžadovaly relativně veliké množství dodatečné práce. Jedná se zejména o přidání responzivního zobrazení pro zařízení s menšími displeji a zajištění komunikace s externím zařízením (spojení s reálnou úlohou). Podrobnější seznam možných úprav je v obsažen v příloze společně s celým zdrojovým kódem.

Při konzultacích s panem profesorem Hofreiterem jsme došli k závěru, že by se pro pokračování v tvorbě vypsalo další téma diplomové práce. Zároveň jsem přislíbil konzultační pomoc případnému pokračovateli.

Použité zdroje

[1] FLANAGAN, David. *JavaScript: The Definitive Guide*. O'Reilly Media. 6th Edition. Sebastopol, CA: O'Reilly Media, 2011. 1098 s. ISBN 978-0-596-80552-4.

[2] SHARMA, Kumar Chetan; Stefanov, Stoyan; Antani, Ved; Hillar, Gastón Carlos. *JavaScript: Object Oriented Programming.* Birmingham: Packt Publishing, 2016. 824 s. ISBN 978-1-78712-359-5.

[3] Scheduling: setTimeout and setInterval. *JavaScript.info* [online]. 2022 [cit. 2022-05-02]. Dostupné z: https://javascript.info/settimeout-setinterval.

[4] Differences Between Programming vs Scripting. *Educba* [online]. 2022 [cit. 2022-04-02]. Dostupné z: https://www.educba.com/programming-vs-scripting/.

[5] SCHILDT, Herbert. *Java: The Complete Reference, Elventh Edition*. 11th edition. McGraw-Hill Education, 2019. ISBN 978-1-260-44023-2.

[6] About JavaScript. *MDN* [online]. 2022 [cit. 2022-05-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

[7] DUCKETT, Jon. *HTML & CSS: design and build websites*. Indianapolis, IN: Wiley, [2011]. ISBN 978-1-118-00818-8.

[8] *Learning Web Design - A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics* [online]. 5th. O'Reilly Media, 2018 [cit. 2022-05-13]. ISBN 978-1-491-96020-2. Dostupné z: https://1lib.cz/book/3556226/072a01.

[9] HOFREITER, Milan. *Základy automatického řízení*. V Praze: České vysoké učení technické, 2012. ISBN 978-80-01-05007-1.

[10] *Samson group* [online]. [cit. 2022-05-04]. Dostupné z: https://www. samsongroup.com/en/

[11] *Chart.js* [online]. [cit. 2022-05-13]. Dostupné z: https://www.chartjs.org/

[12] *Hammer.js* [online]. [cit. 2022-05-13]. Dostupné z: https://hammerjs.github.io/

[13] *Papaparse.js* [online]. [cit. 2022-05-14]. Dostupné z: https://www. papaparse.com/

[14] *Bootstrap.css* [online]. [cit. 2022-05-14]. Dostupné z: https://getbootstrap. com/docs/3.4/css/

[15] *Anime.js* [online]. [cit. 2022-05-23]. Dostupné z: https://animejs.com/

[16] GSAP. *Greensock* [online]. [cit. 2022-05-23]. Dostupné z: https://greensock. com/gsap/

[17] *Kute.js* [online]. [cit. 2022-06-03]. Dostupné z: https://thednp. github.io/kute.js/

Seznam obrázků

Obrázek č. 1: Struktura z pohledu HTML 18
Obrázek č. 2: Struktura z pohledu DOM 19
Obrázek č. 3: Vizualizace pomocí SVG 21
Obrázek č. 4: Vizualizace v elementu canvas 22
Obrázek č. 5: Regulační obvod 25
Obrázek č. 6: Původní virtuální laboratoř 27
Obrázek č. 7: Interaktivní graf 33
Obrázek č. 8: Rozložení stránky 42
Obrázek č. 9: Výstupní graf 43
Obrázek č. 10: Oddíl ukládání dat a volby jazyka 44
Obrázek č. 11: Úloha se skrytým rozšířeným nastavením 44
Obrázek č. 12: Regulační část 45
Obrázek č. 13: Parametry systému 46
Obrázek č. 14: Hlavní ovládací panel 46
Obrázek č. 15: Kontextové menu pro editaci bodů 47
Obrázek č. 16: Blok přednastavených funkcí 48
Obrázek č. 17: Ukázka využití přednastavené funkce skokové změny 48
Obrázek č. 18: Regulování hladiny PID regulátorem 50
Obrázek č. 19: Regulování hladiny PD regulátorem 51

Seznam výpisů zdrojových kódů

Výpis č. 1: Strukturování oddílu do tabulkového zobrazení 16
Výpis č. 2: Animování SVG 22
Výpis č. 3: Metoda UpdateSystem 29
Výpis č. 4: Příklad přiřazení obslužné metody k tlačítku 31
Výpis č. 5: Nastavení datového souboru výstupní veličiny v grafu 32
Výpis č. 6: Ukázka přidání pluginů ke grafu 32
Výpis č. 7: Přidání nového bodu při aktivním vyčítání hodnoty 34
Výpis č. 8: Funkce pro tažení bodu v grafu 36
Výpis č. 9: Ukázka změny jazyka 36
Výpis č. 10: Struktura ukládaného souboru 38
Výpis č. 11: Vytvoření systému pro konkrétní úlohu 39
Výpis č. 12: Načítání dat ze souboru 41
Výpis č. 13: Hlavní cyklus úlohy 49
Výpis č. 14: Načítání akčního zásahu zadaného uživatelem 49

Seznam příloh na přiloženém optickém nosiči

Příloha č. 1: Anotace v českém jazyce

Příloha č. 2: Anotace v anglickém jazyce

Příloha č. 3: Diplomová práce ve formátu PDF

Příloha č. 4: Zdrojové kódy původní virtuální laboratoře

Příloha č. 5: Zdrojové kódy nové virtuální laboratoře