Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky a řídicí techniky

Diplomová práce

Interaktivní virtuální laboratoře prezentující techniky tlumení vibrací

Plzeň, 2012 Jan Reitinger

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne..... .....

Poděkování

Chtěl bych poděkovat vedoucímu své diplomové práce Ing. Martinu Čechovi, Ph.D. za cenné rady, připomínky, metodické vedení a trpělivost při vypracování práce.

Abstrakt

Tato práce popisuje tvorbu virtuální laboratoře pro znázornění technik tlumení vibrací na mechanickém systému, konkrétně na portálovém jeřábu s nákladem uchyceným na laně. V práci jsou odvozeny matematické modely systému jeřábu a motorů k jeho řízení. Tyto modely jsou následně nasimulovány v řídicím systému REX. Je zde také představena technika tlumení vibrací pomocí tvarování vstupu, která je použita pro potlačení nežádaných kmitů. Poté následuje detailní popis, jak simulaci převést do jazyka Java a jak z ní vytvořit jádro virtuální laboratoře. Na konec je představena možnost, jak vytvořit interaktivní trojrozměrný model systému ve VRML, který se poté převede do balíku Java 3D.

Klíčová slova: Virtuální laboratoř, Tlumení vibrací, HMI, Portálový jeřáb, Java, REX, JavaREX, Input shaping filtr, VRML, Java 3D

Abstract

This thesis describes the full development of Virtual Laboratory presenting vibration damping techniques on mechanical models. Mores specifically, a gantry crane with load on rope will be considered as oscillatory process. Mathematical model of both crane system and motion drives will be derived. Afterwards the model and control system will be implemented in the control system REX. Special input shaping filter will be employed for damping of unpleasant oscillations. Next, the automatic transformation of simulation into Java language and creation of Virtual Laboratory core will be documented. Finally, the creation of interactive 3D model of crane system in VRML will be illustrated. The model is rendered using Java3D package.

Keywords: Virtual Laboratory, Vibration damping, HMI, Gantry crane, Java, REX, JavaREX, Input shaping filter, VRML, Java 3D

Obsah

Kapitola 1

Úvod

Tato práce se bude zabývat tvorbou virtuální laboratoře prezentující techniky tlumení vibrací. Virtuální laboratoř si může každý představit jako pokročilý program, který provádí simulaci nějakého děje (pokusu) a výsledky přehledně zobrazuje uživateli. Uživatel může různě měnit vstupní podmínky pokusu a pozorovat závislost změny výstupů. Na rozdíl od klasické simulace může navíc uživatel zasahovat do děje a pozorovat důsledky svých změn.

Virtuální laboratoře jsou v poslední době čím dál populárnější. To je způsobeno hlavně jejich interaktivitou, využíváním moderních prostředků a tím pádem i větší atraktivitou pro uživatele. Dnes je možné na internetu nalézt nepřeberné množství fyzikálních laboratoří určených studentům různých škol. Největší výhodou těchto laboratoří je fakt, že stejný pokus může současně provádět mnoho studentů, aniž by měla škola potřebný počet pokusných systémů. Tím pádem dochází k výrazné úspoře finančních prostředků a uvolněné místo může být využito k jiným účelům. Toto samozřejmě nemůže zcela nahradit práci v reálné laboratoři, nicméně některé pokusy ve virtuální laboratoři mohou být dokonce názornější než pokusy skutečné.

Virtuální laboratoře navíc nevyžadují téměř žádné finance na údržbu. Jejich cena je prakticky pouze součtem nákladů na naprogramování takovéto laboratoře a nákladů na počítač, na kterém laboratoř bude fungovat. Odpadá tedy dokupování použitého, či opotřebovaného materiálu.

Použití virtuálních laboratoří se však v žádném případě neomezuje pouze na výuku na školách. Poměrně často jsou tyto laboratoře používány k zaškolování nové obsluhy zařízení, u kterých hrozí, že by mohlo dojít neodbornou manipulací k velkým škodám na výrobku, či na výrobním zařízení.

Další možností pro využití virtuálních laboratoří jsou obory, kdy je potřeba provést pokus, který jednoduše v realitě provést nejde. Lze si totiž jen těžko představit, že by někdo strhl přehradu kvůli sledování různých krizových scénářů, či někdo provedl ve skutečnosti srážku dvou galaxií.

Cílem laboratoře vytvořené v této práci je prezentovat techniky tlumení vibrací u mechanických systémů. Na příkladu je ukázáno, jak postupovat od vytvoření matematického modelu řízeného systému přes modelování ovládacích motorů, nastavo-

vání jejich regulátorů, nastavování parametrů tlumícího bloku až k tvorbě virtuální laboratoře, simulacím a testování správnosti vytvořeného systému.

Jako vhodný příklad mechanického systému byl vybrán portálový jeřáb. Na tomto příkladu jsou názorně vidět výsledky tlumení vibrací. Takovéto jeřáby se využívají například ve skladech pro přemisťování materiálů. Jeřáb se pohybuje v rovině dvěma směry (v ose X a ose Y) a při pohybu může dojít k nežádoucímu rozkmitání přesouvaného nákladu, který tak může narazit do blízko stojícího objektu. Případně při rozkmitání nákladu musí obsluha jeřábu dlouho čekat, než se náklad ustálí, a bude ho moci uložit. Portálový jeřáb lze reprezentovat jako vozík pohybující se v rovině, na jehož spodní straně je zavěšeno kyvadlo. Cílem úlohy tedy je zajistit co nejmenší kmitání tohoto kyvadla.

Ovládání vozíku je zajištěno dvěma stejnosměrnými motory (jeden pro každou osu pohybu), které jsou nastavovány polohou.

Tlumení vibrací je v této práci vykonáváno pomocí bloku ZV4IS z řídícího systému REX. REX je blíže představen v subsekci 2.1.3 a techniky tlumení, které blok ZV4IS využívá, jsou popsány v sekci 2.4. Zde stačí pouze říci, že blok pracuje na principu tzv. input shapingu (tvarování vstupu).

Vzhledem k tomu, že k tlumení vibrací se využívá blok z řídicího systému REX, je vhodné i pro modely řízené soustavy (vozíku s kyvadlem) a řídicí soustavy (motorů s regulátory) využít tento systém. Namodelování soustavy v REXu má navíc tu výhodu, že jí lze pak snadno převést na odpovídající simulační jádro virtuální laboratoře v jazyce Java.

Kapitola 2

Modelování systému

Jako vhodný příklad, na kterém lze prezentovat způsoby tlumení vibrací pomocí input shapingu, bylo vybráno prostorové kyvadlo, které je umístěno na vozíku, který se pohybuje ve dvou osách. Vozík je ovládán dvěma motory, kterým je možné pomocí soustavy regulátorů nastavit aktuální pozici natočení hřídele a tím i pozici vozíku. Tato pozice je nastavována buď přímo uživatelem a nebo při zapnutém tlumení tlumícím filtrem tak, aby kyvadlo kmitalo co nejméně.

Modelování celého systému lze tedy z důvodu lepšího pochopení a testování rozpadnout do několika částí. První částí je modelování samotného řízeného systému, tedy portálového jeřábu. Druhou částí je vytvoření vhodného modelu motoru pro pohyb vozíku. Další je poté nastavení regulátorů pro motor, aby se motor choval podle našich představ. Poslední částí je správné zapojení tlumícího filtru a jeho nastavení.

Jelikož jsou pro případ netlumených vibrací předpokládány výchylky kyvadla větší než 5°, je vhodnější namodelovat systém jako nelineární. Modelování nelineárního systému je sice nepoměrně složitější, nicméně systém poté poskytuje daleko lepší výsledky, které daleko lépe reprezentují realitu.

K řízení byl použit stejnosměrný motor s buzením permanentními magnety. Motor s cizím buzením byl vybrán kvůli jeho ideální regulovatelnosti, která je zapříčiněna tím, že jeho otáčky lze snadno měnit změnou napětí na kotvě.

Motor je řízen polohou (uživatel bude zadávat souřadnice, kam má vozík dojet) a k návrhu řídicích regulátorů byla použita metoda robustních regionů a laboratoř PIDLab (viz [6]).

K aktivnímu tlumení vibrací je využit tvarovač vstupního signálu. Tento blok realizuje funkci frekvenčního filtru typu pásmová zádrž [15].

2.1 Vozík s kyvadlem a dvěma stupni volnosti

2.1.1 Lagrangeovská formulace mechaniky

Tato subsekce je vytvořena na základě [7] a [8].

Lagrangeovská formulace mechaniky nabízí jinou možnost popisování systémů než klasická Newtonovská mechanika. Newtonovy pohybové rovnice sice umožňují úplně popsat mechanický pohyb, z matematického hlediska je však často lepší využít Lagrangeovy rovnice.

Tento přístup využívá k popisu systému zobecněné souřadnice, což zjednodušuje analýzu systému. Trajektorie pohybu se získávají řešením Lagrangeových rovnic, které vycházejí z variačního počtu. Formálně řečeno: Řešení Lagrangeovy rovnice představuje nalezení cesty, která minimalizuje funkcionál akce, což je veličina, která je integrálem lagrangiánu.

Z výše uvedeného se dá říct, že Lagrangeovy rovnice jsou zobecněním Newtonových rovnic, protože umožňují definovat matematické vztahy pro pohyb i v oblastech, kde Newtonovy rovnice nedávají smysl. To je způsobeno zejména tím, že umožňují jednodušší přepis rovnic do křivočarých souřadnic.

Lagrangeovy rovnice mohou být dvou druhů. Rovnice prvního druhu plynou z d'Alembertova principu, který zobecňují a s využitím tzv. Lagrangeových multiplikátorů definují pohybové rovnice soustavy hmotných bodů. Rovnice druhého druhu využívají k vytvoření pohybových rovnic soustavy hmotných bodů tzv. zobecněné souřadnice. V této práci jsou použity Lagrangeovy rovnice druhého druhu, jelikož povedou k jednodušším výrazům vzhledem k tomu, že je vhodnější zapsat souřadnice kyvadla pomocí zobecněných souřadnic.

Lagrangeovy rovnice druhého druhu mají obecně tvar

$$\frac{d}{dt}\left(\frac{\partial L}{\partial q_i'}\right) - \left(\frac{\partial L}{\partial q_i}\right) = Q_i \quad 1 \leq i \ \leq n \tag{2.1}$$

kde proměnné rovnice jsou:

$L = T - V$		je Lagrangián,
T		je kinetická energie vzhledem k inerciální soustavě,
V		je potenciální energie,
n		je počet stupňů volnosti,
q_i	$i = 1, \ldots, n$	jsou zobecněné souřadnice,
Q_i	$i = 1, \ldots, n$	jsou zobecněné síly.

Derivování vzhledem k q_i' se provádí za předpokladu, že všechny ostatní q_j' a q_j jsou konstantní. Podobné pravidlo platí pro derivování podle q_i.

2.1.2 Matematický model

K odvození pohybových rovnic byla využita Lagrangeova metoda.

Vzhledem k tomu, že se vozík může pohybovat pouze v rovině XY, je jeho poloha dána souřadnicemi [x1; y1; 0]. Hmotný bod na konci kyvadla se již pohybuje ve všech třech osách a jeho souřadnicový systém se obecně určí jako [x2; y2; z2], kde souřadnice hmotného bodu na konci kyvadla lze vyjádřit pomocí souřadnic vozíku jako

$$x2 = x1 + l\cos\varphi, \tag{2.2}$$
$$y2 = y1 + l\cos\psi, \tag{2.3}$$
$$z2 = -l\sqrt{1 - \cos^2\varphi - \cos^2\psi}. \tag{2.4}$$

Úhel φ je výchylka kyvadla od osy x, úhel ψ je výchylka kyvadla od osy y a l je délka kyvadla (viz Obrázek 2.1).

Nutno uvést, že vzorec souřadnice $z2$ není přesný, neboť takto popisuje pouze dolní polokouli možného pohybu kyvadla. Úplný vzorec je

$$z2 = -l \cdot \pm\sqrt{1 - \cos^2\varphi - \cos^2\psi},$$

kde kladné znaménko u odmocniny definuje dolní polokouli a záporné tu horní. Předpokladem je, že klasická manipulace jeřábem nedostane kyvadlo do horní polohy, a proto bylo zavedeno toto zjednodušení.

Obrázek 2.1: Vyznačení souřadnic soustavy v prostoru

Lagrangián je obecně dán jako

$$L = T1 + T2 - V1 - V2$$

kde T je kinetická energie vozíku a kyvadla, V jsou potenciální energie těchto hmotných bodů. Potenciální energie vozíku $V1$ je nulová. Dále je uvažováno takzvané řízení polohou. Jinými slovy systém je řízen pomocí vstupů $x1$ a $y1$. Lagrangián tedy přejde na tvar

$$L = T2 - V2 \tag{2.5}$$

Potenciální energie hmotného bodu na konci kyvadla je dána vztahem

$$V2 = mgz2 = -mgl\sqrt{1 - \cos^2\varphi - \cos^2\psi}. \tag{2.6}$$

Kinetickou energii konce kyvadla lze určit ze vztahu

$$T2 = \frac{1}{2}mv^2, \tag{2.7}$$

kde v je okamžitá rychlost tohoto hmotného bodu. Po dosazení přechází do tvaru

$$v^2 = (x2'^2 + y2'^2 + z2'^2). \tag{2.8}$$

Poloha je závislá na souřadnicích vozíku a výše definovaných výchylkách. Po dosazení z rovnic (2.2-2.4) a následné derivací přecházejí výrazy na tvar

$$x2' = (x1 + l\cos\varphi)' = x1' - l\sin\varphi\varphi', \tag{2.9}$$

$$y2' = (y1 + l\cos\psi)' = y1' - l\sin\psi\psi', \tag{2.10}$$

$$z2' = (-l\sqrt{1 - \cos^2\varphi - \cos^2\psi})' = \frac{-l(\cos\varphi\sin\varphi\varphi' + \cos\psi\sin\psi\psi')}{\sqrt{1 - \cos^2\varphi - \cos^2\psi}}. \tag{2.11}$$

Dosazením (2.9-2.11) do (2.8) a úpravou vychází kvadrát rychlosti jako

$$\begin{aligned} v^2 = \frac{1}{-1 + \cos^2\varphi + \cos^2\psi}(&-x1'^2 + x1'^2\cos^2\varphi + x1'^2\cos^2\psi + \\ &+2x1'l\sin\varphi\varphi' - 2x1'l\sin\varphi\varphi'\cos^2\varphi - 2x1'l\sin\varphi\varphi'\cos^2\psi - \\ &-l^2\varphi'^2 + l^2\varphi'^2\cos^2\varphi + l^2\varphi'^2\cos^2\psi - l^2\varphi'^2\cos^2\psi\cos^2\varphi - y1'^2 + \\ &+y1'^2\cos^2\varphi + y1'^2\cos^2\psi + 2y1'l\sin\psi\psi' - 2y1'l\sin\psi\psi'\cos^2\varphi - \\ &-2y1'l\sin\psi\psi'\cos^2\psi - l^2\psi'^2 + l^2\psi'^2\cos^2\psi + l^2\psi'^2\cos^2\varphi - \\ &-l^2\psi'^2\cos^2\varphi\cos^2\psi - 2l^2\cos\varphi\sin\varphi\varphi'\cos\psi\sin\psi\psi'). \end{aligned} \tag{2.12}$$

Kvadrát rychlosti (2.12) byl dosazen do vzorce kinetické energie (2.7)

$$\begin{aligned} T2 = \frac{1}{2}\frac{1}{-1 + \cos^2\varphi + \cos^2\psi}m(&-x1'^2 + x1'^2\cos^2\varphi + x1'^2\cos^2\psi + \\ &+2x1'l\sin\varphi\varphi' - 2x1'l\sin\varphi\varphi'\cos^2\varphi - 2x1'l\sin\varphi\varphi'\cos^2\psi - \\ &-l^2\varphi'^2 + l^2\varphi'^2\cos^2\varphi + l^2\varphi'^2\cos^2\psi - l^2\varphi'^2\cos^2\psi\cos^2\varphi - y1'^2 + \\ &+y1'^2\cos^2\varphi + y1'^2\cos^2\psi + 2y1'l\sin\psi\psi' - 2y1'l\sin\psi\psi'\cos^2\varphi - \\ &-2y1'l\sin\psi\psi'\cos^2\psi - l^2\psi'^2 + l^2\psi'^2\cos^2\psi + l^2\psi'^2\cos^2\varphi - \\ &-l^2\psi'^2\cos^2\varphi\cos^2\psi - 2l^2\cos\varphi\sin\varphi\varphi'\cos\psi\sin\psi\psi'). \end{aligned} \tag{2.13}$$

Nyní již lze konečně dosazením (2.6) a (2.13) do (2.5) určit finální podobu Lagrangiánu

$$\begin{aligned} L = \frac{1}{2}\frac{1}{-1+\cos^2\varphi+\cos^2\psi} m(&-x1'^2 + x1'^2\cos^2\varphi + x1'^2\cos^2\psi + \\ &+2x1'l\sin\varphi\varphi' - 2x1'l\sin\varphi\varphi'\cos^2\varphi - 2x1'l\sin\varphi\varphi'\cos^2\psi - \\ &-l^2\varphi'^2 + l^2\varphi'^2\cos^2\varphi + l^2\varphi'^2\cos^2\psi - l^2\varphi'^2\cos^2\psi\cos^2\varphi - y1'^2 + \\ &+y1'^2\cos^2\varphi + y1'^2\cos^2\psi + 2y1'l\sin\psi\psi' - 2y1'l\sin\psi\psi'\cos^2\varphi - \\ &-2y1'l\sin\psi\psi'\cos^2\psi - l^2\psi'^2 + l^2\psi'^2\cos^2\psi + l^2\psi'^2\cos^2\varphi - \\ &-l^2\psi'^2\cos^2\varphi\cos^2\psi - 2l^2\cos\varphi\sin\varphi\varphi'\cos\psi\sin\psi\psi') + \\ &+mgl\sqrt{1-\cos^2\varphi-\cos^2\psi}. \end{aligned} \tag{2.14}$$

Systém má dva stupně volnosti a existují tedy dvě Lagrangeovy rovnice. Vzhledem k řízení modelu polohou jsou všechny zobecněné síly Q_i nulové a pravé strany Lagrangeových rovnic jsou rovny nule. Po dosazení z 2.1 jsou tedy získány rovnice

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \varphi'}\right) - \left(\frac{\partial L}{\partial \varphi}\right) = 0 \tag{2.15}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \psi'}\right) - \left(\frac{\partial L}{\partial \psi}\right) = 0 \tag{2.16}$$

Je možné na pravé strany dosadit tlumící síly, nicméně tlumení je raději realizováno později v simulačním modelu. Výhodou tohoto řešení je, že lze případně snáze měnit hodnotu třecího koeficientu.

Dosazením do (2.15), zderivováním a upravením výrazu vychází

$$
\begin{aligned}
-(ml(l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi'^2\cos^2\psi\cos\varphi\sin\varphi\ (2.17)\\
-2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos\varphi\sin\varphi\cos^2\psi\psi'^2-\\
-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi'^2\cos^4\psi\cos\varphi\sin\varphi+\\
+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos^3\varphi\sin\varphi\cos^2\psi\psi'^2+\\
+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos\varphi\sin\varphi\cos^4\psi\psi'^2+\\
+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi'\cos\psi\cos^2\varphi\sin\psi\psi'-\\
-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos\varphi\sin\varphi\cos\psi\sin\psi\psi''-\\
-2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi'\cos\psi\cos^4\varphi\sin\psi\psi'+\\
+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos^3\varphi\sin\varphi\cos\psi\sin\psi\psi''+\\
+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos\varphi\sin\varphi\cos^3\psi\sin\psi\psi''+\\
+2\sqrt{1-\cos^2\varphi-\cos^2\psi}x1''\sin\varphi\cos^2\varphi\cos^2\psi-\\
-3l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''\cos^2\psi\cos^2\varphi+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''\cos^2\psi\cos^4\varphi+\\
+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''\cos^4\psi\cos^2\varphi+\sqrt{1-\cos^2\varphi-\cos^2\psi}x1''\sin\varphi\cos^4\varphi+\\
+\sqrt{1-\cos^2\varphi-\cos^2\psi}x1''\sin\varphi\cos^4\psi-2\sqrt{1-\cos^2\varphi-\cos^2\psi}x1''\sin\varphi\cos^2\varphi-\\
-2\sqrt{1-\cos^2\varphi-\cos^2\psi}x1''\sin\varphi\cos^2\psi-2g\cos\varphi\sin\varphi\cos^2\psi+\\
+2g\cos^3\varphi\sin\varphi\cos^2\psi++g\cos\varphi\sin\varphi\cos^4\psi+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''\cos^2\varphi+\\
+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''\cos^2\psi-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''\cos^4\varphi-\\
-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''\cos^4\psi+g\cos\varphi\sin\varphi+\sqrt{1-\cos^2\varphi-\cos^2\psi}x1''\sin\varphi-\\
-2g\cos^3\varphi\sin\varphi+g\cos^5\varphi\sin\varphi-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi''+\\
+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos\varphi\sin\varphi\psi'^2-\\
-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos^3\varphi\sin\varphi\psi'^2))/(\sqrt{1-\cos^2\varphi-\cos^2\psi}\\
(1-2\cos^2\varphi-2\cos^2\psi+\cos^4\varphi+2\cos^2\varphi\cos^2\psi+\cos^4\psi))=0
\end{aligned}
$$

Obdobný vztah je získán pro (2.16)

$$
\begin{aligned}
&-(ml(-3l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''\cos^2\varphi\cos^2\psi+ \\
&+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''\cos^4\varphi\cos^2\psi+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''\cos^2\varphi\cos^4\psi+ \\
&+2\sqrt{1-\cos^2\varphi-\cos^2\psi}y1''\sin\psi\cos^2\varphi\cos^2\psi+ \\
&+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi'^2\cos^2\varphi\cos\psi\sin\psi- \\
&-2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos^2\varphi\varphi'^2\cos\psi\sin\psi- \\
&-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi'^2\cos^4\varphi\cos\psi\sin\psi+ \\
&+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos^4\varphi\varphi'^2\cos\psi\sin\psi+ \\
&+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos^2\varphi\varphi'^2\cos^3\psi\sin\psi-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''+ \\
&+\sqrt{1-\cos^2\varphi-\cos^2\psi}y1''\sin\psi-2g\cos^3\psi\sin\psi+g\cos^5\psi\sin\psi+ \\
&+g\cos\psi\sin\psi+\sqrt{1-\cos^2\varphi-\cos^2\psi}y1''\sin\psi\cos^4\varphi+ \\
&+\sqrt{1-\cos^2\varphi-\cos^2\psi}y1''\sin\psi\cos^4\psi-2\sqrt{1-\cos^2\varphi-\cos^2\psi}y1''\sin\psi\cos^2\varphi- \\
&-2\sqrt{1-\cos^2\varphi-\cos^2\psi}y1''\sin\psi\cos^2\psi- \\
&-2g\cos\psi\sin\psi\cos^2\varphi+g\cos\psi\sin\psi\cos^4\varphi+2g\cos^3\psi\sin\psi\cos^2\varphi+ \\
&+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''\cos^2\psi+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''\cos^2\varphi- \\
&-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''\cos^4\psi-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi''\cos^4\varphi+ \\
&+2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi'\cos\varphi\cos^2\psi\sin\varphi\varphi'- \\
&-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos\varphi\sin\varphi\varphi''\cos\psi\sin\psi- \\
&-2l\sqrt{1-\cos^2\varphi-\cos^2\psi}\psi'\cos\varphi\cos^4\psi\sin\varphi\varphi'+ \\
&+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos^3\varphi\sin\varphi\varphi''\cos\psi\sin\psi+ \\
&+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\cos\varphi\sin\varphi\varphi''\cos^3\psi\sin\psi+ \\
&+l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi'^2\cos\psi\sin\psi- \\
&-l\sqrt{1-\cos^2\varphi-\cos^2\psi}\varphi'^2\cos^3\psi\sin\psi))/(\sqrt{1-\cos^2\varphi-\cos^2\psi} \\
&(1-2\cos^2\varphi-2\cos^2\psi+\cos^4\varphi+2\cos^2\varphi\cos^2\psi+\cos^4\psi))=0.
\end{aligned}
\tag{2.18}
$$

Nyní již lze z rovnic vyjádřit druhé derivace úhlů φ, ψ

$$\varphi'' = -(-\sqrt{-1+\sin^2\varphi+\sin^2\psi}x1''\sin\varphi^2\sin\psi^2 + \tag{2.19}$$
$$-2l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\varphi'\cos\psi\sin\psi\psi'\sin\varphi + \sqrt{-1+\sin^2\varphi+\sin^2\psi}x1''\sin\varphi^2 +$$
$$+2\sqrt{-1+\sin^2\varphi+\sin^2\psi}l\varphi'\cos\psi\sin\psi\psi'\sin\varphi^3 -$$
$$-\sqrt{-1+\sin^2\varphi+\sin^2\psi}x1''\sin\varphi^4 + 2g\cos\varphi\sin\varphi^2 + 2\cos\varphi g\sin\psi^2 - \cos\varphi g +$$
$$+l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\varphi'^2\cos\varphi\sin\varphi^2 + l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\varphi'^2\cos\varphi\sin\psi^2 -$$
$$-l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\varphi'^2\cos\varphi - \cos\varphi g\sin\varphi^4 -$$
$$-\cos\varphi l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\sin\varphi^2\psi'^2\sin\psi^2 -$$
$$-2g\cos\varphi\sin\varphi^2\sin\psi^2 + \cos\varphi\sqrt{-1+\sin^2\varphi+\sin^2\psi}\cos\psi y1''\sin\varphi^2 +$$
$$+\cos\varphi\cos\psi\sqrt{-1+\sin^2\varphi+\sin^2\psi}y1''\sin\psi^2 -$$
$$-\cos\varphi\sqrt{-1+\sin^2\varphi+\sin^2\psi}\cos\psi y1'' - \cos\varphi g\sin\psi^4 -$$
$$-\cos\varphi l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\sin\varphi^2\varphi'^2\sin\psi^2)/(\sin\varphi(-1+\sin\varphi^2+\sin\psi^2)^{\frac{3}{2}}l)$$

$$\psi'' = -(-g\cos\psi\sin\varphi^4 - \cos\psi l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\sin\varphi^2\varphi'^2\sin\psi^2 + \tag{2.20}$$
$$+2\cos\psi g\sin^2\varphi + \cos\varphi\cos\psi\sqrt{-1+\sin^2\varphi+\sin^2\psi}x1''\sin\varphi^2 -$$
$$-l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\psi'^2\cos\psi\sin\varphi^2\sin\psi^2 +$$
$$+l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\psi'^2\cos\psi\sin\varphi^2 - 2\cos\psi g\sin\psi^2\sin\varphi^2 -$$
$$-\sqrt{-1+\sin^2\varphi+\sin^2\psi}y1''\sin\psi^2\sin\varphi^2 +$$
$$+2l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\psi'\cos\varphi\sin\varphi\varphi'\sin\psi^3 -$$
$$-2l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\psi'\cos\varphi\sin\varphi\varphi'\sin\psi - l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\psi'^2\cos\psi$$
$$+2\cos\psi g\sin\psi^2 - \cos\psi g + \cos\psi l\sqrt{-1+\sin^2\varphi+\sin^2\psi}\psi'^2\sin\psi^2 - \cos\psi g\sin\psi^4 +$$
$$+\sqrt{-1+\sin^2\varphi+\sin^2\psi}y1''\sin\psi^2 + \cos\varphi\cos\psi\sqrt{-1+\sin^2\varphi+\sin^2\psi}x1''\sin\psi^2 -$$
$$-\cos\varphi\cos\psi\sqrt{-1+\sin^2\varphi+\sin^2\psi}x1'' -$$
$$-\sqrt{-1+\sin^2\varphi+\sin^2\psi}y1''\sin\psi^4)/((-1+\sin\varphi^2+\sin\psi^2)^{\frac{3}{2}}l\sin^4\psi).$$

Rovnice (2.19) a (2.20) jsou rovnicemi zkoumaného nelineárního systému.

2.1.3 Řídicí systém REX

Řídicí systém REX (dále REX) je systém vyvíjený společností REX Controls s.r.o. Jedná se o otevřený a škálovatelný systém vhodný pro vnořené řízení (embedded control). Jedna z hlavních výhod REXu je, že je přenositelný na různé platformy s překladači jazyka C a C++ od jednoúčelových řídicích desek s jednoduchou exekutivou reálného času až po procesní stanice se standardními operačními systémy měkkého i pevného reálného času (Windows NT/2000/XP/Vista/7, Windows CE, Phar Lap ETS, apod.) [9].

REX byl od počátku vyvíjen tak, aby modelované systémy bylo možné spouštět i v prostředí Matlab/Simulink, které je v současné době velmi rozšířené. Modely jsou ukládány ve formátu *.mdl a lze tedy při simulaci a testování modelu využít všech možných prostředků Simulinku. Základy používání Simulinku se navíc vyučují na většině škol zaměřených na automatické řízení (kybernetiku). Potenciální uživatelé by tedy měli s REXem mít usnadněné začátky [9] a [10].

Řídicí algoritmy lze převést do binárních konfiguračních souborů (*.rex). Tyto soubory je následně možné poslat pomocí standardu TCP/IP do cílového zařízení, které může následně podle nich zahájit řízení, aniž by došlo k odstavení tohoto zařízení. Algoritmy je možné konfigurovat pomocí umisťování, nastavování a propojování funkčních bloků, které jsou umístěny v rozsáhlé knihovně RexLib (viz [11]). Tyto bloky jsou vytvořeny ve verzi jak pro Simulink, tak pro všechny cílové platformy, a tím je zajištěno ekvivalentní chování simulace [9] a [10].

Systém REX lze rozdělit do tří hlavních částí [10]:

- Host – Tato část slouží k obsluze běžícího systému, sledování jeho stavu, ale i například k jeho vytváření a testování. Aplikace této části zpravidla běží na procesních stanicích, neboť přes ně uživatel REX ovládá.
- Target (Core) – Jde o vlastní jádro systému, ve kterém probíhají veškeré výpočty a stará se o sběr informací z řízeného systému a jejich distribuci přes komunikační vrstvu. Taktéž samozřejmě provádí řízení systému.
- Komunikační vrstva – Propojuje části Host a Target. Nejčastěji ke komunikaci využívá protokol TCP/IP.

Mezi hlavní aplikace hostitelské části patří vývojové prostředí RexDraw, diagnostický nástroj RexView a kompilační program RexComp.

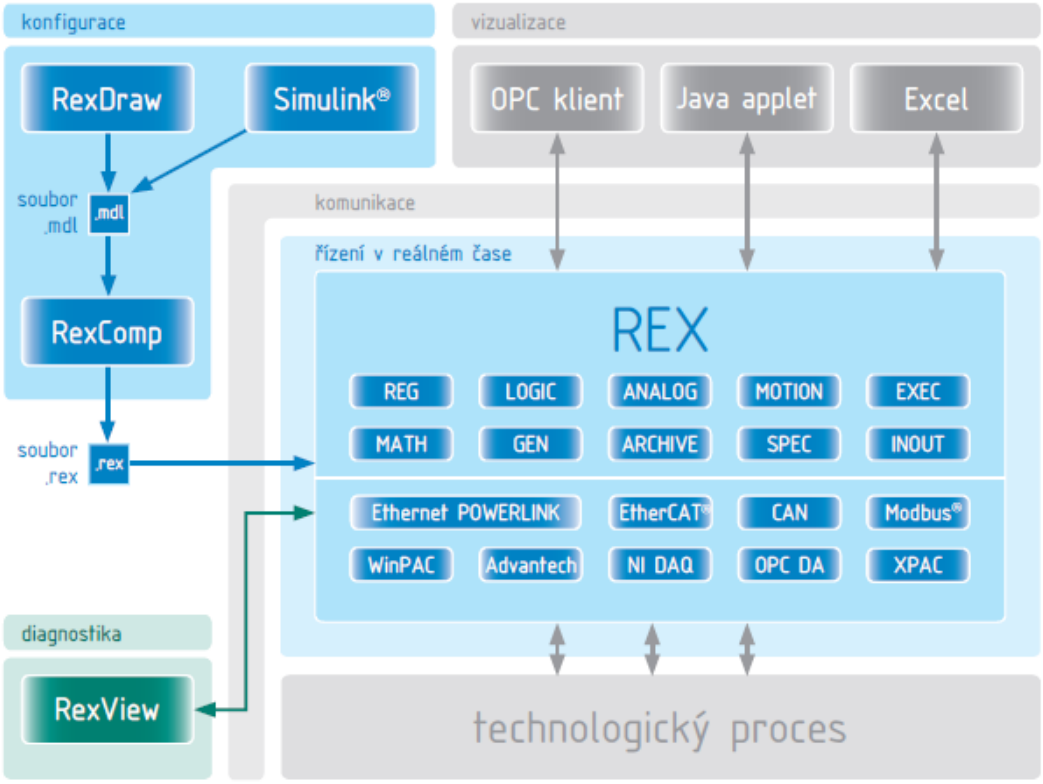
Aplikace RexDraw slouží k navrhování řídicích programů pomocí funkčních bloků z knihovny RexLib (bloky jsou popsány v [11]). Bloky pracují v diskrétním čase, přičemž jsou všechny automaticky diskretizovány pro konkrétní vzorkovací periodu. U každého vyvíjeného projektu je nutné, aby obsahoval minimálně dva soubory *.mdl. Jedním souborem je hlavní soubor (exekutiva), ve kterém se konfigurují jednotlivé úlohy, ovladače, priority, časování, atd. Další soubory obsahují jednotlivé algoritmy (tasky). Návod, jak vytvářet aplikace, lze nalézt v [9].

Program RexView je určen k sledování stavu jádra. S jeho pomocí lze číst a nastavovat hodnoty veličin, proto je důležitým nástrojem při testování a ladění projektů.

Je ale vhodný i k diagnostice chyb při rutinním provozu. RexView totiž zobrazuje běžící aplikaci i s jejím detailním hierarchickým uspořádáním. Díky protokolu TCP/IP si lze vybrat, zda se připojit k běžícímu jádru na lokálním počítači, v lokální síti, či ve vzdálené síti (přes Internet). [9]

RexComp slouží ke generování binárního konfiguračního souboru *.rex. Tento soubor je generován na základě hlavního souboru projektu (*.mdl), ve kterém jsou uloženy všechny informace o projektu (nastavení projektu a reference na jednotlivé tasky). RexComp vypisuje informace o generovaných souborech a případné chyby či upozornění při generování. V případě výskytu chyby se program ukončí a výsledný soubor vůbec nevytvoří. Překladač je možné z praktických důvodů spouštět z aplikace RexDraw pomocí menu Compiler/Compile. [9]

Schéma řídicího systému REX se všemi jeho základními částmi, knihovnami a podporovanými průmyslovými sběrnicemi a cílovými zařízeními je vidět na obrázku 2.2.



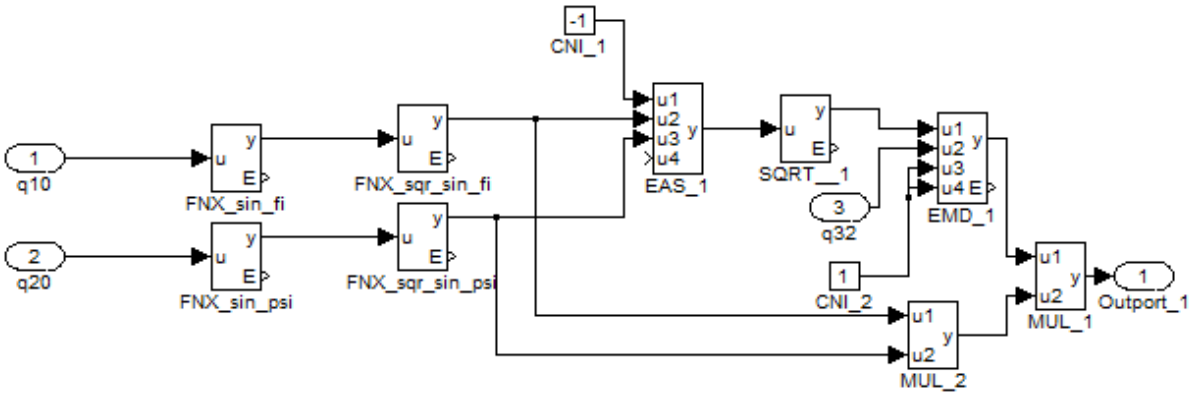
Obrázek 2.2: Struktura REXu (převzato z [30])

2.1.4 Simulační model

Simulace systému je prováděna pomocí řídicího systému REX. Hlavním důvodem pro vybrání tohoto systému je fakt, že výslednou simulaci lze snadno převést na jádro virtuální laboratoře v jazyce Java a pomocí množství tříd, které jsou v Javě již předdefinované, vytvořit laboratoř podle našich představ. REX je primárně určen ke vzdálenému řízení procesů. Z toho vyplývá, že díky volbě simulace v tomto systému může být v budoucnu z virtuální laboratoře vytvořena vzdálená laboratoř, pokud bude k dispozici reálný model jeřábu (rozdíl mezi virtuální a vzdálenou laboratoří je popsán v sekci 5.1). V podstatě by stačilo nahrát jádro do zařízení, které řídí fyzický model portálového jeřábu. Zařízení by poté s laboratoří komunikovalo pomocí protokolu TCP/IP.

Základním stavebním kamenem pro samotné modelování vozíku s kyvadlem budou rovnice (2.19) a (2.20). Tyto rovnice mohou být namodelovány buď pomocí bloků z knihovny MATH a nebo pomocí speciálního bloku REXLANG.

Využití knihovny MATH se však vzhledem ke složitosti systému ukázalo jako nevhodné. Takto namodelovaný systém měl přes 540 bloků a hledání chyb či modifikace systému byly časově velmi náročné. Příklad takto namodelovaného výrazu je na obrázku 2.3.



Obrázek 2.3: Subsystém pro výraz $-\sqrt{-1+\sin^2\varphi+\sin^2\psi}x1''\sin^2\varphi\sin^2\psi$

Vzhledem k tomu, že takovýchto systémů je pro každou rovnici dvacet, byla po čase raději vybrána varianta popisu rovnic pomocí bloků REXLANG.

Blok REXLANG je speciálně určen pro implementaci funkcí, které nelze efektivně vytvořit z tradičních bloků. K tomuto účelu blok využívá skriptovací jazyk, který je velmi podobný jazyku C. V praxi to znamená, že místo modelování funkce pomocí matematických bloků lze vytvořit podle této funkce skript, který REXLANG spouští a na základě vstupních hodnot generuje adekvátní výstupy. Bližší informace o bloku REXLANG i skriptovacím jazyku lze nalézt v [11].

Volba implementace matematického modelu pomocí REXLANG má tu nevýhodu, že ve skriptu, který automaticky převádí simulační model do programovacího

jazyka Java, není převod tohoto bloku hotový. Je tedy potřeba dopsat funkce tohoto bloku ručně. Podrobněji je tento problém a jeho řešení popsáno v sekci 5.5 a subsekci 5.7.1.

Funkce φ'' je definována skriptem model_kyvadla_q12.stl a funkce ψ'' skriptem model_kyvadla_q22.stl.

Vstupními proměnnými funkce jsou $\varphi, \varphi', \psi, \psi', x1'' \text{ a } y1''$. Hodnoty úhlů a jejich první derivace jsou získány zintegrováním výstupů funkcí a zavedením opět na vstup. Délka kyvadla je pro začátek zvolena jako $l = 1\,\mathrm{m}$. Ve virtuální laboratoři je poté možné délku kyvadla měnit, aby mohl být demonstrován způsob nastavení filtru tak, aby došlo k potlačení vibrací. Hodnota gravitačního zrychlení byla zvolena jako $g = 9{,}806\,65\,\mathrm{m\,.\,s^{-2}}$, což je dohodnutá střední hodnota normálního tíhového zrychlení (viz. [12]).

Výstupními veličinami modelu řízeného systému jsou φ'' a ψ''. Přičemž nižší derivace těchto veličin se pomocí zpětné vazby s integrátorem zavádějí na vstup systému. Jediné vstupy, kterými může být systém ovlivněn zvenčí, jsou tedy druhé derivace polohy vozíku (zrychlení vozíku) $x1''$ a $y1''$.

Vzhledem k tomu, že při matematickém modelu není neuvažován vliv tření ani odpor vzduchu, bylo do simulačního modelu zavedeno tzv. viskózní tření. Jelikož není k dispozici žádný reálný systém, podle kterého lze určit hodnotu tření, byla tato hodnota zvolena experimentálně. Samotné tření je tedy modelováno pomocí vztahu

$$\varphi = k\varphi' + \int \varphi' \mathrm{d}t,$$

kde k je hodnota tření a je nastavena na $k = 0{,}02$.

Derivace polohy vozíku byla řešena pomocí bloku pro diferenci a bloku pro zesílení podle vztahu

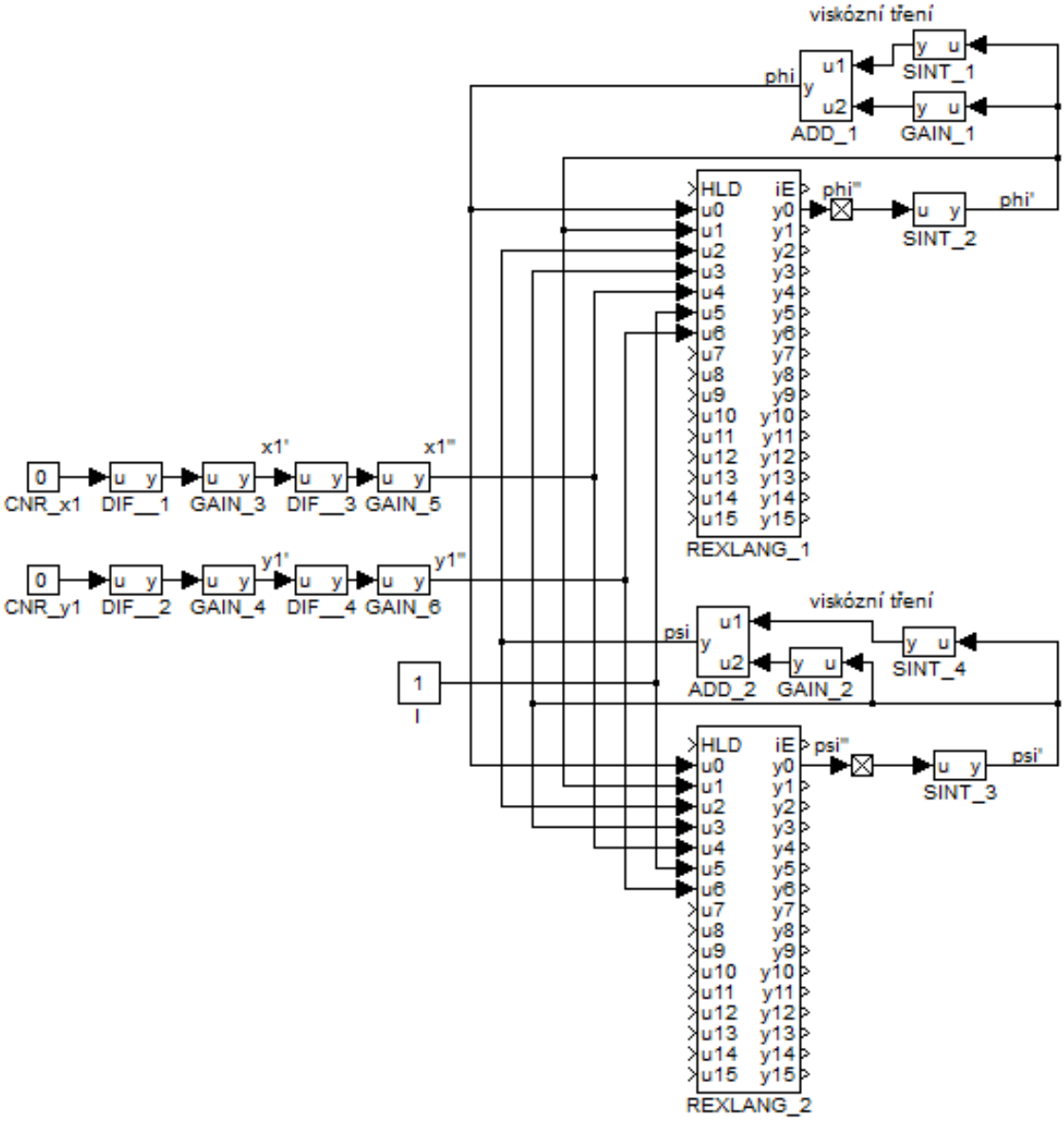
$$x' = \frac{\mathrm{diff}(x)}{T_s},$$

kde T_s je perioda spouštění jádra řídicího systému (viz kapitola 3 Simulace systému). Úloha je spouštěna s periodou $T_s = 0{,}008\,\mathrm{s}$. Po dosazení lze tedy nahlédnout, že

$$x' = \frac{\mathrm{diff}(x)}{0{,}008} = 125\,\mathrm{diff}(x).$$

Hodnota zesílení je tedy 125 a to pro každou derivaci.

Celý model kyvadla na vozíku s dvěma stupni volnosti je vidět na obrázku 2.4.



Obrázek 2.4: Model vozíku s kyvadlem a dvěma stupni volnosti pomocí bloků REXLANG

2.2 Motor pro řízení vozíku

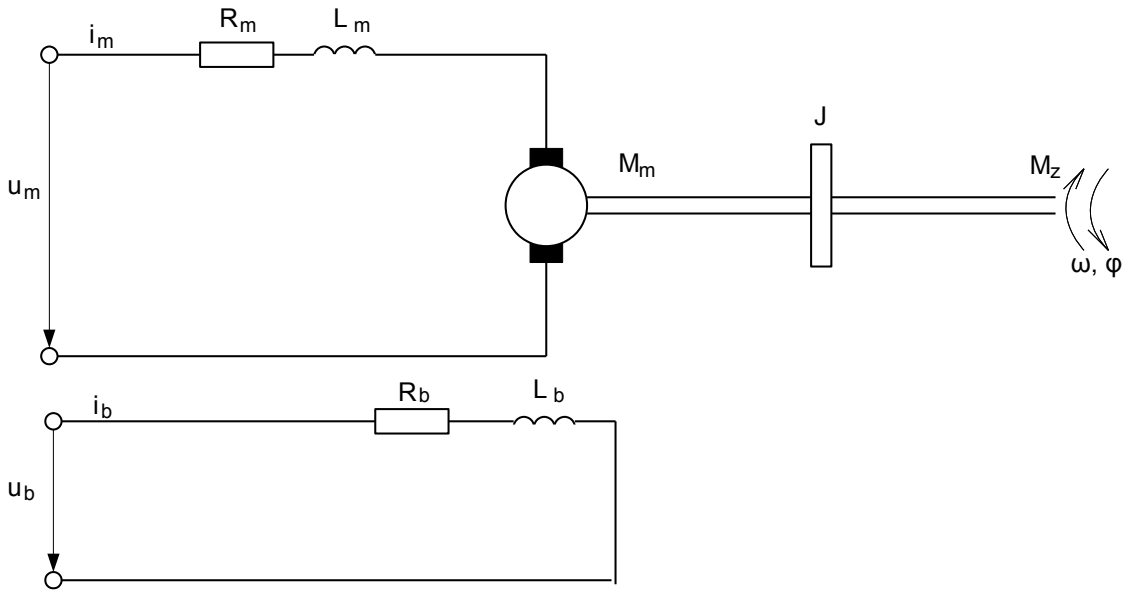
Vzhledem k tomu, že tato práce si neklade za cíl složité popisování chování motorů a motor je využit pouze k pojezdu vozíku, zvolíme tento motor jako stejnosměrný s buzením permanentními magnety a na modelu provedeme zjednodušení.

Stejnosměrný motor má na rozdíl od střídavých motorů velmi jednoduchý matematický popis a ačkoliv dnes je již možné řídit velmi přesně otáčky střídavých motorů pomocí frekvenčních měničů, je v této práci raději využita možnost řídit motor pomocí kaskády P a PI regulátorů. Více o rozdílech mezi jednotlivými druhy motorů se lze dočíst v [13].

Řízení stejnosměrných motorů se provádí změnou napětí kotvy nebo budícího proudu. Otáčky motoru se mohou pohybovat v širokém rozsahu, který není vázán na kmitočet sítě. Řízení pomocí napětí kotvy má navíc tu výhodu, že systém se dá s jistou aproximací popsat jako lineární [14].

2.2.1 Matematický model

Náhradní schéma stejnosměrného motoru s cizím buzením je znázorněno na obrázku 2.5. Pro buzení permanentními magnety je schéma stejné, ale předpokládá se, že $i_b = \text{konst}$.



Obrázek 2.5: Náhradní schéma stejnosměrného motoru

Pro definování matematického popisu jsou potřeba tyto veličiny:

u_m [V]	napětí na kotvě
i_m [A]	proud kotvy
R_m [Ω]	celkový odpor všech vinutí v kotvě
L_m [H]	celková indukčnost všech vinutí v kotvě
u_b [V]	napětí budícího obvodu
i_b [A]	proud budícího obvodu
R_b [Ω]	celkový odpor všech vinutí budícího obvodu
L_b [H]	celková indukčnost všech vinutí budícího obvodu
J [kg . m^2]	celkový moment setrvačnosti vztažený k hřídeli motoru
k [Nm . A^{-1}]	elektromechanická konstanta motoru
φ [rad]	úhel natočení hřídele
$\omega = \frac{\mathrm{d}\varphi}{\mathrm{d}t}$ [rad . s^{-1}]	úhlová rychlost hřídele
M_m [Nm]	elektromagnetický moment motoru
M_z [Nm]	zatěžovací moment motoru
B [Nm . s . rad^{-1}]	viskózní tření

Matematický model systému lze vytvořit pomocí druhého Kirchhoffova zákona a rovnice mechanické části motoru. Při vytváření modelu byl zanedbán rozptylový magnetický tok budícího vinutí, vzájemné transformační působení jednotlivých vinutí, vliv vířivých proudů v magnetickém obvodu a úbytek napětí v kartáčích [14].

2. Kirchhoffův zákon:

$$u_m = R_m i_m + L_m \frac{\mathrm{d}i_m}{\mathrm{d}t} + u_i,$$

kde u_i je indukované napětí na kotvě motoru a je dáno jako

$$u_i = k\omega.$$

Po dosazení a vyjádření derivace proudu kotvy je získán vztah

$$\frac{\mathrm{d}i_m}{\mathrm{d}t} = \frac{1}{L_m}\left[u_m - k\omega - R_m i_m\right]. \tag{2.21}$$

Mechanickou část motoru lze popsat jako

$$M_m = k i_m = J\frac{\mathrm{d}\omega}{\mathrm{d}t} + B\omega + M_z.$$

Vyjádřením ω' je získáno

$$\frac{\mathrm{d}\omega}{\mathrm{d}t} = \frac{1}{J}\left[k i_m - B\omega - M_z\right]. \tag{2.22}$$

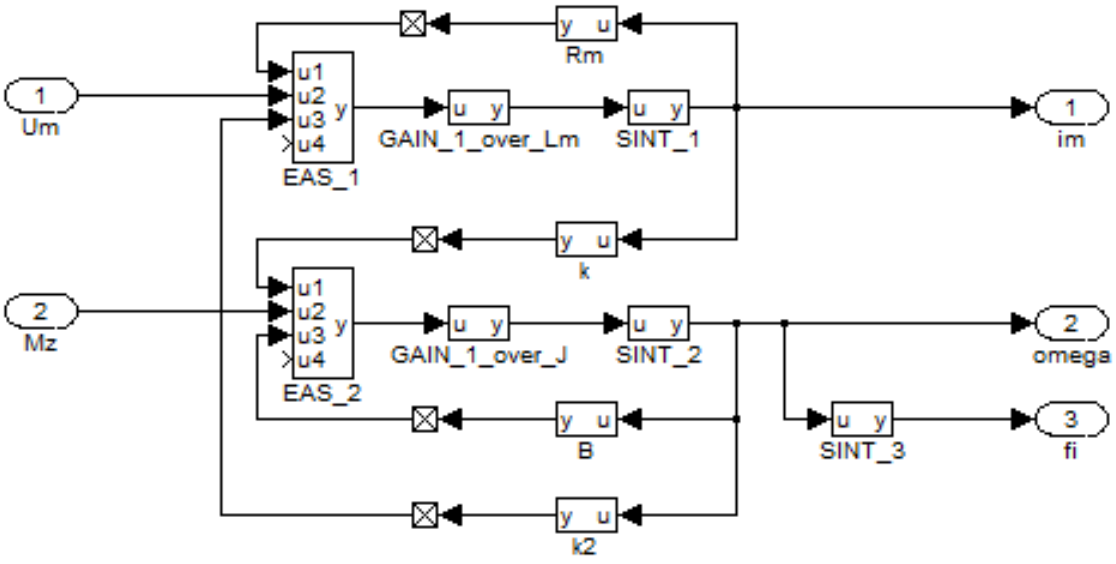
Matematickým modelem námi vybraného motoru jsou rovnice (2.21) a (2.22).

2.2.2 Simulační model

Pro simulaci motoru je nutné určit parametry L_m, k, J, B a R_m. Tyto parametry byly zvoleny podle parametrů motoru, který se nalézá v laboratoři UL 506. Tato volba má výhodu, že to jsou parametry skutečného motoru a lze tedy předpokládat, že model bude mít reálné vlastnosti a pokud se bude realizovat řízení skutečného systému, nebude potřeba tyto parametry měnit.

$$\begin{aligned} R_m &= 2\,\Omega \\ L_m &= 0{,}1\,\mathrm{H} \\ J &= 0{,}0667\,\mathrm{kg\,.\,m^2} \\ k &= 0{,}3\,\mathrm{Nm\,.\,A^{-1}} \\ B &= 0{,}01\,\mathrm{Nm\,.\,s\ .\,rad^{-1}} \end{aligned}$$

Simulační model byl opět sestaven v REXu pomocí rovnic (2.21) a (2.22). Tento model je na obrázku 2.6.



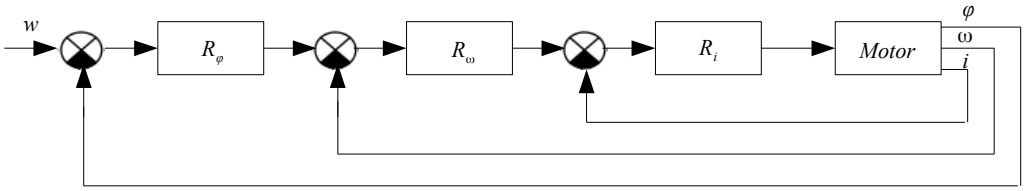
Obrázek 2.6: Model stejnosměrného motoru s buzením permanentními magnety

2.3 Řízení motoru

Vzhledem k tomu, že je potřeba, aby byl systém vozíku s kyvadlem řízen pomocí polohy vozíku (respektive jeho zrychlením), je nutné, aby výstupem motoru byla poloha. Je však potřeba si uvědomit, že tento výstup bude astatický, protože převádíme otáčivý pohyb na posuvný. S touto skutečností je nutné počítat při návrhu regulátoru.

S tím, že jedním výstupem motoru bude poloha, bylo počítáno již při návrhu motoru, a proto byl u něj tento výstup realizován. Jedná se v podstatě o zintegrovanou úhlovou rychlost. Z toho také vyplývá, že pro řízení polohy budeme muset řídit úhlovou rychlost. Pro řízení úhlové rychlosti je potom zase nutné řídit proud kotvy.

Celkem jsou tedy potřeba tři regulátory se třemi zpětnými vazbami. Regulátory jsou postupně nastavovány tzv. kaskádní regulací. To znamená, že je nejprve nastaven regulátor pro proud na kotvě, poté přidán a nastaven regulátor pro úhlovou rychlost a nakonec regulátor pro polohu. Celkové schéma zapojení regulátorů je zobrazeno na obrázku 2.7.



Obrázek 2.7: Nákres regulace motoru

Při postupu návrhu regulátorů bylo postupováno podle [13]. Tamtéž lze také nalézt další podrobnosti.

2.3.1 Regulátor proudu kotvy

Ze všeho nejdříve je nutné určit, jaký druh regulátoru použít. Určitě je nutné využít proporcionální složku, protože tato složka poskytuje rychlý přechodový děj, který je při řízení vozíku potřeba. Nevýhodou je, že při použití samotného P regulátoru vznikají po ustálení trvalé regulační odchylky. Tento problém řeší integrační složka regulátoru, proto je použita také. Derivační složka by v systému zesilovala šumy, proto nebude použita. Ve výsledku je tedy proud kotvy regulován pomocí PI regulátoru.

V tomto případě je požadována nekmitavá přechodová charakteristika. Je samozřejmě požadován co nejrychlejší přechod.

K určení parametrů regulátoru je využita rovnice (2.21), kde je zanedbán člen $k\omega$, jelikož je mnohem pomalejší než náběh proudu. Po zlaplaceování rovnice a její úpravě dostáváme vztah

$$pL_mI_m + R_mI_m = U_m$$

Z rovnice vytvoříme přenos systému

$$F(p) = \frac{I_m(p)}{U_m(p)} = \frac{1}{pL_m + R_m}.$$

Dosazením výše vybraných hodnot $R_m = 2\,\Omega$ a $L_m = 0{,}1\,\mathrm{H}$ dostáváme

$$F(p) = \frac{1}{0{,}1p + 2} = \frac{0{,}5}{0{,}05p + 1}. \tag{2.23}$$

Jelikož je již k dispozici přenos (2.23), lze využít appletu PID Control Laboratory 3.0 k určení přesných parametrů regulátoru. Applet je dostupný na adrese *www.pidlab.com* a využívá tzv. metodu robustních regionů.

PI regulátor je hledán ve tvaru

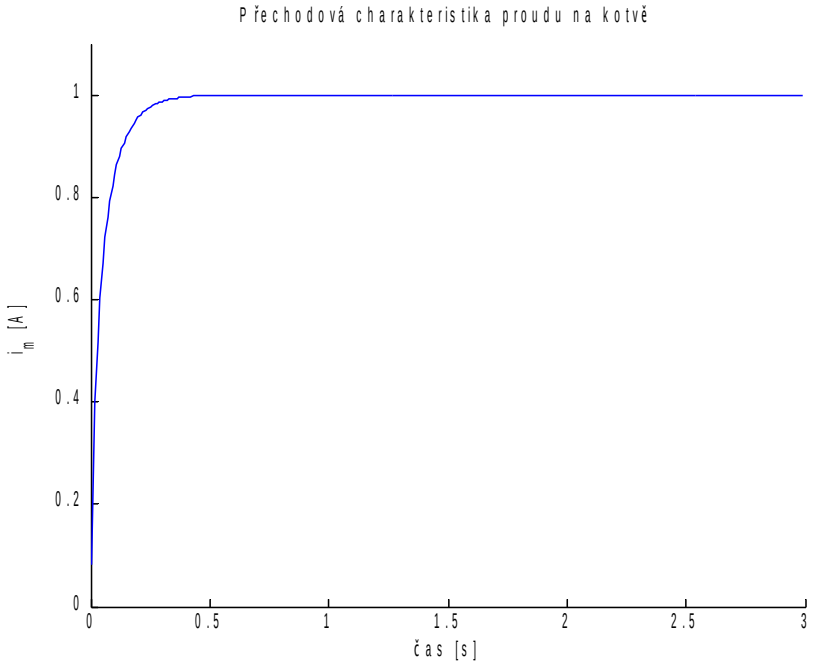
$$F(p) = K_p \left(1 + \frac{1}{T_i p}\right).$$

Regulátor má tedy parametry

$$K_p = 1{,}52$$

$$T_I = 0{,}050\,921\,273\,032.$$

Přechodová charakteristika proudu při použití navrženého regulátoru je vidět na obrázku 2.8.



Obrázek 2.8: Přechodová charakteristika proudu na kotvě

2.3.2 Regulátor úhlové rychlosti

Při určení druhu použitého regulátoru lze postupovat stejně jako u regulátoru proudu kotvy. Opět je nutné využít proporcionální složku kvůli rychlosti děje a integrační složku k odstranění trvalé regulační odchylky. Derivační složka opět není použita kvůli zesílení šumů systému. Opět je tedy využit PI regulátor. V tomto případě ale nevadí malý překmit.

Při určování parametrů regulátoru je využit opět přenos systému. Ten vyjádříme z rovnice (2.22). V této rovnici je uvažován nulový zatěžovací moment. Zlaplaceováním a úpravou rovnice dostáváme

$$pJ\Omega + B\Omega = kI_m.$$

Přenos systému má tedy tvar

$$F(p) = \frac{\Omega(p)}{I_m(p)} = \frac{k}{Jp + B}.$$

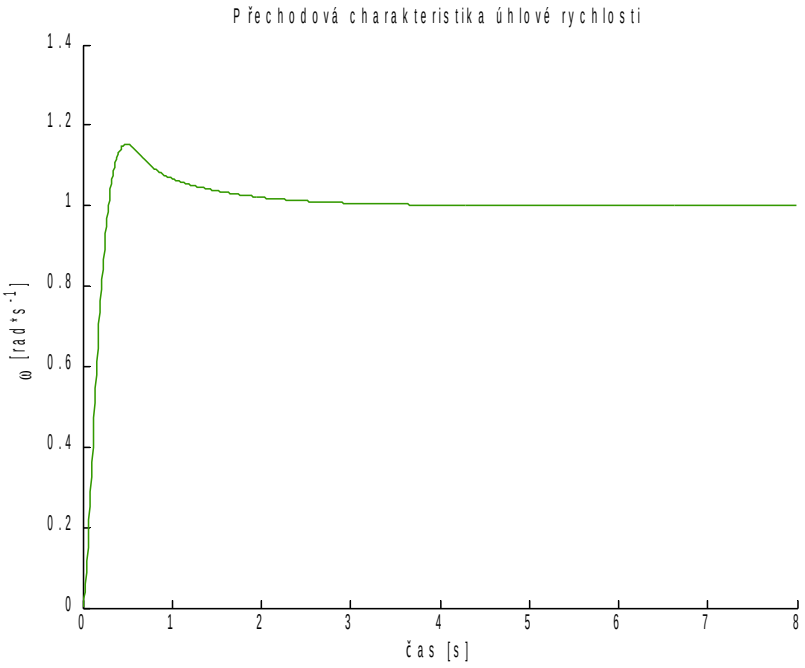
Dosazením hodnot $B = 0{,}01\,\mathrm{Nm}\,.\,\mathrm{s}\,.\,\mathrm{rad}^{-1}$, $J = 0{,}0667\,\mathrm{kg}\,.\,\mathrm{m}^2$ a $k = 0{,}3\,\mathrm{Nm}\,.\,\mathrm{A}^{-1}$ dostáváme

$$F(p) = \frac{0{,}3}{0{,}0667p + 0{,}01} = \frac{30}{6{,}67p + 1}. \tag{2.24}$$

Regulátor má parametry

$$K_p = 1{,}35, T_I = 1.$$

Přechodová charakteristika úhlové rychlosti při použití navrženého regulátoru je vidět na obrázku 2.9.



Obrázek 2.9: Přechodová charakteristika úhlové rychlosti

2.3.3 Regulace polohy

Při regulaci polohy je opět využita proporcionální složka. Není však v tomto případě využita integrační složka. Je potřeba si totiž uvědomit, že řízení polohy motoru je astatický problém a tudíž již sám v sobě integrační složku obsahuje a je tedy zajištěno, že uzavřený systém s regulátorem nebude mít po ustálení žádnou regulační odchylku. V tomto případě je tedy dostatečné použít P regulátor.

Vzhledem k tomu, že je hledán pouze jeden parametr, je nejrychlejší metodou k získání tohoto parametru metoda pokus/omyl. Postupně tedy byla zvyšována hodnota K_p, dokud se neobjevily překmity a poté byla hodnota pomalu snižována dokud překmit nezmizel.

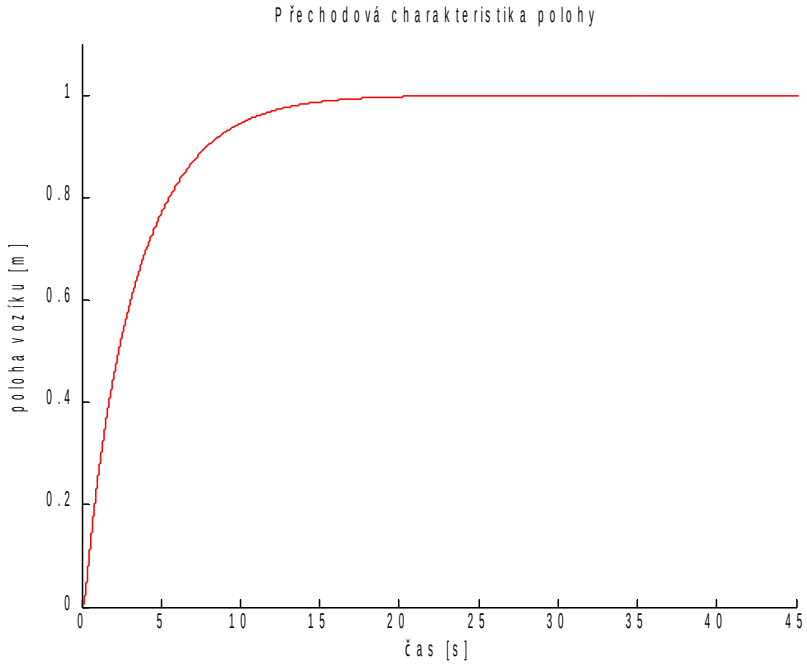
Přechodová charakteristika je vyžadována bez překmitu z toho důvodu, že cílem práce je minimalizovat kmitání kyvadla při dojezdu vozíku. Překmit při řízení vozíku by znamenal další nežádoucí vybuzení kmitání kyvadla.

Z hlediska nejrychlejšího přejezdu bez překmitu je vhodným parametrem

$$K_p = 1{,}92.$$

Při této hodnotě však dochází u kyvadla k přetáčení, k čemuž nebyl systém navrhnut, proto bylo nutné tuto hodnotu snížit na $K_p = 0{,}288$.

Přechodová charakteristika polohy vozíku při použití navrženého regulátoru je vidět na obrázku 2.10.

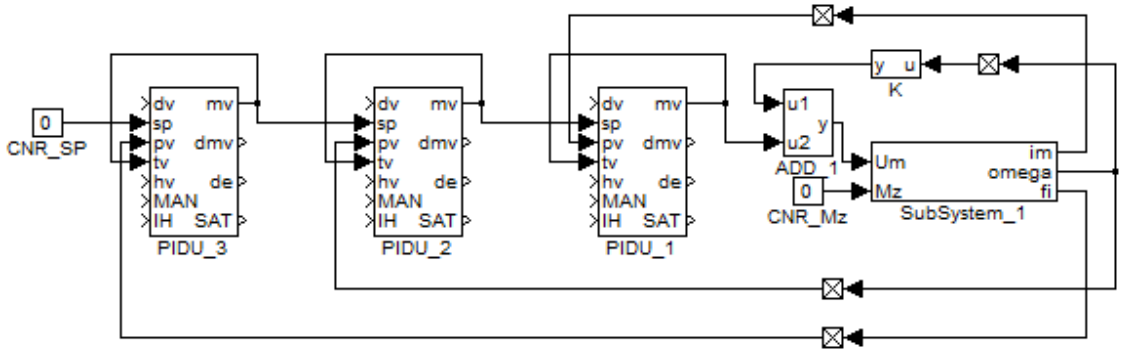


Obrázek 2.10: Přechodová charakteristika polohy vozíku

2.3.4 Kaskádní regulace

Nyní, když jsou nastaveny všechny regulátory, lze jejich zřetězením přistoupit ke kaskádní regulaci. Je vhodné postupovat podle nákresu na obrázku 2.7. V nejvnitřnější smyčce je tedy regulace proudu, v další je regulace úhlové rychlosti a ve vnější smyčce pak regulace polohy natočení hřídele a tím i zároveň polohy vozíku. Uživatel si pak bude nastavovat požadovanou polohu vozíku v dané ose a kaskáda regulátorů bude řídit motor tak, aby tam v co nejkratším čase bez překmitů dojel.

Při zapnutém ZV filtru bude toto nastavení samozřejmě ovlivněno filtrem tak, aby se minimalizovalo vybuzení na nežádoucí frekvenci (více viz sekce 2.4 Input shaping filtr). Zapojení motoru s viskózním třením a kaskádou regulátorů v REXu je uvedeno na obrázku 2.11.

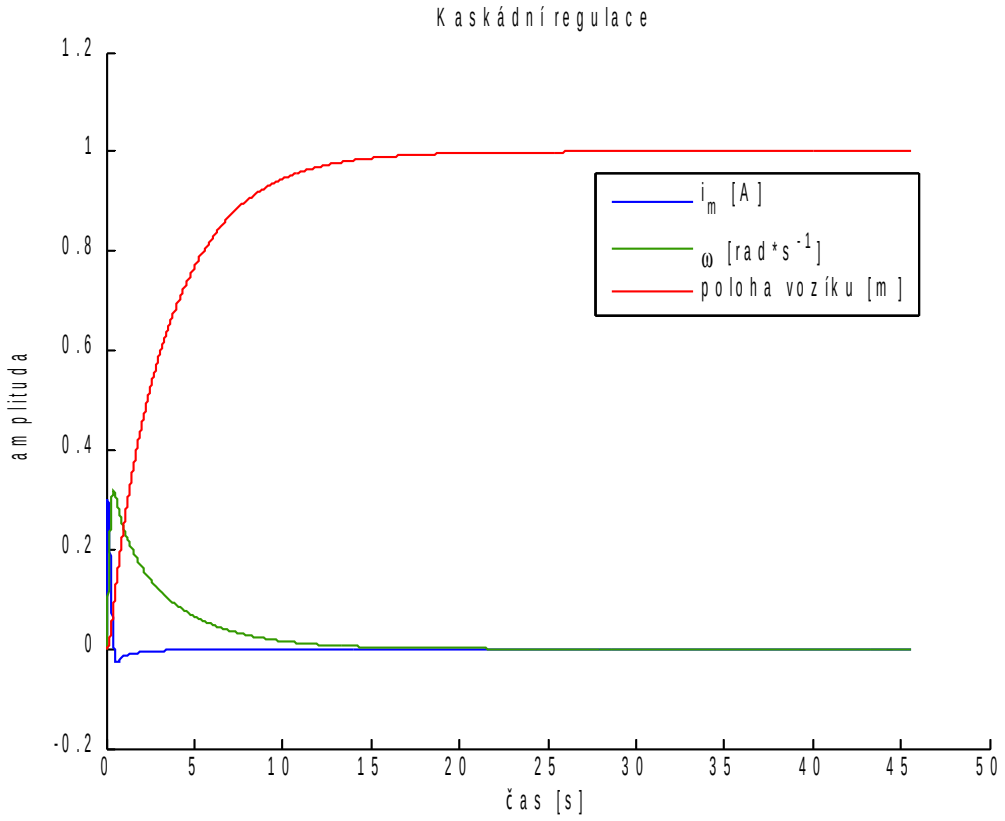


Obrázek 2.11: Schema zapojení motoru s regulátory v REXu

V bloku SubSystem_1 je obsažen model motoru, který je uveden v subsekci 2.2.1. Viskózní tření je zavedeno pomocí kladné zpětné vazby podle stejného principu jako v sekci 2.1 Vozík s kyvadlem a dvěma stupni volnosti, subsekci 2.1.4 Simulační model. Hodnota tření je v tomto případě nastavena na $k = 0{,}03$, protože je předpokládáno, že v motoru působí větší třecí síly než u zavěšení kyvadla na vozíku.

Průběh kaskádní regulace při přejezdu vozíku z počátku do vzdálenosti 1 m je znázorněn na obrázku 2.12.

Tento průběh je zaznamenán bez aktivního IS filtru a tedy požadovaná poloha není nijak ovlivněna. Z trendů je vidět, že vozík přejede z počáteční do cílové polohy za zhruba dvacet sekund. Doba přejezdu se pro různé vzdálenosti nemění, protože nedochází k saturaci regulátorů v modelu motoru díky vysokým mezím. Při používání modelu se nepředpokládají přejezdy vozíku v řádech desítek metrů, a proto by nemělo docházet k saturaci regulátorů ani u reálného motoru a tím pádem je model akceptovatelnou reprezentací reálného motoru.



Obrázek 2.12: Průběh kaskádní regulace při přejezdu vozíku z počátku do vzdálenosti 1 m

2.4 Input shaping filtr

Informace pro tuto sekci byly čerpány z [11] a [15]. Tamtéž lze také nalézt další podrobnosti.

Základní myšlenkou tzv. input shaping filtrů (dále jen IS filtrů) je rozložit skok požadované hodnoty v čase tak, aby nedošlo k vybuzení kmitání systému na frekvenci, kterou chceme potlačit. Filtr tedy místo toho, aby skokově změnil požadovanou veličinu, rozloží tuto změnu na několik menších skoků, mezi nimiž jsou časové prodlevy. Velikosti dílčích skoků a časová zpoždění jsou vypočteny tak, aby se pokud možno vyrušilo vybuzení systému na dané frekvenci. Díky této vlastnosti se IS filtrům také někdy říká zero vibration filtry.

Nevýhodou tohoto způsobu potlačení vibrací je právě časové zpoždění, které filtr zavádí do systému. Toto zpoždění závisí na frekvenci, kterou je potřeba vyrušit (platí zde přímá úměra). Při správném nastavení filtru bude však systém i s časovým zpožděním kmitat s daleko nižší amplitudou, než systém bez IS filtru ve stejném časovém okamžiku.

Výhodou takto řešeného tlumení vibrací je konečná impulsní odezva, zaručená stabilita a monotónní přechodová charakteristika filtru.

Metody návrhu parametrů IS filtru popsané v článku [15] se dají prakticky použít v bloku ZV4IS řídicího systému REX (viz [11]). U tohoto bloku jsou v zásadě dvě možnosti, jak lze navrhnout parametry filtru. Buď lze využít jednoho z již přednastavených typů filtrů a nebo si pomocí tří pomocných parametrů nastavit filtr vlastní.

Z již nastavených filtrů lze vybírat například mezi ZV, ZVD, ZVDD, MISZV filtry. Tyto filtry se liší tvarem frekvenční charakteristiky a šířkou nepropustného pásma. Vzhledem k tomu, že lze přesně určit frekvenci a tlumení kmitavého módu, které má být potlačeno, je vhodné zvolit ZV filtr.

Druhou možností, jak nastavit IS filtr, je nastavit přímo parametry filtru. Tyto parametry jsou celkem tři a mají jasný fyzikální význam. Parametr asymetrie *p_ alpha* udává polohu sedla nepropustné oblasti frekvenční charakteristiky filtru vůči potlačované frekvenci. Parametr *p_ a2* udává šířku a úroveň útlumu nepropustného pásma. Poslední parametr *p_ a3* se nastavuje pouze pro symetrické filtry ($p\_\ alpha = 0$) a je vhodné mu nastavit stejnou hodnotu jako parametru *p_ a2*.

U obou možností nastavení filtru je však nutné zadat úhlovou frekvenci, kterou je třeba v systému potlačit (zadává se v $\mathrm{rad}\,.\,\mathrm{s}^{-1}$), a tlumení kmitavého módu. Potlačovanou úhlovou frekvencí je myšlena frekvence kmitání kyvadla. Tato frekvence je dána vztahem

$$f = \frac{1}{T} = \frac{1}{2\Pi\sqrt{\frac{l}{g}}} = \frac{1}{2\Pi}\sqrt{\frac{g}{l}}.$$

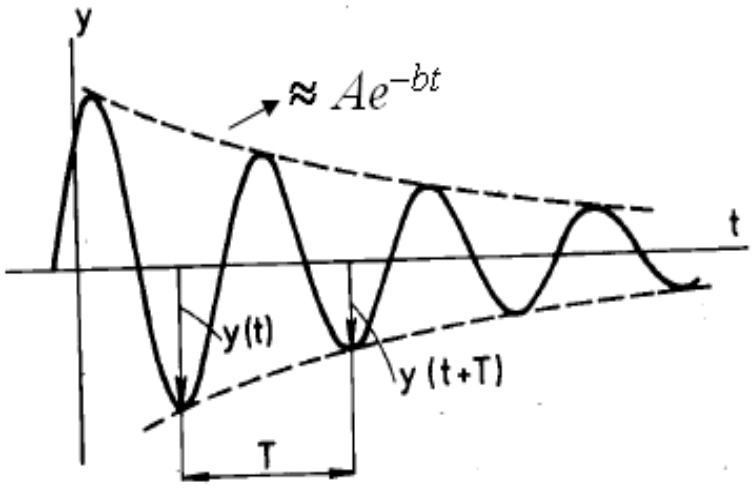
Úhlovou frekvenci lze určit jako

$$\omega = 2\Pi f = \sqrt{\frac{g}{l}}.$$

Po dosazení střední hodnoty normálního tíhového zrychlení $g = 9{,}806\,65\,\mathrm{m}\,.\,\mathrm{s}^{-2}$ a délky kyvadla $l = 1\,\mathrm{m}$ lze určit frekvenci jako

$$\omega = \sqrt{\frac{9{,}806\,65}{1}} \doteq 3{,}131\,557\,\mathrm{rad}\,.\,\mathrm{s}^{-1}.$$

Tlumení kmitavého módu lze popsat jako obalovou exponencielu kmitání, které je třeba potlačit (předpokládáme, že kmitání je tlumené, což je u reálných systémů přirozené). Situace je znázorněna na obrázku 2.13.



Obrázek 2.13: Tlumení kmitavého módu

A je amplituda kmitu v počátku a b je právě kmitavý mód systému, který je dán hodnotou tření. Při modelování kyvadla byla hodnota tření určena jako

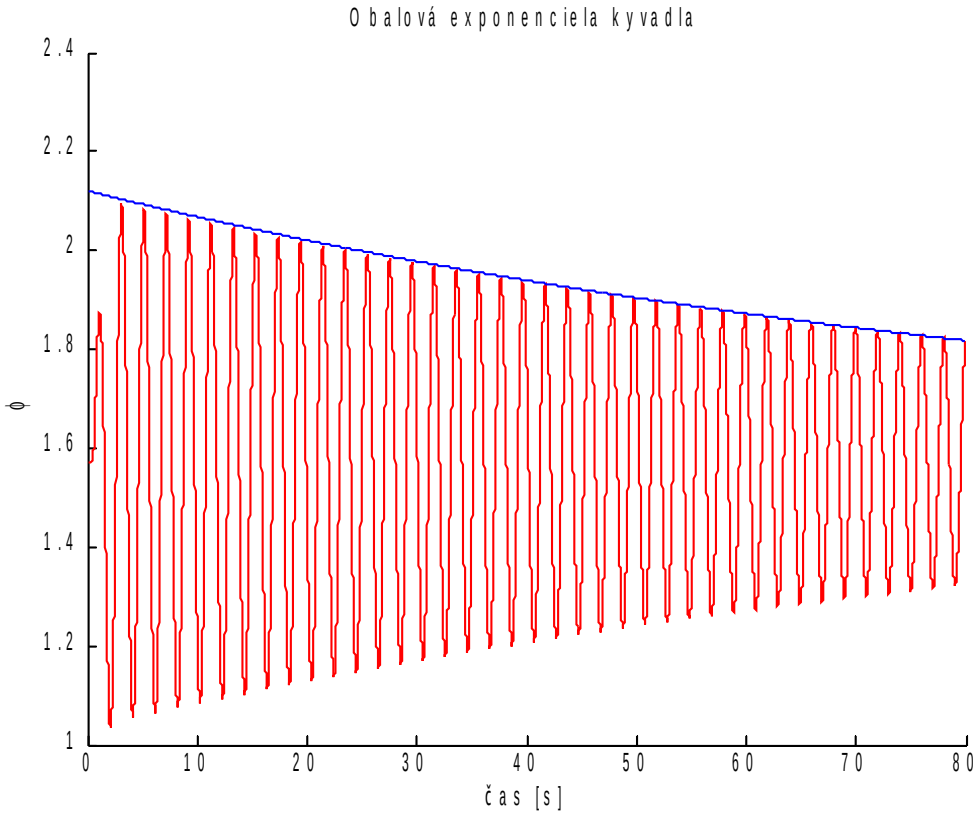
$$b = 0{,}02.$$

To, že tato hodnota je rovna hodnotě kmitavého módu lze experimentálně ověřit. Na obrázku 2.14 je znázorněn průběh netlumeného kmitání kyvadla a vrchní obalová exponenciála, která je dána vzorcem

$$y = A\mathrm{e}^{-bt} = 0{,}55\mathrm{e}^{-0{,}02t}.$$

Amplituda se samozřejmě mění v závislosti na počátečním vybuzení systému, tlumení kmitavého módu však zůstává stejné.

Vzhledem k tomu, že je známa přesná úhlová frekvence a tlumení kmitavého módu, je vhodne realizovat IS filtr jako typ ZV. Tento typ má úzkou šířku nepropustného pásma, ale zato zavádí do systému nejmenší zpoždění.



Obrázek 2.14: Obalová exponenciála netlumených kmitů kyvadla

Kapitola 3

Simulace systému

3.1 Perioda spouštění simulace

Ještě před začátkem simulace je nutné nastavit dostatečně malou periodu spouštění jádra řídicího systému REX T_s. Jde o to, že se snažíme pomocí diskrétní simulace simulovat spojitý proces a z toho vyplývá, že čím menší je perioda jádra, tím více se diskrétní simulace blíží spojité a tím je také v porovnání se skutečností přesnější. Tuto periodu lze orientačně zvolit podle vztahu

$$\frac{T_s}{T_c} \approx \frac{1}{100} \cdots \frac{1}{20},$$

kde T_c je kritická perioda a v našem případě je to perioda kmitání kyvadla.

Délka kyvadla půjde v laboratoři interaktivně měnit a jelikož kritická perioda závisí na délce kyvadla, je nutné určit kritickou periodu pro nejkratší možnou délku kyvadla, což bude $l = 0{,}1\,\mathrm{m}$. Jednoduchým výpočtem lze zjistit, že

$$T_c = 2\pi\sqrt{\frac{l}{g}} = 2\pi\sqrt{\frac{0{,}1}{9{,}806\,65}} \approx 0{,}634\,5\,\mathrm{s}.$$

Perioda T_s byla tedy zvolena jako $0{,}008\,\mathrm{s}$.

Perioda je nastavena jako velmi krátká a tím pádem bude kladen velký důraz na výpočetní nároky stroje, na kterém poběží jádro řídicího systému. Při vyšších hodnotách se však systém nechová podle předpokladů a fyzikálních zákonů. Je také předpokládáno, že systém bude fungovat na dostatečně výkoném počítači a tudíž by s výpočetními nároky neměl být problém.

3.2 Simulační model v REXu

Po namodelování dílčích částí systému a odzkoušení jejich funkčnosti (viz kapitola 2 Modelování systému) je možné začít části spojovat do jediného funkčního bloku, který bude převeden na jádro virtuální laboratoře.

Pohyb vozíku s kyvadlem je řízen dvěma stejnými motory (pro každou osu pohybu vozíku jeden motor), u kterých je řízena poloha natočení hřídele a tím i poloha vozíku. Motory mohou mít předřazeny IS filtry pro tlumení (respektive co nejmenší vybuzení) kmitů kyvadla.

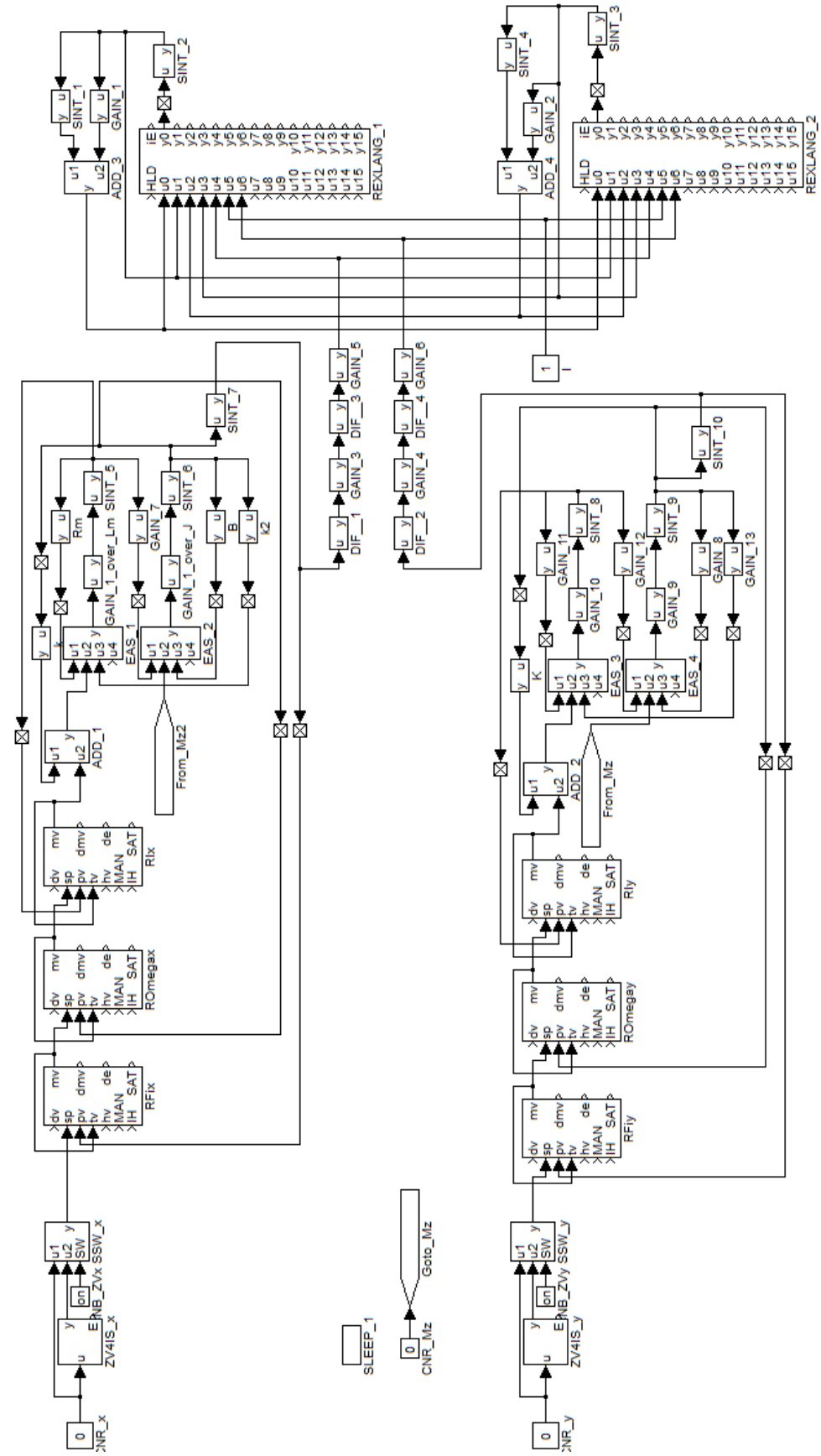
Celý systém by měl tedy fungovat ve výsledku takto:

- uživatel nastaví délku kyvadla, zapne/vypne IS filtry pro každou osu a nastaví požadovanou pozici vozíku
- pokud bude zapnutý IS filtr, pak na základě známé úhlové frekvence a tlumení kmitavého módu kyvadla rozloží filtr skok v souřadnicích na několik menších tak, aby bylo minimalizováno kmitání kyvadla
- pokud filtr nebude aktivní, pak se požadovaná pozice vozíku přenese přímo na set point kaskády regulátorů řídících polohu vozíku
- motory na základě výstupu z IS filtrů, či v případě vypnutého filtru na základě požadované pozice řídí polohu vozíku
- u kyvadla dochází v závislosti na zrychlení vozíku k vybuzení kmitů

Celý simulační systém byl následně převeden do třídy v jazyce Java. Tato třída definuje pomocí již dříve vytvořené knihovny JavaRexLib chování systému v Javě. JavaRexLib obsahuje většinu bloků, které se dají v REXu použít. Nicméně neobsahuje podporu pro subsystémy a pro blok RexLang. Z tohoto důvodu bylo nutné odstranit subsystémy pro modely motorů a bloky pro simulaci těchto modelů byly umístěny přímo do kompletního schématu. Odstranění problému s bloky RexLang je popsáno v sekci 5.5 Převod modelu systému do jazyka Java.

Ve finále by měl mít uživatel k dispozici jednoduchý applet, ve kterém by měl být schopen nastavit souřadnice, kam chce, aby vozík dojel. Stav celé simulace může uživatel sledovat pomocí trendů umístěných přímo v appletu. Pro sledování aktuálního stavu systému by měla také sloužit 3D vizualizace vozíku s kyvadlem. Simulační systém jsme se snažili vytvářet s ohledem na tyto skutečnosti a neměl by být tudíž problém sledovat jakoukoliv veličinu a nastavovat požadované souřadnice vozíku, délku kyvadla a zapínat/vypínat IS filtry.

Celé zapojení simulace systému je vidět na obrázku 3.1.

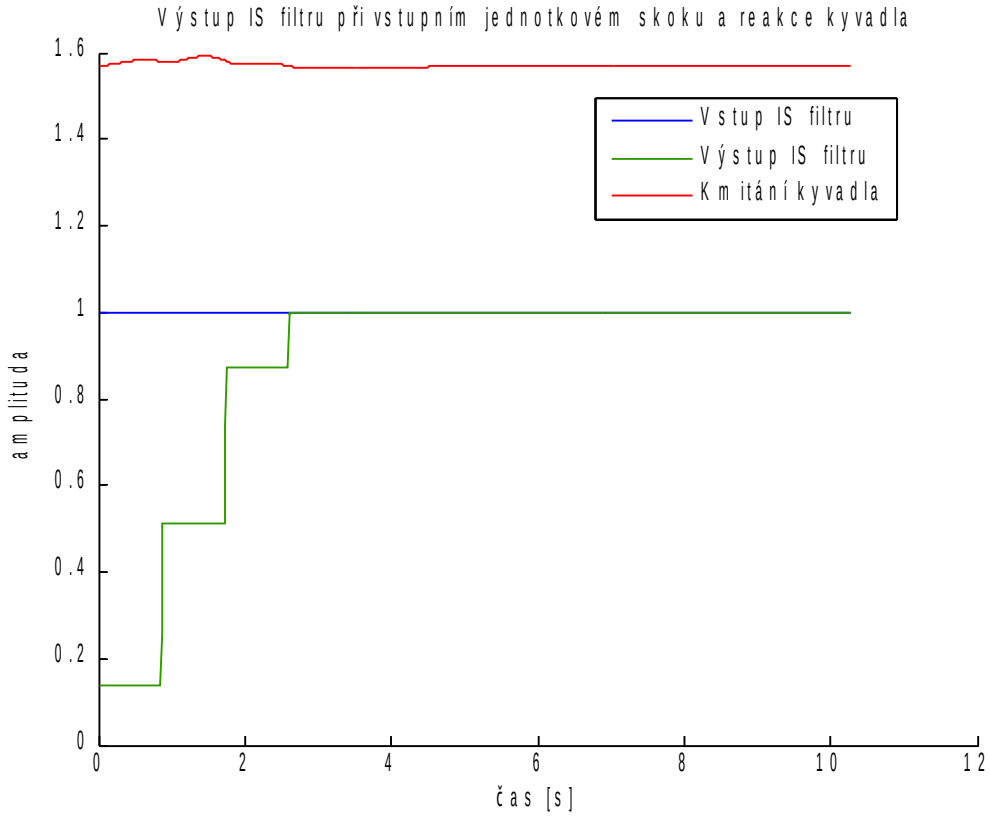


Obrázek 3.1: Kompletní simulační schema

3.3 Testování kompletního systému

Na obrázku 3.2 je zobrazeno kmitání kyvadla s délkou $l = 1\,\mathrm{m}$ v jedné ose, pokud vozík přejede z počáteční polohy do vzdálenosti $x = 1\,\mathrm{m}$ a to pouze ve směru osy X. Pro zajímavost je zde také ukázáno, jak IS filtr rozloží tuto požadovanou skokovou změnu na několik postupných skoků tak, aby systém nebyl vybuzen na zadané úhlové frekvenci. Celková doba zpoždění, kterou filtr zavede do systému při tomto přejezdu je zhruba 2,6 s, což je podstatně méně než doba ustálení kyvadla při nulovém tlumení. Doba ustálení záleží samozřejmě na tření systému. Práce vychází z předpokladu, že pro systémy s velkým třením by nemělo smysl tlumit jejich kmitání.

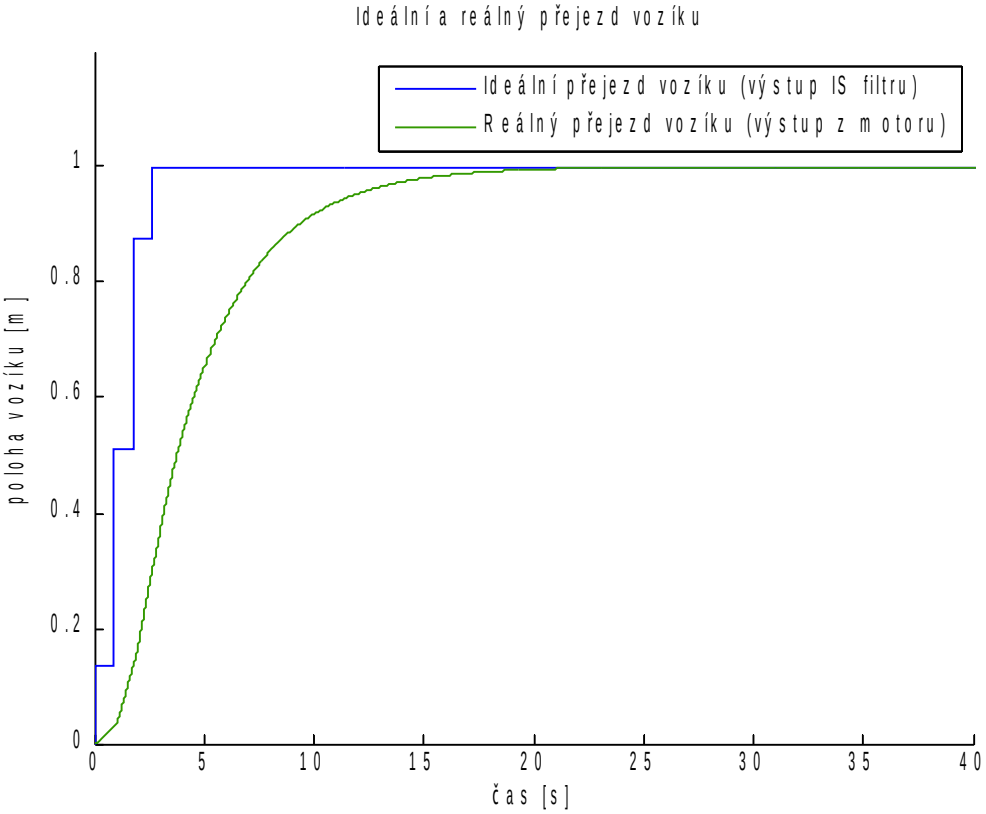
Na průběhu stavu kyvadla je vidět, že při každé změně výstupu IS filtru následuje ráz. Tyto rázy jsou však takové, že jejich působení se vzájemně vyruší téměř v celém rozsahu.



Obrázek 3.2: Tlumené kmitání kyvadla v závislosti na změně požadované polohy vozíku

Kmity kyvadla se samozřejmě nevyruší zcela. Důvod je nasnadě a zní takto: Pokud budeme předpokládat dokonalou činnost IS filtru v tom smyslu, že bude generovat přesně takovou posloupnost přechodů, aby systém nebyl vybuzen na frekvenci, se kterou kmitá kyvadlo, bylo by pro dokonalé vyrušení kmitání ještě potřeba, aby přechody mezi jednotlivými mezistupni byly ideální (nekonečně rychlé). Jelikož ale se o přejezd vozíku stará motor s reálnými parametry, mají přejezdy vozíku nenulovou dobu trvání a přechod tedy není stejný, jak ho generuje IS filtr. Tato situace

je přehledně vidět na obrázku 3.3.



Obrázek 3.3: Srovnání ideálního a reálného přejezdu vozíku

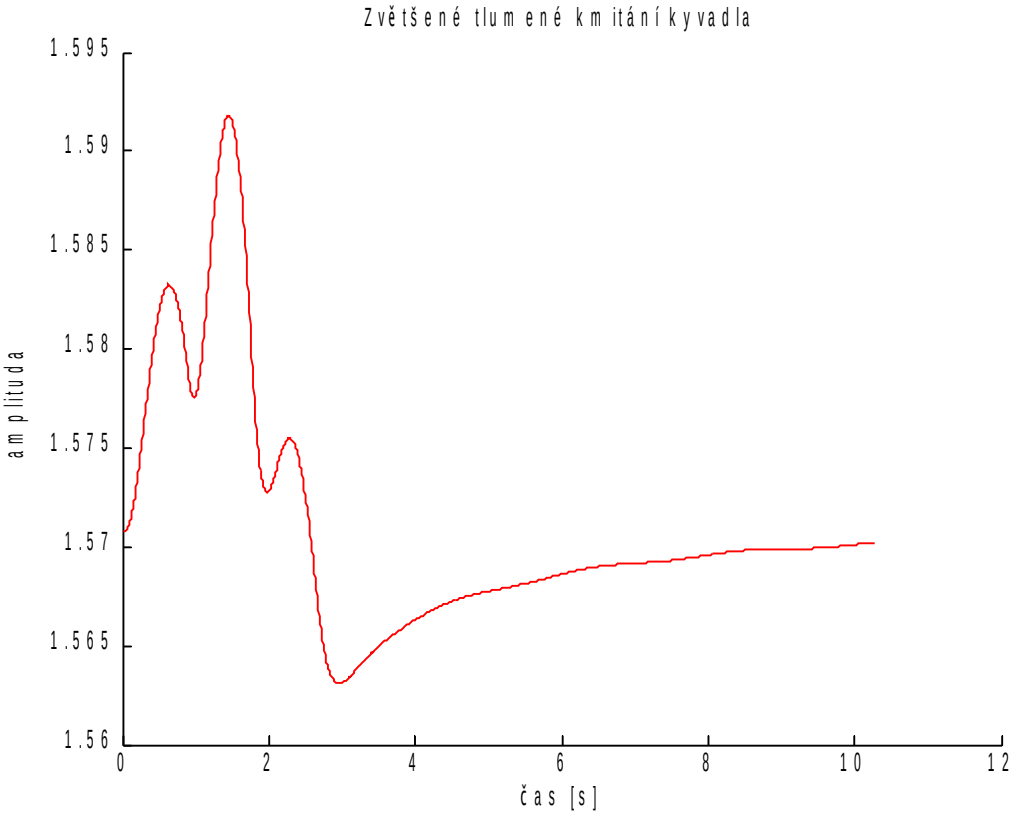
Je dobré poznamenat, že rychlost skoků generovaných IS filtrem závisí na periodě kmitů kyvadla, která závisí na délce kyvadla. Čím delší je kyvadlo, tím delší je tedy i prodleva mezi jednotlivými skoky a tím delší čas má i motor pro přejezd vozíku na požadované souřadnice. Z toho vyplývá, že pro delší kyvadlo jsou reálné přejezdy více podobné ideálním a tím pádem bude i nižší výchylka kyvadla po dojetí vozíku do cílové polohy.

Amplituda zbytkových kmitů je v této situaci tak malá, že kyvadlo lze již považovat za stabilizované. Tato amplituda se pohybuje v řádu tisícin radiánu (pro tento konkrétní případ je počáteční velikost kmitu rovna jedné tisícině). Největší hodnota rázů kyvadla způsobených přejetím vozíku je přibližně rovna hodnotě 0,075 rad. Zvětšený průběh kmitání kyvadla je vidět na obrázku 3.4.

Na obrázku 3.5 je znázorněn průběh netlumeného kmitání kyvadla v závislosti na stejný požadovaný přechod vozíku jako je tomu na obrázku 3.2. Při porovnání těchto dvou grafů je vidět, že v systému sice není žádné zpoždění způsobené input shapingem, nicméně amplituda kmitů je více než pětisetnásobně vyšší než u tlumeného kmitání a osmkrát vyšší, než je nejvyšší hodnota výchylky způsobená rázem.

Jak je uvedeno výše, pro délku kyvadla $l = 1\,\mathrm{m}$ je zpoždění zavedené IS filtrem rovné zhruba hodnotě 2,6 s. Pokud tedy posuneme čas u netlumených kmitů o tuto

hodnotu, zjistíme, že amplituda kmitů je neustále několiksetnásobně vyšší, než v případě použitíi IS filtru.

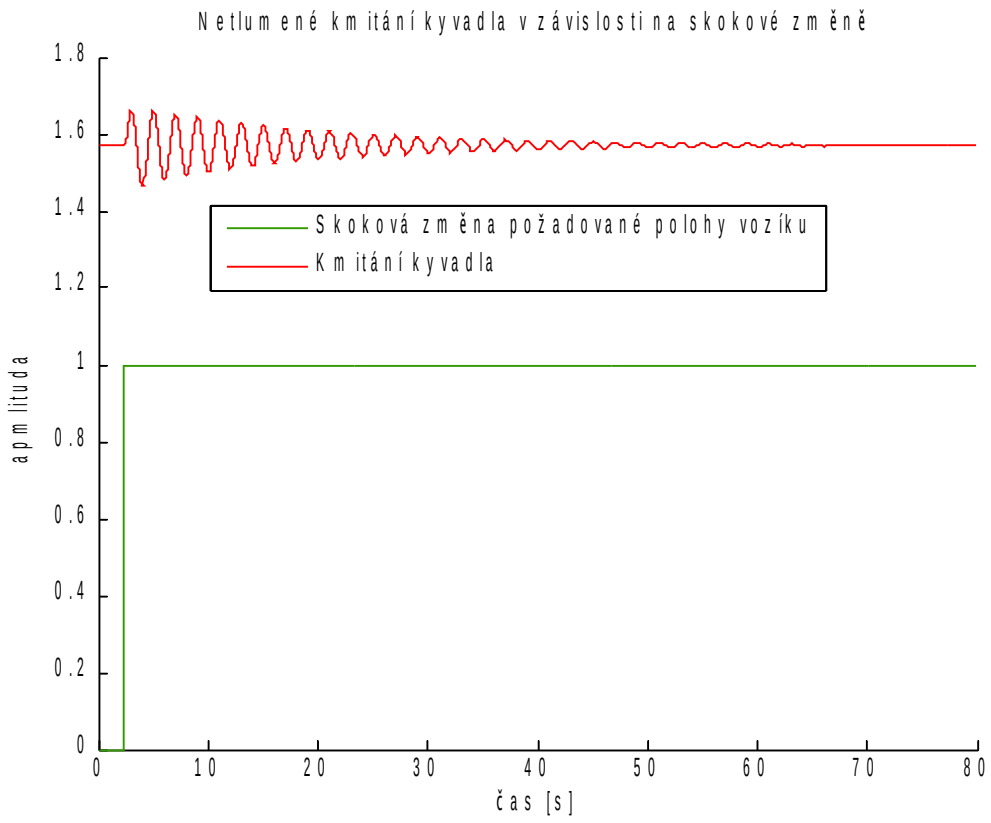


Obrázek 3.4: Zvětšené tlumené kmity kyvadla

Systém logicky nemůže fungovat pro jakoukoli délku kyvadla. Zde se jedná spíše o dolní mez intervalu délky kyvadla. Pro nějakou nízkou hodnotu již bude frekvence kmitů tak vysoká, že již nebude dostačovat zvolená perioda spouštění jádra a systém se začne chovat nesmyslně. Je tedy nutné zvolit pracovní interval délky kyvadla. Je předpokládáno, že uživatel bude moci volit délku kyvadla pouze v tomto intervalu, a je tedy nutné pro tento interval ověřit správné fungování systému.

Horní hranice intervalu je závislá na té spodní. Na grafickém modelu, který bude navržen pro zobrazování vozíku s extrémně krátkým kyvadlem, bude nepřehledné, když kyvadlo bude dlouhé a naopak. Jako praktické rozmezí se jeví rozmezí dvou dekád. Jestliže tedy byla zvolena spodní mez intervalu jako 0,1 m, bude horní mez 10 m. Trojrozměrný model já konstruován pro délku kyvadla 1 m a měla by na něm být tedy přehledně vidět situace pro obě mezní délky.

S délkou kyvadla také souvisí rozmezí, ve kterém se může vozík pohybovat ve směrech X a Y. Je předpokládáno, že jeřáb s provazem dlouhým 0,1 m nebude potřebovat pojezd dlouhý v řádech stovek metrů a naopak jen těžko si lze představit jeřáb s provazem dlouhým 10 m, který bude manipulovat na prostoru v řádech desetin metrů čtverečních. Jako rozumné meze pohybu se tedy jeví intervaly $< 0; 10\,\mathrm{m} >$



Obrázek 3.5: Netlumené kmitání kyvadla

pro každou osu.

Nyní když jsou určené meze pohybu vozíku jeřábu a meze možných délek provazu (kyvadla), lze provést otestování systému při mezních hodnotách. Práce vychází z předpokladu, že pokud se systém bude chovat fyzikálně správně pro tyto hodnoty, bude se chovat správně pro jakékoliv hodnoty z těchto intervalů. Výstupy systému pro zapnutý i vypnutý IS filtr a různé hodnoty parametrů jsou uvedeny v grafech na obrázcích 3.6 - 3.11.

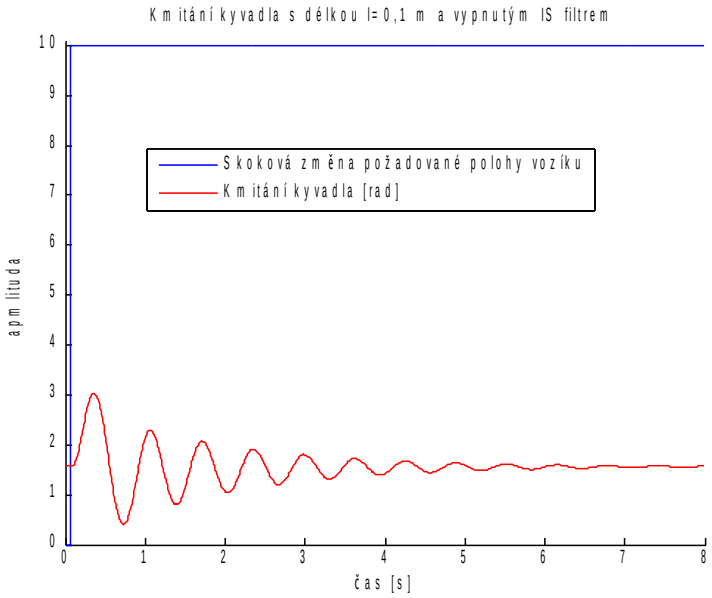
Na obrázcích 3.6 a 3.7 jsou grafy pro přejezd vozíku o deset metrů v jedné ose při délce kyvadla $l = 0{,}1\,\mathrm{m}$ při vypnutém i zapnutém IS filtru. Díky vysoké frekvenci kmitání se systém ustálí velmi rychle. V tomto případě je tedy zkoumána hlavně funkce filtru potlačení amplitudy kmitání a ne zkrácení času ustálení. Nicméně čas ustálení při zapnutém filtru je také kratší. Maximální výchylka kyvadla při zapnutém filtru je přibližně 0,543 rad, tedy 31,123°. Při vypnutém filtru je maximální výchylka přibližně 1,463 rad, tedy 83,835°. Zde se však jedná o extrémní případ, kdy malé kyvadlo přejíždí velkou vzdálenost a není předpokládáno, že by tlumení takovýchto kmitů bylo časté.

Na obrázcích 3.8 a 3.9 jsou vidět stejné grafy, ale délka kyvadla je v tomto případě $l = 1\,\mathrm{m}$. Při této délce kyvadla již není tak vysoká frekvence kmitání kyvadla, a proto kmity odezní později. Na grafech je vidět, že při zapnutém filtru se kyvadlo zcela neustálí ani za osmdesát sekund, kdežto se zapnutým filtrem je kyvadlo ustá-

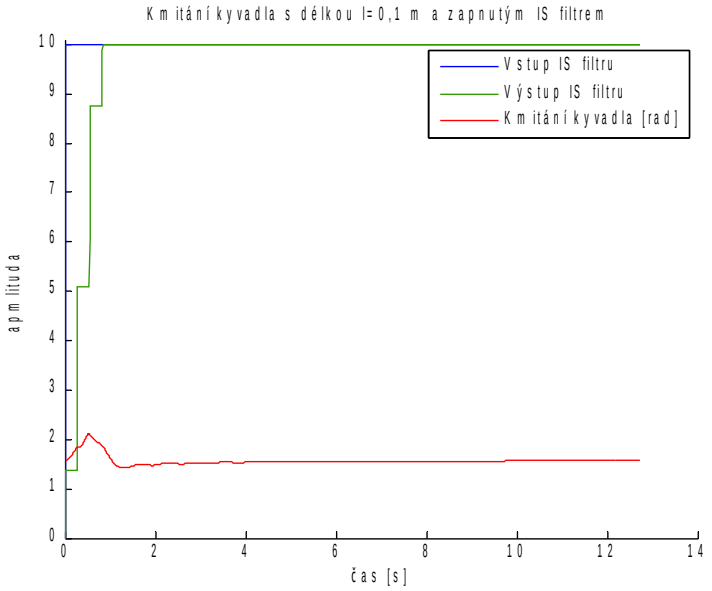
lené zhruba po šesti vteřinách a po dojezdu vozíku na pozici jsou již kmity okem nepozorovatelné. Největší výchylka při vypnutém IS filtru je 0,381 rad (21,841°). Zapnutím filtru lze tuto hodnotu snížit na 0,209 rad (11,986°). To je navíc výchylka způsobená rázem při přejezdu vozíku a dochází k ní tedy před dojezdem vozíku na cílovou pozici.

Obrázky 3.10 a 3.11 obsahují grafy stejného přejezdu, avšak s délkou kyvadla $l = 10\,\mathrm{m}$. Na tomto případě je nejvíce markantní zkrácení doby ustálení kyvadla, protože při vypnutém filtru se po osmdesáti vteřinách sníží amplituda kmitů jen nepatrně, kdežto při zapnutém filtru lze považovat kyvadlo ustálené již po patnácti sekundách. Maximální výchylka se zapnutým filtrem je 0,051 rad (2,934°). Vypnutím filtru se tato hodnota zvýší na 0,320 rad (18,323°).

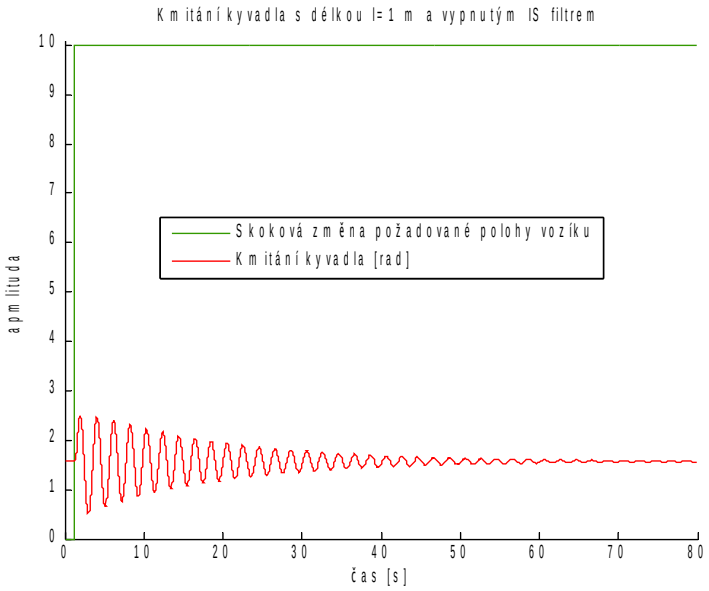
Pro různé délky kyvadla byly spočteny teoretické frekvence kmitání a následně byly porovnány s frekvencí určenou z grafů. Hodnoty se shodují.



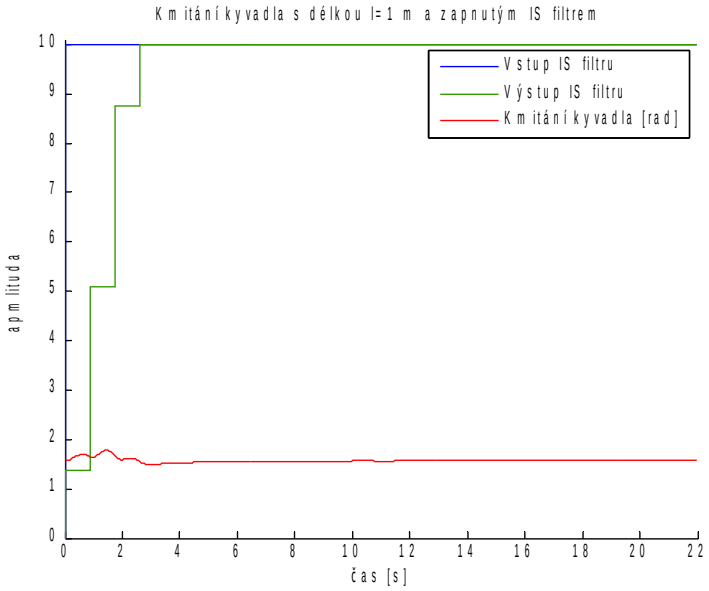
Obrázek 3.6: Odezva kyvadla l = 0,1 m na přejezd vozíku o 10 m bez IS filtru



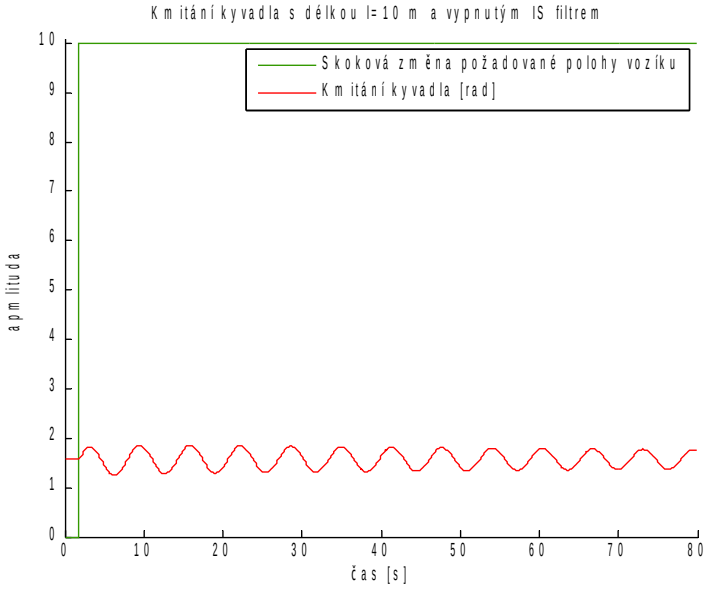
Obrázek 3.7: Odezva kyvadla l = 0,1 m na přejezd vozíku o 10 m s IS filtrem



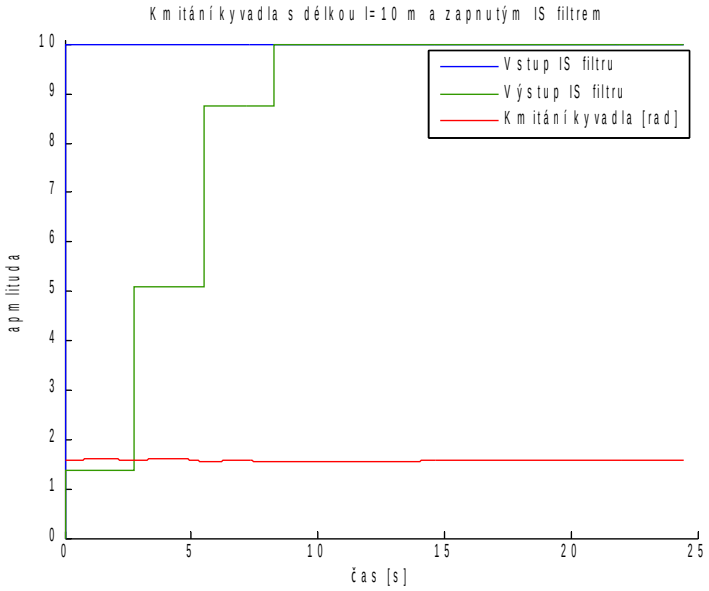
Obrázek 3.8: Odezva kyvadla l = 1 m na přejezd vozíku o 10 m bez IS filtru



Obrázek 3.9: Odezva kyvadla l = 1 m na přejezd vozíku o 10 m s IS filtrem



Obrázek 3.10: Odezva kyvadla l = 10 m na přejezd vozíku o 10 m bez IS filtru



Obrázek 3.11: Odezva kyvadla l = 10 m na přejezd vozíku o 10 m s IS filtrem

Kapitola 4

3D model systému

Ve virtuální laboratoři je k dispozici také 3D model portálového jeřábu. Parametry tohoto modelu (poloha vozíku, délka kyvadla, výchylky kyvadla) jsou svázány s hodnotami systému. Tím pádem se model pohybuje v závislosti na stavu systému a uživatel tak může názorně sledovat, co se se systémem děje.

V laboratoři jsou sice další možnosti, jak sledovat stav systému (například trendy výchylek kyvadla z rovnovážné polohy). Těmto možnostem nicméně chybí názornost, kterou si slibujeme od trojrozměrného modelu. Trendy slouží spíše k přesnému odečítání hodnot.

Model po dohodě vytvořila Ing. Kateřina Bícová v CAD systému CATIA. Tento model byl poté převeden do VRML jazyka postupem, který je blíže popsán dále v této kapitole. Po drobných úpravách modelu může Java v závislosti na vývoji veličin nastavovat pozici vozíku a natočení kyvadla a vznikne tak názorná vizualizace systému.

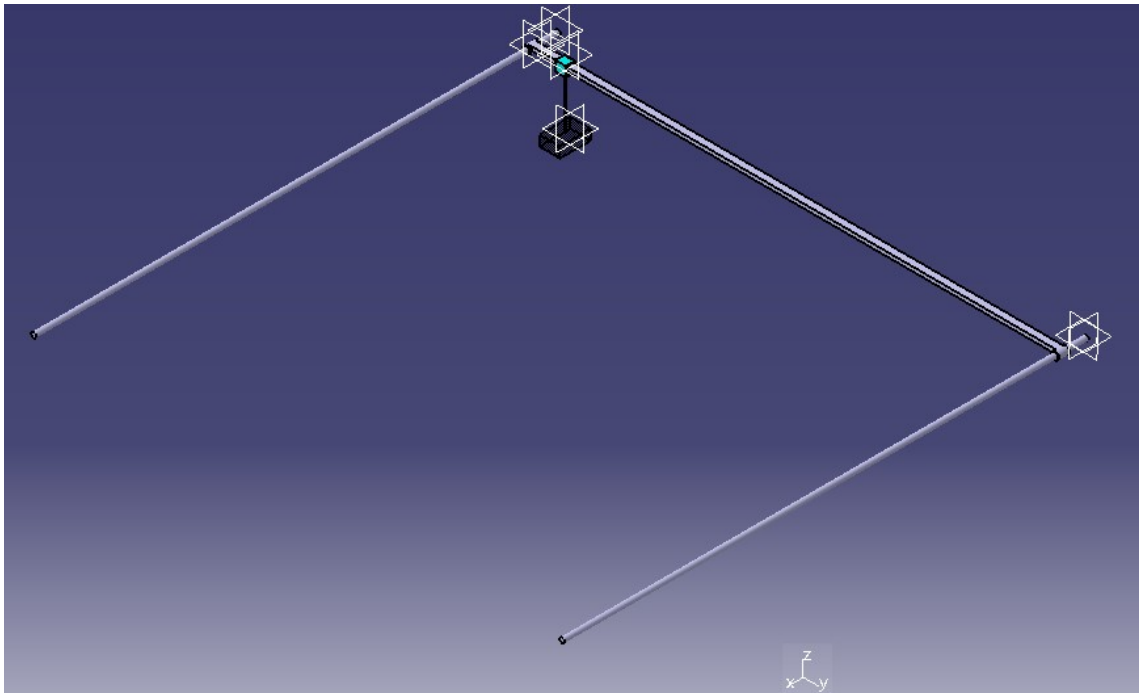
4.1 Popis modelu vytvořeného v CAD systému CATIAV5R19

Model je složen z několika pohyblivých částí. Základ tvoří dvě tyče kruhového profilu rovnoběžné s osou X. Tyto tyče vymezují pohyb vozíku v souřadnici X a jsou pevně ukotvené v okolním světě (nelze s nimi pohybovat). Po těchto tyčích se pohybuje další tyč, položená rovnoběžně s osou Y. Vozík je uložen na tyči Y a jeho pohybem po této tyči je zajištěn bod volnosti v ose Y. Na vozíku je zavěšeno lano a na něm náklad kvádrového tvaru. Lano je samostatnou částí modelu, jelikož se prodlužuje v závislosti na nastavené délce. Pokud by bylo součástí vozíku nebo nákladu, bylo by nutné zajistit, aby se tato část neprodlužovala s ním. Náklad je zvolen jako kvádr, aby na něm byla dobře viditelná aktuální výchylka kyvadla.

Na modelu není nijak řešeno umístění motorů, které je z hlediska řešení tlumení kmitání kyvadla nepodstatné, a proto bylo jejich umístění vynecháno. Model slouží pouze pro ilustraci aktuální situace a není v žádném případě určen jako výkres pro konstruování skutečného portálového jeřábu. Z toho důvodu v něm nejsou řešeny

žádné nosnosti. Nejsou zde řešeny ani žádné mechanické zábrany pro omezení pohybu v ose X v podobě například čepů na koncích tyčí. V práci je předpokládáno, že o zamezení vyklouznutí se poté postará sama virtuální laboratoř.

Model je z důvodu vývoje navržen v měřítku 1:10. Při nastavování polohy pohyblivých částí je tedy nutné polohu násobit jednou desetinou, což není problém. Maximální poloha v ose X je 10 m. K této hodnotě je nutné připočíst šířku tyče rovnoběžné s osou Y (v měřítku 20 mm), aby tato tyč při maximálním přejezdu nevyjela z uložení. Celková délka tyčí rovnoběžných s osou X bude tedy v měřítku 1 020 mm, což odpovídá 10 200 mm ve skutečnosti. Celkový pohled na vytvořený model je na obrázku 4.1.



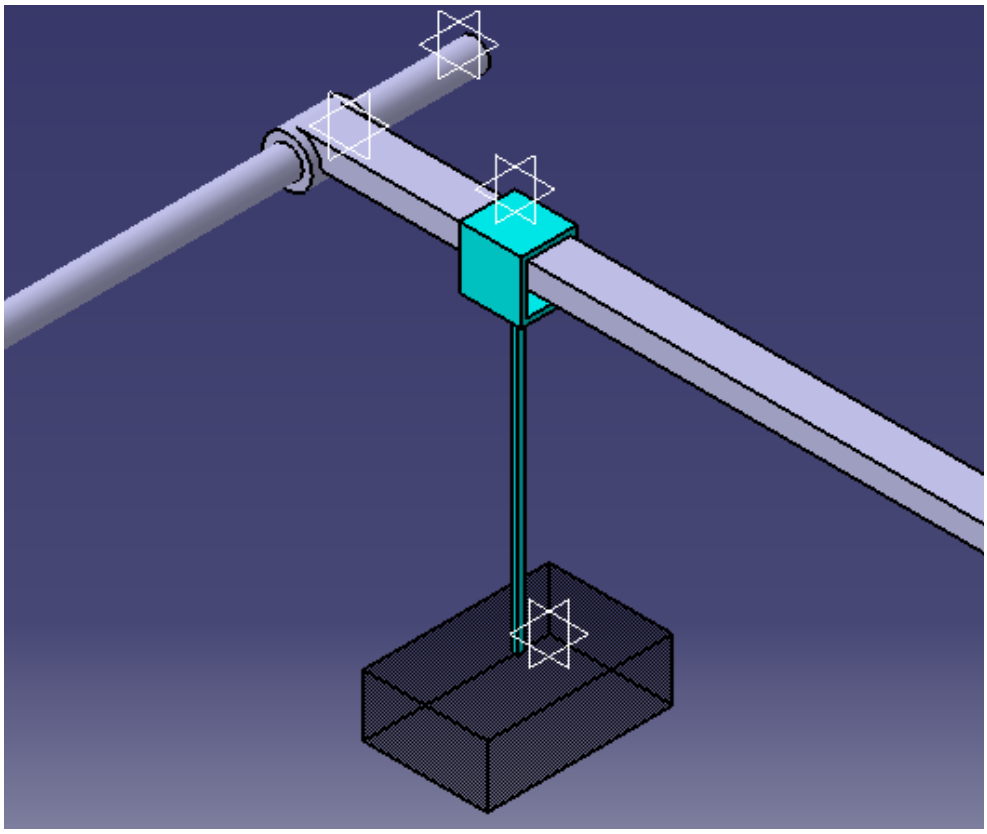
Obrázek 4.1: 3D model portálového jeřábu

Podobně je nutné přičíst pro pohyb vozíku v ose Y k maximální délce pojezdu i šířku vozíku v této ose, neboť jinak by vozík narážel do uchycení na konci tyče a jeho maximální pojezd by byl menší. Šířka vozíku je v měřítku 20 mm a výsledná délka tyče je tedy opět 1 020 mm, což je ve skutečnosti 10 200 mm. Po přičtení průměru tyče X (8 mm) dostáváme jejich vzdálenost 1 028 mm.

Z hlediska pozdějšího využití v programu při pohybu modelu jsou důležité ještě některé rozměry modelu. Tyto rozměry jsou uváděny tak, jak jsou nastaveny v modelu a pro jejich skutečnou velikost je nutné je přenásobit deseti.

počáteční délka lana	$z = 100\,\mathrm{mm}$
rozměry nákladu	$x = 60\,\mathrm{mm}, \mathrm{y} = 40\,\mathrm{mm}, \mathrm{z} = 20\,\mathrm{mm}$

Pro přesnější náhled situace je na obrázku 4.2 vidět detail uložení tyče Y na tyči X a je zde také detailně zobrazen náklad umístěný na vozíku.



Obrázek 4.2: Detailní pohled na uložení tyče Y a vozík s nákladem

Model je v souřadnicovém systému orientován v souladu s původním návrhem. Souřadnice v Java 3D jsou ve výchozím nastavení natočeny jinak (osa Z roste směrem k pozici pozorovatele). To však lze vyřešit posunem pozice pozorovatele (viz subsekce 4.2).

4.2 Jazyk VRML

Virtual Reality Modeling Language (dále jen VRML) je jazyk určený k popisu trojrozměrných scén a virtuální reality. VRML začal vznikat již v 80. letech a poslední verze VRML 2.0 (VRML 97) byla definována v roce 1997. Tato verze je specifikována normou ISO/IEC 14772-1:1997, což do jisté míry zaručuje stejné zobrazení v různých prohlížečích včetně balíku Java 3D. Prostorová tělesa se zde popisují pomocí seznamů souřadnic vrcholů a plochami určenými indexy svých vrcholů. Více o VRML, jeho struktuře a datových typech lze nalézt ve zdroji [16], odkud jsou čerpány informace pro tento odstavec.

K popisu trojrozměrného modelu slouží ve VRML stromová reprezentace. V souboru je nejprve nutné udat typ souboru a kódování, které je v souboru použito. Následují všeobecné metainformace o souboru a poté již následuje výše zmíněná stromová struktura. Kořenovým uzlem bývá virtuální svět, který má různé děti (children). Těmito dětmi jsou buď objekty, umístěné ve světě, ale můžou to být i zdroje světla osvětlující celou scénu, pozice pozorovatele, nebo nějaký senzor, který při nějaké události spouští ve světě nějaký specifický děj [17].

Každé těleso (definováno uzlem Shape) ve světě má další poduzly (děti), které definují jeho vlastnosti. Pro nastavení vizuálního vzhledu objektu slouží uzel appearance. V tomto uzlu lze nastavit například grafické vlastnosti materiálu, ze kterého je předmět vyroben (poduzel material), lze předmět pokrýt nějakou texturou atp. Dalším poduzlem objektu je uzel geometry, který definuje geometrické vlastnosti objektu [17].

Pro manipulaci s objekty slouží uzel Transform, který specifikuje použitelné lineární transformace objektu jako je posun, otočení a změna měřítka. Objekty, na které se mají definované transformace vztahovat, se umisťují do uzlu Transform jako děti (objektů může být i více). Konkrétní použití transformací a další informace o hierarchii stromu lze nalézt v [17].

Vzhledem k tomu, že nastavování vlastností materiálů těles není triviální, existuje na internetu spousta stránek, kde jsou definovány různé materiály. Materiály pro model jsme vybírali z [18].

Ve světě vytvořeném pomocí VRML se lze pohybovat. Programátor může zvolit způsob, jakým se bude uživatel moci pohybovat. V tomto případě nejsou respektovány hranice objektů a uživatel jimi může procházet. Pomocí uzlů Viewpoint lze předdefinovat nějaké body, ze kterých se může uživatel dívat na model systému. V každém takovém uzlu lze definovat orientaci natočení světa, pozici pohledu uživatele a šířku zorného pole. Body Viewpoint lze také přiřadit do Transform a svázat tak pohled s konkrétním objektem (viz otázka číslo 7 v [19]).

4.3 Převod modelu do VRML

V tomto případě je ruční vytváření modelu ve VRML složité, a proto byl volen postup nejdříve model vytvořit v CAD systému a následné ho převést do VRML.

Konverzi modelu do souboru typu VRML umí samotný systém CATIA V5R19. Nicméně výsledný soubor obsahuje položky, které Java 3D neumí přečíst, pojmenování definovaných těles je nepřehledné (například _05420368) a hlavně jednotlivé řádky vygenerovaného souboru jsou nepřehledně odsazované, což způsobuje špatnou orientaci v souboru při úpravách modelu.

K převodu modelu byl tedy použit komplikovanější, ale za to ověřený postup (viz [5]). V CATII byl model uložen ve formátu STEP (.stp). Tento formát by měl sloužit jako univerzální formát pro přenos modelů mezi různými CAD systémy. Nevýhodou je, že CATIA do tohoto formátu nepřenáší informace o materiálu, ze kterého jsou jednotlivé části vytvořeny. Model je jednobarevný, nicméně není problém poté ve VRML atributy materiálů dodefinovat (viz sekce 4.2).

Dalším krokem je využití programu CAD exchanger k exportu modelu do souboru typu X3D. Tento program také umožňuje převod přímo do VRML, je vhodné se však stále držet ověřeného postupu.

Posledním krokem při konverzi je využití programu Flux Studio 2.1 k převedení do konečného VRML souboru. Tento program má předdefinovanou volbu ukládání komprimované varianty, kterou Java 3D není schopna přečíst. Proto je potřeba tuto možnost odoznačit.

Ve VRML nyní nejsou nijak postihnuty žádné vazby. Například tedy při pohybu vozíku zůstane lano s nákladem na stejném místě místo toho, aby se pohybovalo s vozíkem. Lze samozřejmě posouvat jak s vozíkem, tak i s lanem a nákladem zvlášť. Přirozenější volbou je však dodefinovat tyto vazby. To lze jednoduše udělat například ve Flux Studiu tím, že vnoříme uzel Transform lana do transformace vozíku, tím je zaručeno, že při změně polohy vozíku se změní i poloha lana. I nadále přitom lze přistupovat přímo k transformačnímu uzlu lana pro jeho natáčení.

Nyní máme již vytvořen model v jazyku VRML. Tento model je nicméně před spuštěním ještě nutné upravit. V první řadě je nutné přidat body Viewpoint, bez kterých se model nespustí a které se tímto postupem nevygenerovaly. Je možné buď vytvořit vlastní body a nebo zkopírovat některé body ze souboru vytvořeného přímo z CATIE, která generuje krom ISO pohledu i pohledy z různých stran. Dále je potřeba pro jednotlivé součásti přepsat subuzel material, abychom nastavili vzhledy těles. Parametry materiálů byly vybrány z [18].

Pro následnou práci s modelem je nutné znát označení transformací pro jednotlivé části modelu. Tyče rovnoběžné s X jsou pojmenovány AxeX1T a AxeX2T, nicméně ty se v modelu pohybovat nebudou. Transformace určená pro pohyb tyče Y má označení AxeY1T, pro vozík Cart1T, pro provaz Rope1T a konečně transformace nákladu je označena jako Load1T.

Kapitola 5

Tvorba virtuální laboratoře

5.1 Virtuální laboratoř

Virtuální laboratoří je myšlena laboratoř, ve které se nepracuje s reálnými objekty (neprovádí se reálné pokusy), ale pracuje se s počítačovou reprezentací reality, kterou je nějaký model. Tento přístup má v porovnání s klasickou laboratoří mnoho výhod, například [20]:

- cena – například při opakovaných destrukčních zkouškách, ale často je nižší i pořizovací cena a cena údržby,
- rychlost – lze nastavovat rychlost virtuálního plynutí času a urychlit tak pomalé pokusy, či naopak prodloužit rychlé, aby obsluha byla schopna ovlivnit jejich průběh,
- bezpečnost – například lze zaškolovat obsluhu nebezpečných procesů,
- lze simulovat procesy za námi daných podmínek, což jinak nelze zajistit (například při zavádění nové technologie do provozu),
- možnost současného využití mnoha uživateli.

Při sestavování virtuálních laboratoří je však potřeba mít na zřeteli několik věcí, na které si musíme dát pozor. Například [20]:

- model – je nutné předem určit, co přesně se od laboratoře vyžaduje a podle toho vytvořit dostatečně kvalitní model reprezentující danou realitu, což může být v některých případech složité,
- problém ověření validity modelu - ideální je samozřejmě porovnat simulovaný proces s procesem reálným, to však ne vždy jde, je tedy vhodné provést různé simulace a konfrontovat je s fyzikálními zákony a empirickými vztahy,
- problém nepřesnosti a nestability numerických metod,
- výpočetní náročnost (v dnešní době již málo častý problém).

Virtuální laboratoře bývají často zaměňovány za vzdálené laboratoře. Důvodem je, že interakce s uživatelem probíhá často stejně. Dokonce jsou i laboratoře, které mohou fungovat jak jako virtuální, tak vzdálené, a je tedy použito pro oba druhy jedno a to samé HMI. Rozdíl mezi těmito laboratořemi je ten, že vzdálená laboratoř sice vzdáleně (zpravidla přes internet), ale přesto pracuje s reálným systémem a uživatel tak provádí skutečné experimenty.

Laboratoř jsme se rozhodli vytvořit v jazyce Java. Tento jazyk byl vybrán kvůli jeho nezávislosti na operačním systému a jeho celosvětovému rozšíření. Javu lze navíc spouštět pomocí internetového prohlížeče v podobě appletů. Díky tomu lze aplikaci spustit odkudkoliv pomocí internetu.

Jak je tedy vidět, virtuální laboratoře mají dvě hlavní části. První je model systému (experimentu), který byl sestaven v kapitole 2 Modelování systému a následně validován v kapitole 3 Simulace systému. Druhou částí je vrstva pro komunikaci mezi uživatelem a systémem - HMI (Human Machine Interface). Touto vrstvou se bude zabývat následující kapitola.

5.2 Human Machine Interface (HMI)

Human Machine Interface (dále již jen HMI) je, jak již název napovídá, ta část aplikace, která se stará o interakci s uživatelem. Jinými slovy zobrazuje aktuální stav systému, průběh vývoje jeho stavu v čase (trendy veličin) a dovoluje uživateli systém podle předem stanovených pravidel ovládat.

K tvorbě HMI byla využita již předdefinovaná kostra laboratoří v Javě s REXovským modelem (viz od sekce Obecná struktura programu až do konce této kapitoly). Toto si lze dovolit díky již předvytvořeným komponentám pro laboratoř (bloky JavaRexLib, panely pro vykreslování trendů, 3D model, atd. viz dále v kapitole). Z těchto komponent si stačí vybrat ty, které se hodí k dané aplikaci, a dodefinovat je podle našich potřeb.

V tomto konkrétním HMI může uživatel nastavovat délku kyvadla, polohu vozíku a frekvenci, kterou se bude snažit ZV filtr v systému potlačit. Tuto frekvenci je také možno automaticky spočítat a nastavit podle aktuální délky kyvadla, aby docházelo k co nejmenšímu kmitání. Je samozřejmě možné i ZV filtr vyřadit z provozu pro porovnání s průběhem netlumených kmitů v systému.

K zobrazování stavu systému slouží schéma modelu (částečně pro názornost fungování modelu a částečně pro nastavování jeho parametrů), trojrozměrný model jeřábu, který se pohybuje v závislosti na vypočtené poloze vozíku a výchylky kyvadla, a trendy výchylek kyvadla.

5.3 Programovací jazyk Java

Java prodělala od svého vzniku a speciálně za posledních patnáct let obrovský boom. V současné době lze nalézt Java aplikaci na jakémkoliv zařízení od čipových karet, mobilních telefonů, PDA, zabudovaných (embedded) zařízení, tabletů až po klasické počítače. Vzhledem ke své velké využitelnosti je Java rozdělena do několika edic v závislosti na tom, pro jaké zařízení je určena. Pro malá zařízení typu čipových karet je JavaCard, pro střední zařízení typu mobilních telefonů a zabudovaných zařízení je Micro Edition (ME). Java Standard Edition (SE) je určena pro klasické osobní počítače. Pro rozsáhlé projekty na více počítačích je určena Java Enterprise Edition (EE) [1], [21]. Více o jednotlivých distribucích se lze dočíst například v [10].

Java patří do kategorie moderních, objektově orientovaných jazyků. Mezi hlavní důvody jejího masivního rozšíření patří například to, že je volně přístupná a že obsahuje velké množství ucelených knihoven. Daleko větší výhodou je však její platformová nezávislost, díky čemuž může být použita na tolika různých druzích zařízení. Díky této vlastnosti lze snadno provozovat jako webová aplikace a je tedy pro nás vhodnou volbou kvůli možnosti vybudování vzdálené laboratoře [1].

Nezávislost Javy je zaručena díky tomu, že se jedná o interpretovaný jazyk, jelikož ke spuštění aplikací je nutné mít zprovozněnou na cílovém zařízení Java Virtual Machine (platí pro SE a EE). Programy nicméně nejsou interpretovány přímo ze zdrojového kódu (souboru *.java), nýbrž jsou ještě mezitím překládány do optimalizovaného, platformově nezávislého bytového kódu (*.class). Tento kód poté JVM interpretuje dané cílové platformě [1], [10].

5.4 JavaREX - Java rozhraní pro řídicí systém REX

Java rozhraní pro REX bylo vybudováno kvůli tvorbě přehlednějších aplikací pro pozorování stavu RexCore a jeho jednodušší obsluze, než je tomu u programu RexView. RexView je univerzální aplikace pro obsluhu jádra s jakýmkoliv programem, což je její ohromná výhoda, nicméně i velký zápor. Java rozhraní by mělo sloužit k snadné tvorbě HMI „ušitých“ na míru přímo jednotlivým řídicím algoritmům. Případně k obecnému vykreslování trendů z RexCore, jejich prohlížení a možnému převedení do Matlabu (viz aplikace RexTrend).

Největší výhodou rozhraní je platformová nezávislost, kterou zaručuje právě implementace v Javě. Díky tomu můžou být aplikace vystavěné na tomto rozhraní použity v nejrůznějších internetových prohlížečích a operačních systémech na klientských zařízeních [1].

Díky tomu, že REX používá ke komunikaci protokol XDG (viz níže), který je založen na standardu TCP, který je Javou plně podporován, lze pomocí JavaREX plně využít potenciálu REXu. Se spojením možností Javy v oblasti GUI vzniká silný nástroj pro tvorbu přehledných a uživatelsky líbivých vzdálených/virtuálních laboratoří [1]. Navíc díky tomu, že laboratoře budou využívat jádro REXu, poběží na rozdíl od mnoha jiných laboratoří ve skutečném čase (real-time) [2].

5.4.1 XDG protokol v REXu

Kvůli architektuře Klient- Server, která je v REXu použita z důvodu vzdáleného řízení procesů, běží i při simulaci řízení na stejném počítači procesy RexCore a ostatní aplikace REXu (nástroje pro diagnostiku, HMI, atd.) vždy v jiných procesech. Proto je nutné, aby mezi nimi docházelo ke komunikaci. Tato komunikace je zprostředkovávána pomocí XDG (reX DiaGnostic protocol), který je implementován ve většině programů REXu [1].

V laboratoři se sice XDG přímo nevyužívá, protože Panel čte hodnoty přímo z Tasku (viz sekce 5.7 a 5.10), nicméně je zde tato kapitola uvedena pro možné rozšíření laboratoře na vzdálenou laboratoř.

XDG obsahuje několik typů příkazů:

- Pro **připojení** – inicializace, ukončování a udržování běhu spojení,
- **Identifikační** – získávání verze REXu, hledání a převod symbolů REXu,
- Pro **nastavování/čtení** proměnných z bloků,
- **Skupinové** – určené k vytváření a odstraňování skupin proměnných, jejich čtení a zápisu do nich a jejich aktualizaci,
- **Objektové** – čtení bloků proměnných, diagnostických informací z I/O ovladačů, subsystémů, atd.,
- Příkazy k zobrazování a archivaci **trendů**,
- **Download/upload** příkazy – slouží k posílání a čtení souborů a konfigurace systému pro spouštění a zastavování řízení.

Všechny tyto příkazy kromě poslední skupiny mají svůj adekvátní protějšek i v JavaREX. Kromě třídy XDGProtocol jsou v Javě napsány další třídy určené k obsluze protokolu. Tyto třídy lze rozdělit do čtyř základních tříd:

- Třídy pro **komunikaci** – zde se nalézá samotná třída XDGProtocol a třídy RexInputStrem a RexOutputStream,
- Třídy **objektů REXu** – třídy definující typy proměnných (XAV), reprezentující časové značky (RexTStamp), implementující vlajky stavu objektů (RexFlags), verzi REXu (RexVersion), skupin hodnot veličin (RexGroup) a třídu pro uchovávání hodnot pro trendy RexTrend,
- Třídy **transparentních rexovských objektů** – třídy pro inicializaci, konfiguraci a obsluhu objektů,
- **Další** třídy– chybových hlášení (RexError), konstant (RexConst) a výjimek (RexServerException).

O implementaci a obsluhu protokolu se v laboratoři starají již vytvořené třídy a stačí již jen volat příslušné metody. Detailnější rozbor XDG tedy není nutný. Protokol XDG a jeho vlastnosti jsou podrobněji popsány v [1].

5.4.2 Struktura JavaREXu

Při spojování Javy a REXu dochází k tomu problému, že Java není jazyk určený k psaní aplikací reálného času, který je u řízení žádaný a který REX podporuje (hard real-time). JavaREX slouží k překonání tohoto problému a pomocí tohoto rozhraní lze vytvářet aplikace měkkého reálného času (soft real-time) [2].

K pochopení principu, jak JavaREX funguje, je potřeba si zhruba popsat, z jakých balíků se rozhraní skládá a jaký mají tyto balíky účel [2].

1. Globální část rozhraní (balík xGlobal)
   Tento balík obsahuje pomocné třídy potřebné k provozu ostatních tříd (například definice základních datových typů, struktur, konstant a chybových hlášek).

2. Jednoduché simulační jádro (balík xCore)
   Pro potřeby spouštění jádra v Javě a na Internetu bylo jádro REXu redukováno. Na rozdíl od původního jádra totiž nepodporuje xCore spouštění několika úloh (tasks) s různou prioritou a periodou vzorkování najednou, nýbrž dovoluje tvůrci laboratoře vytvořit maximálně jednu úlohu s libovolným počtem bloků a jejich propojením. Tato úloha je vložena do exekutivy a je spouštěna v Javě jako samostatné vlákno s danou periodou. To zaručuje, že jádro běží v měkkém reálném čase. Díky tomu lze interaktivně měnit periodu spouštění jádra a tím například zrychlovat dlouhé simulace, či zpomalovat rychlé. Manuální převod bloku do Javy bude ukázán v subsekci 5.7.1. Automatický převod je poté demonstrován v sekci 5.5. Nejdůležitějšími třídamy jádra jsou:

   Block je základní abstraktní třída pro funkční bloky a úlohy. Obsahuje metody init(), main() a exit(). Každý blok musí obsahovat jméno, vzorkovací periodu, referenci na úlohu, která blok vlastní a neměnný počet vstupů, výstupů, stavů a polí.

   Task rozšiřuje třídu Block o řídicí algoritmus sekvence bloků. Metody init(), main() a exit() volají příslušné metody jednotlivých bloků sekvence. Ve zděděné třídě se budou instance bloků umisťovat do metody init() pomocí metod declareBlock(), addBlock() a connectBlock().

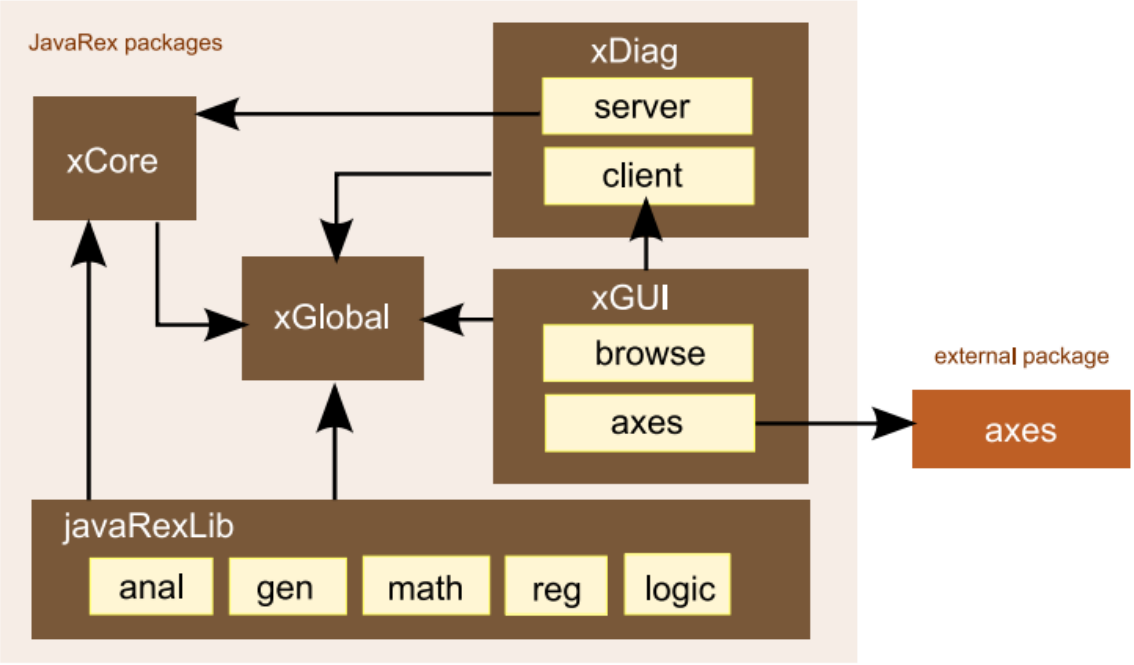
   XExecutive je rozšířením třídy Thread v Javě. Objekt třídy Task je vkládán do konstruktoru XExecutive. Vlákno se spouští metodou start() po zavolání metody initExecutive(). Třída periodicky spouští metodu main() úlohy, dokud se nezavolá metoda stopExecutive() nebo pauseExecutive(). Pokud byla exekutiva pouze pozastavena, lze ji následně opět spustit pomocí continueExecutive().

3. Knihovna funkčních bloků (balík JavaRexLib)
   Vzhledem k rozsáhlosti knihovny RexLib by bylo neefektivní tuto knihovnu překládat do Javy ručně. Významná část zmíněné knihovny je tedy překládána z jazyka C++ do Javy automatickou technikou popsanou v [2]. Díky tomu je zaručena konzistence mezi simulací, reálným časem a Javou. Oproti generování úloh (tasks) stačí knihovnu JavaRexLib vygenerovat a zkompilovat pouze jednou. Spolu s automaticky generovanými úlohami garantuje JavaRexLib stejné chování řídicích algoritmů na všech podporovaných platformách.

JavaRexLib je rozdělena do několika subbalíků stejně jako RexLib (anal, gen, logic, math, param a reg). Každý blok je zděděn od třídy Block, která je doplněna o definici vstupů, výstupů, stavů a polí. Každá proměnná má nastaveny meze a vyřešenu automatickou konverzi datových typů při spojování vstupů a výstupů bloků.

4. Komunikační a diagnostické rozhraní (balík xDiag)
   Tento balík obsahuje protokol XDG a třídy nutné pro jeho správnou funkci a je blíže představen v subsekci 5.4.1. Balík je zde doplněn o metody komunikace ze strany serveru kvůli snadnému připojení klienta k běžící instanci XExecutive(). Díky implementaci diagnostického rozhraní ze strany serveru lze snadno číst a zapisovat hodnoty proměnných v blocích dané úlohy.
   Vzhledem k tomu, že Java applety jsou soběstačné (self-contained), musí obsahovat obě části rozhraní XDG. Klientská část žádá server-část o čtení/zápis hodnot proměnných a server tyto příkazy provádí a informuje o výsledku. O periodické generování příkazů a aktualizaci GUI se stará samostatné vlákno vytvořené uživatelem. Tato část je sice v JavaREX připravena, nicméně se momentálně nevyužívá. Proto ještě nebyla úplně vytvořena část serveru.

5. Grafické uživatelské rozhraní (balík xGUI)
   Tento balík obsahuje některé komponenty REXu jako například komponenty pro jednoduché trendy a komponenty pro procházení struktury RexCore pomocí stromů.



Obrázek 5.1: Struktura rozhraní JavaREX (převzato z [2])

Grafické znázornění balíků rohraní JavaREX a vztahů mezi nimi je na obrázku 5.1. Tato subsekce je výtahem nejdůležitějších myšlenek z [2].

5.5 Převod modelu systému do jazyku Java

K převodu modelu ze souboru *.mdl do souboru *.java slouží program RexComp. Z modelu je vhodné odstranit všechny bloky TRND, neboť pro uchovávání a zobrazování trendů jsou v Javě již připravené třídy a tyto bloky jsou tam tedy zbytečné. Dále je také nutné odstranit bloky RexLang, protože RexComp v současné době neumí tyto bloky převádět. Funkce RexLang byla do programu doplněna později ručně.

K tomu, aby se model převedl, je také nutné, aby obsahoval blok SLEEP. Model se totiž převádí bez exekutivy, takže v něm chybí informace o tom, s jakou periodou má být diskretizován (RexComp model nepřeloží a ukončí se s chybovou hláškou). Informaci o periodě spouštění obsahuje právě blok SLEEP, proto ho je potřeba umístit na libovolné místo do modelu a nastavit mu již dříve určenou periodu 0,008 s. Blok SLEEP je vidět na obrázku 3.1 a je primárně určen pro zajištění spouštění modelu v reálném čase v systému Matlab/Simulink (viz [11]).

Výrazným zjednodušením pro převod modelu je skript JRLComp.bat. Tento skript spouští RexComp s pro nás vhodnými parametry a stačí mu zadat pouze jméno souboru s převáděným modelem. Toto jméno je zadáváno bez přípony souboru a výstupem je java-soubor se stejným jménem.

Pro správné přeložení je potřeba, aby měl RexComp k dispozici soubor JRLMacrs.h. Tento soubor je dobré dát buď do stejné složky jako je JRLComp.bat, upravit cestu v bat souboru a nebo přidat cestu k JRLMacrs.h do systému.

Vygenerovaným souborem je soubor Task (viz subsekce 5.7). Tento soubor obsahuje objekty tříd javaRexLib, a proto je nutné přidat tuto knihovnu do zdrojů projektu. Knihovna javaRexLib obsahuje javovskou verzi bloků REXu a v Tasku již tedy stačí vytvořit objekty těchto tříd s konkrétními parametry a pospojovat je ve funkční celek. To samozřejmě dělá RexComp podle konkrétního modelu.

Task slouží jako výpočetní jádro virtuální laboratoře. Vypočítává chování systému na základě interakce s uživatelem a poskytuje hodnoty veličin pro vizualizaci. Momentálně v něm však chybí ještě třídy bloků RexLang pro výpočet modelu kyvadla. Tyto třídy ještě v javaRexLib neexistují, a proto jsou vytvořeny v projektu. Funkce těchto tříd nejsou zatím volně programovatelné jako je tomu v REXu, ale jsou pevně naprogramovány podle našich požadavků. Vzhledem k pevnému naprogramování funkcí je nutné vytvořit pro každý blok RexLang jednu třídu, neboť oba bloky mají jiné funkce. Více je napsáno níže v subsekci 5.7.1.

5.6 Obecná struktura laboratoře

Základní kostra HMI laboratoře byla vytvořena již dříve a nám stačí ji dodefinovat podle našich požadavků a potřeb. Význam základních tříd laboratoře je nastíněn zde a podrobně popsán v následných subsekcích.

Jádrem každé virtuální laboratoře našeho typu je třída **Task**. Tato třída slouží pro simulaci chování reálného systému podle specifikovaného a otestovaného modelu, který jsme převedli z klasického REXu. Obsahuje definici typů jednotlivých bloků, jejich parametrů a jejich vzájemného propojení [3].

Task je spouštěn z **Frame**, což je třída obalující GUI panel laboratoře, aby mohla být laboratoř spouštěna jako desktop aplikace [3]. Náš Frame je rozšířením XGFrame, což je vlastně na míru REXu upravený JFrame (zobrazovací okno - základní startovní bod každé grafické aplikace v Javě využívající Swing). Konstruktor této třídy přiřadí virtuální laboratoři právě námi definovaný Task a poté instanci třídy SingleTrendApplicationPanel, která byla vytvořena pro zobrazování průběhu simulace a pro možnost interakce s uživatelem.

Třída **Applet** obaluje GUI panel Java appletem, aby mohla být laboratoř vložena na webovou stránku [3]. Třída je rozšířením třídy XGApplet, což je na míru upravená Swingovská třída JApplet.

Třída **SingleTrendApplicationPanel** a její konkrétní rozšíření Panel obsahuje panel, který zobrazuje virtuální laboratoř. Panel se poté rozpadá na panely pro řízení systému uživatelem, panel pro možné měnění parametrů některých bloků modelu (zde je zobrazeno schéma modelu), panel pro 3D vizualizaci jeřábu a jeho aktuálního stavu a v neposlední řadě panel pro zobrazování trendů některých veličin (v našem případě výchylek kyvadla z rovnovážné polohy a polohy vozíku).

Ve schématu modelu je možné přidávat displeje pro zobrazovaní hodnot signálů. Výstup každého bloku může být navíc vložen do panelu pro vykreslování trendů [3].

Všechny nápisy v laboratoři jsou vloženy ve zdrojovém (resources) souboru. Díky tomu může být velmi snadno provedena lokalizace laboratoře bez zásahu do zdrojového kódu [3].

5.7 Task

Třída Task byla sice vygenerována automaticky, chybí v ní ale bloky RexLang, které jsou součástí modelu kyvadla a které RexComp zatím nedokáže převést do Javy. Tento převod je nutné provést manuálně a postupuje se stejně, jak by postupoval program RexComp. Nicméně funkce bloků jsou napevno dodefinovány ručně. Popis tvorby třídy bloku je uveden v subsekci 5.7.1. Zde je uvedeno, jak přidat již vytvořenou třídu do úlohy, což může sloužit jako návod pro přidání jakéhokoliv bloku manuální cestou.

V metodě init() jsou nejdříve vytvořeny instance tříd a deklarují je zavoláním declareBlock(). To je potřeba, aby úloha znala počty proměnných daného bloku [3]. Po přidání všech instancí je nutné zavolat metodu allocateMemory(), která přiděluje pole vstupů, výstupů, stavů a bufferů pro všechny deklarované bloky [3]. Příslušný segment zdrojového kódu je uveden níže.

Deklarace bloků

```
RexLangQ12 q12=new RexLangQ12("q12");declareBlock(q12);
RexLangQ22 q22=new RexLangQ22("q22");declareBlock(q22);
allocateMemory();
```

Dále je v metodě init() nutno deklarované bloky přidat, nastavit jejich parametry a poté je inicializovat. Bloky RexLang nemají žádné nastavitelné parametry kromě periody vzorkování. Příklad přidání, nastavení parametrů a inicializace bloku je demonstrován na prvním bloku.

Přidání bloku q12 a jeho inicializace

```
addBlock(q12);
q12.setPeriod(getPeriod());
q12.init(bWarmStart);
```

Jako poslední je nutné vytvořenou instanci bloku propojit s ostatními. K tomu slouží metoda connectBlock(), které jsou předávány čtyry celočíselné parametry. První parametr určuje pořadové číslo bloku, ze kterého signál vychází. Druhý určuje pořadové číslo výstupního portu daného bloku. Třetí určuje pořadové číslo bloku, do kterého signál vchází. Poslední parametr určuje pořadové číslo vstupního portu tohoto bloku. Pořadová čísla bloků jsou závislá na tom, v jakém pořadí byly bloky přidány pomocí metody addBlock(). Je důležité mít na paměti, že pořadová čísla bloků i portů jsou počítána od nuly.

Příklad přivedení signálu z prvního (a jediného) výstupu bloku ADD3 na první vstup bloku REXLANG_1 (v Task blok q12) podle obrázku 3.1 je uveden níže. Blok ADD3 je v pořadí čtrnáctý přidaný blok. Q12 je celkově padesátý devátý přidaný blok.

Propojení bloků ADD3 a q12

```
connectBlock(13,0,58,0);
```

5.7.1 Třídy RexLang

Jako první je při doplňování bloku RexLang nutné vytvořit třídu, která bude tento blok reprezentovat. Třída je vytvořena ze třídy bloku ADD, protože obě mají více vstupů a pouze jeden výstup. Stačilo tedy pouze dodefinovat zbylé vstupy a funkci bloku. Pro každý RexLang byla definována nová třídu. Jelikož se ale liší pouze v definici hlavní metody, je postup tvorby demonstrován pouze na jednom z bloků.

Pro funkci RexLang samozřejmě neexistují ve zdrojovém souboru zatím jména bloku a dostatečného počtu vstupních portů. Je tedy zatím použito jméno bloku ADD. První čtyry vstupní porty jsou označeny jako "IDS_BLK_U1_ANAL"až "IDS_BLK_U4_ANAL", a protože v současné době není ve zdrojovém souboru počítáno s tím, že by měl blok více jak čtyři vstupy, zbylé tři jsou označeny opět jako "IDS_BLK_U1_ANAL"až "IDS_BLK_U3_ANAL". Toto označení si lze dovolit, protože je v zobrazovaném schématu celý model řízené soustavy nahrazen z důvodu úspory místa jedním blokem (viz subsekce 5.10.2).

V poli XInInitVar[] xInInitVars je tedy definováno sedm analogových desetinných vstupů. Jako příklad je níže uvedeno definování prvního portu.

První vstupní port

```
new XInInitVar("u1",
  getResource(RexConsts.JAVAREXLIB_RESOURCES, "IDS_BLK_U1_ANAL"),
  new XInCfg(
    (short) (XInVar.XCFI_INPUT | XInVar.XCFI_IMPL_CONST_VAR | 0),
    (short) (XAV.XATM_XANY_VAR), XAV.MIN_DOUBLE, XAV.MAX_DOUBLE,
    new XInVar(XAV.XATC_XDOUBLE)))
```

Pole výstupních portů XOutInitVar[] xOutInitVars zůstává nezměněné. Nicméně další změna je provedena v konstruktoru třídy, kde je potřeba nastavit do proměnné, kolik vstupů blok bude mít. To lze provést příkazem nExtInCount = 7.

Nyní již stačí dodefinovat funkci bloku, aby pomocí hodnot přiváděných na vstupy počítal námi požadovaný výstup. Tato funkce je definována v metodě main(). Jelikož je funkce bloku v REXu napsána v jazyce podobném jazyku C, byla tato funkce pouze zkopírována a následně upravena pro Javu. Metoda zde z důvodu rozsahu výpočtové funkce není uvedena.

5.8 Frame a Applet

Třídy Frame a Applet obě slouží k obalení virtuální laboratoře, aby mohla být dále spouštěna buď jako desktopová aplikace, případně aby mohla být umístěna na webové stránky.

Obě třídy jsou velmi jednoduché jde u nich pouze o přiřazení správného Tasku, GUI panelu a zdrojového souboru s popisky v laboratoři. Rozdíl při přiřazování je pouze v tom, že ve Frame se provádí přímo v konstruktoru a v Appletu se provádí v metodě init(), kde se ještě navíc provádí inicializace všech komponent v appletu a panelu pomocí initComponents(). Příkazy pro přiřazení jsou uvedeny níže.

Přiřazovací příkazy

```
setTask(new PendulumTask(PendulumPanel.TITLE_STR));
setPanel(new PendulumPanel((Task) getTask()));
setSettingsBundle(ResourceBundle.getBundle(
  "pendulum.pendulum_settings"));
```

5.9 Canvas3D

Tato třída se stará o načtení trojrozměrného modelu ze souboru *.wrl do Java 3D API, které poté umožňuje s modelem v závislosti na hodnotách proměnných v jádře pohybovat. Tím vzniká trojrozměrná vizualizace aktuálního stavu řízeného systému, která na rozdíl od průběhu trendů dává jasnější představu o tom, co se s jeřábem děje.

K načtení VRML modelu je v projektu nutné mít přidánu knihovnu j3d-vrml97, kterou lze získat například v [22]. Dále je zapotřebí samotné API, které je dostupné například na [23].

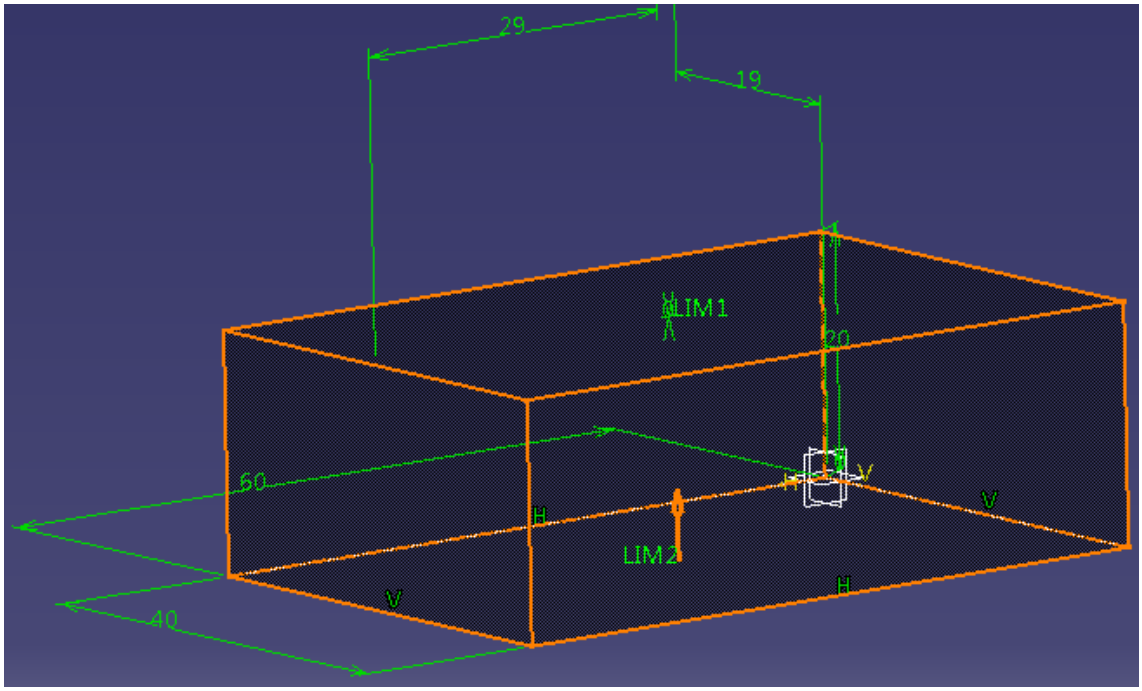
VRML model je načítán ze souboru *crane.wrl*, který obsahuje model popsaný v kapitole 4. Pro umožnění pohybu jednotlivých částí modelu je nutné znát jména jednotlivých transformací. Ty jsou popsány v posledním odstavci sekce 4.3.

Model byl vytvořen v měřítku 1:10, proto jsou všechny translační pohyby násobeny hodnotou **double** SCALE = 0.1. Dále je nutné mít na paměti, že počátky lokálních souřadnic vozíku a modelu nejsou umístěny ve středu objektu, ale v jeho rohu. Pozice se vždy nastavuje vzhledem k lokálnímu počátku objektu, proto je potřeba k této hodnotě přičítat či odčítat hodnotu, aby se na požadovanou pozici dostal střed objektu. Ve třídě jsou tedy nastaveny v metrech hodnoty potřebných rozměrů objektů (viz níže), o jejichž jednu polovinu se většinou upravují hodnoty pozic. Tyto hodnoty jsou nastaveny z pohledu laboratoře a pro účely modelu se tedy musejí také přenásobit měřítkem.

Potřebné rozměry objektů

```
private static final double LOAD_Z_LENGTH = 0.2;
private static final double CART_Y_LENGTH = 0.18;
```

Celá situace je demonstrována na nákladu jeřábu, jehož nákres je na obrázku 5.2. Bílé čtverce vyznačují jednotlivé roviny a jejich průnik pak bod počátku. Výška nákladu je 20 mm, čili 0,02 m a jelikož je model vyobrazen v měřítku 1:10, je výsledná hodnota 0,2 m. Stejným způsobem byl určen i rozměr vozíku v ose Y.



Obrázek 5.2: Zobrazení rozměrů nákladu

Díky vytvoření vazeb ve VRML již není potřeba se starat o většinu pohybů (například pohybu lana s vozíkem), což velmi redukuje počet těchto potřebných konstant.

5.9.1 Java 3D

Java 3D je API určené k vytváření programů pro vykreslování a interakci trojrozměrných prostředí. Jedná se o standardní rozšíření Java2SE SDK a nabízí k dispozici kolekci vysokoúrovňových konstrukcí pro vytváření trojrozměrné geometrie, pro manipulaci s ní a samozřejmě její vykreslování. Pomocí tohoto API lze vytvářet obrázky, vizualizace, animace a interaktivní 3D aplikace. [24]

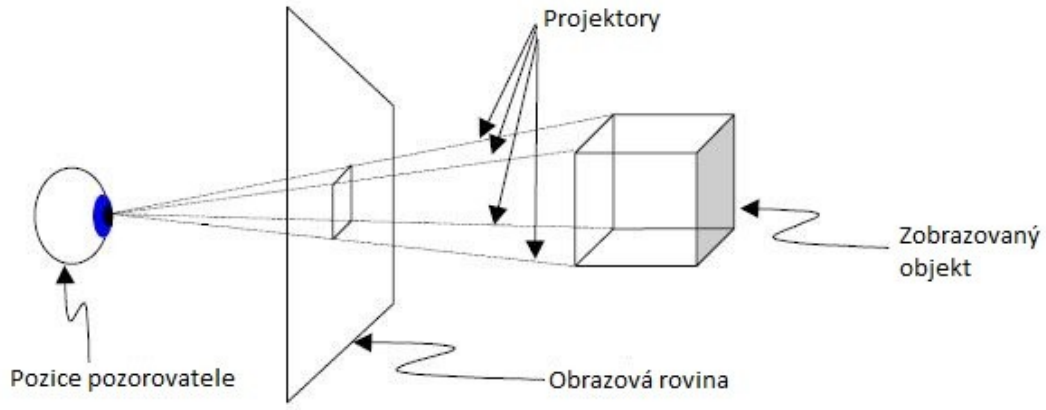
Každé prostředí v Java 3D je virtuálním vesmírem, který může dosahovat astronomických rozměrů, ale mít také rozměr v řádu velikosti molekul. Záleží jen na tvůrci, jak tento vesmír nadefinuje a co do něj umístí. Do vesmíru lze umisťovat objekty z hierarchie tříd Java 3D, či si pomocí nich vytvářet objekty své vlastní. Fyzikální vztahy mezi těmito objekty (geometrie, zvuk, světla, orientace a umístění) a vlastnosti vesmíru jsou definovány v tzv. grafu scény. [24]

Graf scény je datová struktura složená z uzlů a referencí. Uzly jsou ve scéně instance tříd Java 3D. Reference jsou vztahy mezi instancemi. Nejběžnějším vztahem

je vztah rodič-dítě. Uzel může mít libovolný počet dětí, ale vždy jen jednoho rodiče. List poté nemá žádné dítě. Druhým možným vztahem je odkaz (reference). Odkaz spojuje objekt NodeComponent s grafem scény Node. NodeComponent definují geometrické a vzhledové atributy použité k vykreslení objektů. [24]

Vytváření vždy celého prostředí od úplného začátku je velmi náročné a neefektivní. Z toho důvodu v API existuje třída SimpleUniverse, jejíž instance má již předdefinované některé obecné charakteristiky každého prostředí a není tedy potřeba je vždy znovu definovat. Je zde vytvořen téměř celý graf scény a stačí dodefinovat jednotlivé objekty, které se budou ve scéně nacházet. Tato třída však neobsahuje více pohledů ve virtuálním vesmíru. [24]

Vizuální část větve grafu scény obsahuje tzv. obrazovou rovinu, na kterou se promítá zobrazovaná část vesmíru (viz obrázek 5.3). Vesmír je zobrazován na rovinu tak, jak by na něj nahlížel uživatel z definované pozice. Toto zobrazování znázorňují na obrázku projektory. [24]



Obrázek 5.3: Vztah mezi obrazovou rovinou, pozicí pozorovatele a vesmírem (převzato z [24])

Podle výchozího nastavení je obrazová rovina umístěna v počátku vesmíru. Osy jsou v Java 3D definovány následujícím způsobem: Osa X je osou horizontální, jejíž kladná poloosa je orientována směrem doprava. Osa Y je vertikální úsečka v obrazové rovině jejíž hodnoty rostou směrem nahoru. Osa Z roste směrem k pozici pozorovatele. Počátek (0, 0, 0) je umístěn v centru obrazové roviny. [24]

Objekty se definují v takzvané obsahové větvi grafu scény. Ta je formována v metodě createSceneGraph(). Všechny objekty jsou umisťovány do prostředí jako děti kořenového uzlu, který metoda createSceneGraph() poté vrací.

Mezi nejdůležitější třídy Java 3D patří hlavně třídy BranchGroup, TransformGroup a Transform3D, které jsou níže krátce popsány.

BranchGroup je základním stavebním kamenem k tvorbě grafů scény. Její instance jsou kořeny subgrafů. Tyto objekty jako jediné objekty mohou být dětmi ob-

jektů Locale (referenčních bodů, k nimž se vztahuje pozice objektů). Mohou mít libovolný počet dětí. Děti objektu BranchGroup mohou být objekty Group nebo Leaf. [24]

TransformGroup je podtřídou třídy Group. Její instance se používají k vytvoření grafů scény a obsahují kolekci dětí, což jsou reference na objekty, na které je následně uplatňována daná transformace. Transformace je typicky vytvořena objektem Transform3D, který není objektem grafu scény. [24]

Transform3D je třída určená pro provádění základních afinních geometrických transformací, jako je posun, rotace nebo změna měřítka (zkosení je vynecháno). Tyto transformace jsou definovány například v [26]. Matice definující jednotlivé transformace v trojrozměrném prostoru jsou uvedeny v [27]. Tyto objekty se obyčejně používají pouze k definování transformace objektu TransformGroup, či k jeho vytváření. [24]

Posun objektu se obvykle provádí posunutím jedné, či více souřadnic vektoru. Tento vektor může být reprezentován například třídou Vector3d (viz [28]), který lze z objektu Transform3D získat metodou `get()`.

Rotace podél příslušné osy se provádí metodami `rotX()`, `rotY()` nebo `rotZ()` příslušné instance Transform3D. Úhel je zadáván v radiánech a rotace je prováděna vždy kolem osy procházející počátkem.

Změna měřítka je přirozeně používána k zvětšování, či zmenšování objektů. K měnění měřítka slouží metoda `setScale()`, které stačí předat reálnou hodnotu. Tato hodnota uvádí, kolikrát se objekt zvětší vzhledem k současné situaci.

Je důležité mít na paměti, že pokud je prováděno s objektem několik transformací, záleží na jejich pořadí. Je velký rozdíl, pokud je objekt posunut a následně otočen, či nejdříve otočen a následně posunut. To lze logicky odůvodnit tím, že postupné aplikování transformací je vlastně násobení matic jednotlivých transformací. Pořadí transformací a tím i matic ovlivňuje konečnou matici [27].

Pokud je požadována současná rotace objektu ve dvou směrech, je k tomu potřeba dvou instancí třídy Transform3D. Při rotaci v jedné ose a následné rotaci v druhé vyjde totiž jiný výsledek, než při současném otočení v obou. To lze opět dokázat na násobení matic. Správného výsledku je dosaženo, pokud je otočena každá instance v jedné ose a výsledné matice jsou poté vynásobeny. Pro násobení disponuje Transform3D metodou `mul()`.

Více o Java 3D a jeho používání se lze dočíst například na [24], [29], případně na stránkách autora tohoto API [25].

5.9.2 Interaktivní model portálového jeřábu

Při vytváření grafu scény je práce velmi usnadněna díky převodu modelu z VRML. Celá obsahová větev grafu scény je importována ze souboru pomocí níže uvedeného kódu a není potřeba ji tedy ručně vytvářet. Druhý řádek ukládá do Javy 3D body prohlížení (ViewPoints) nadefinované ve VRML. Tyto body se nahrávají ve stejném pořadí, v jakém jsou uvedeny ve VRML a nezáleží na tom, jestli je nějaký bod uvnitř uzlu Transform, či je definovaný mimo stromovou strukturu.

Import grafu scény

```
Scene s~= loadVRML(this.getClass().getResource(VRML_PATH));
this.setViewGroups(s.getViewGroups());
```

Ve vizualizaci jsou definovány celkem tři pohledy, které lze měnit. Prvním je detailní pohled na vozík. Druhý pohled je nastaven tak, aby byl vidět celý model i s pohybem vozíku. Posledním pohledem je ISO pohled, kde je opět vidět celý model. Z důvodů popsaných v subsekci 5.10.3 nelze mezi těmito pohledy v laboratoři přepínat a je jako výchozí nastaven druhý pohled, nicméně ostatní pohledy jsou k dispozici pro možné pozdější využití.

Plátno také může umožňovat všechny možné pohledy ze stran. To je však připraveno již v předchůdci Canvas3D a není tedy potřeba se o vytváření těchto pohledů starat.

Nyní již pouze stačí tuto subscénu přiřadit kořenovému uzlu, který vrací metoda createSceneGraph(), ve které byl graf definován. V metodě createSceneGraph() jsou nadefinovány ještě jednotlivé transformace, kterými je uskutečňován pohyb objektů. Příklad přidání transformace pro tyč Y je uveden opět níže. Podobným způsobem je vytvořena instance TransformGroup všech objektů.

Určení transformace pro tyč Y

```
axeY1T = loadTransformGroup(s, AXEY1_NAME);
```

Nyní je již v laboratoři umístěn trojrozměrný model portálového jeřábu. K tomu, aby se však mohl model pohybovat, je nutné určit vztahy, podle kterých se má pohyb vykonávat. Model se pohybuje v závislostech na aktuálních hodnotách veličin REXu a měl by se úměrně rychle aktualizovat. O to se stará metoda refresh(), do které jsou dodefinovány potřebné vztahy.

Plátno Canvas3D předpokládá, že hodnoty veličin REXu jsou předávány v konstuktoru v pořadí, v jakém jsou uvedeny v tabulce níže. Jelikož je k nim v objektu XAV přistupováno jako k poli pomocí indexů, je nutné, aby bylo dané pořadí zachováno.

Jméno	Blok:Port	Poř. číslo v XAV
Délka kyvadla	l:ycn	0
Pozice vozíku X	SINT_7:y	1
Pozice vozíku Y	SINT_10:y	2
Výchylka nákladu ψ	q12:u1	3
Výchylka nákladu ϕ	q12:u3	4

Níže je ukázáno na příkladu pohybu vozíku, jak lze nastavovat polohu objektu a tím jím posouvat. Hodnota polohy získaná z REXu je násobena konstantou SCALE, čímž je prováděna úprava měřítka (viz sekce 4.1). Při posunu v ose Y je navíc k hodnotě přičítána polovina délky vozíku, aby vozík nebyl při nulové poloze uprostřed tyče. Je dobré si všimnout, že při pohybu vozíku se staráme pouze o pohyb v ose Y. Pohyb v ose X je již totiž zajištěn posouváním tyče Y po tyčích X.

Pohyb vozíku

```
// moving pendulum cart
cart1T.getTransform(tmpT3D);
// Save translation
tmpT3D.get(tmpV3D);
tmpV3D.y = SCALE*(getXAV(2).toXDOUBLE()+CART_Y_LENGTH/2);
// Put translation back
tmpT3D.setTranslation(tmpV3D);
cart1T.setTransform(tmpT3D);
```

Podobně je posouváno s tyčí Y. Její poloha se však mění pouze v ose X. Označení tyč Y je v tomto případě možná trochu zavádějící, protože se tyč pohybuje ve směru X. Nicméně toto označení říká, že tyč umožňuje pohyb vozíku ve směru Y, nikoliv že se samotná tyč v tomto směru pohybuje.

Pohyb lana je způsobován jednak pohybem vozíku, s nímž se posouvá, a jednak kmitáním kyvadla, podle nějž se v prostoru natáčí. Pohyb s vozíkem je zajištěn tím, že transformace lana je dítětem transformace vozíku. Natáčení lana budeme realizovat pomocí metod rotX() a rotY(). Současného natočení v obou osách je poté dosaženo vynásobením výsledných transformačních matic. Níže je opět uveden ukázkový kód. Je však vynecháno získávání a ukládání transformací, které je stejné.

Natáčení kyvadla

```
tmpT3D.rotZ(getXAV(3).toXDOUBLE()-Math.PI/2);
tmpT3D2.rotX(Math.PI-getXAV(4).toXDOUBLE());
tmpT3D.mul(tmpT3D2);
```

Lano lze dynamicky prodlužovat změnou měřítka, které je závislé na hodnotě nastavené délky. Metoda setScale() umožňuje měnit měřítko pro každou osu zvlášť, čehož je využito a lano je prodlužováno pouze v požadované ose. To je vidět v kódu níže.

Změna velikosti lana

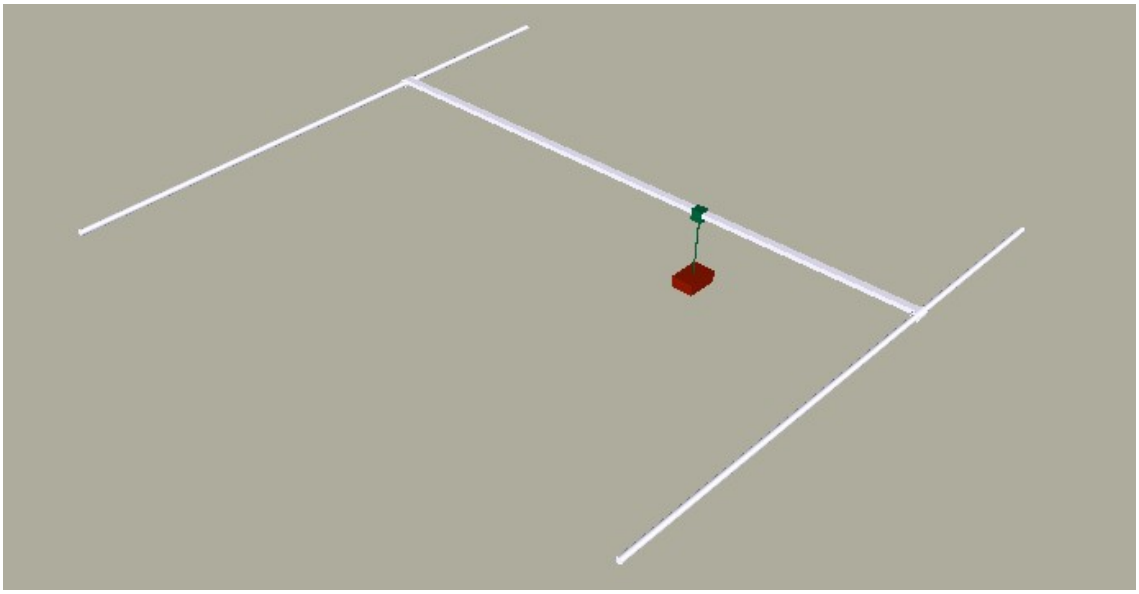
```
tmpT3D.setScale(new Vector3d(1,1,getXAV(0).toXDOUBLE()));
```

Nepříjemným důsledkem umístění transformace nákladu v transformaci lana je, že se změnou měřítka lana se mění i měřítko nákladu a ten se tak také prodlužuje. Je nutné ho tedy změnit opět zpátky na původní velikost. Tím, že náklad má počátek lokálních souřadnic na své spodní straně, bude při změně na původní velikost posunut níže, než by měl, proto je potřeba ho následně vrátit na původní místo.

Změna velikosti lana

```
// Reduction load back
tmpT3D.setScale(new Vector3d(1,1,1/getXAV(0).toXDOUBLE()));
// Save translation
tmpT3D.get(tmpV3D);
tmpV3D.z = -SCALE*(1+LOAD_Z_LENGTH/getXAV(0).toXDOUBLE());
```

Výsledná vizualizace portálového jeřábu v Java 3D je na obrázku 5.4.



Obrázek 5.4: Výsledný model v Java 3D

5.10 Panel

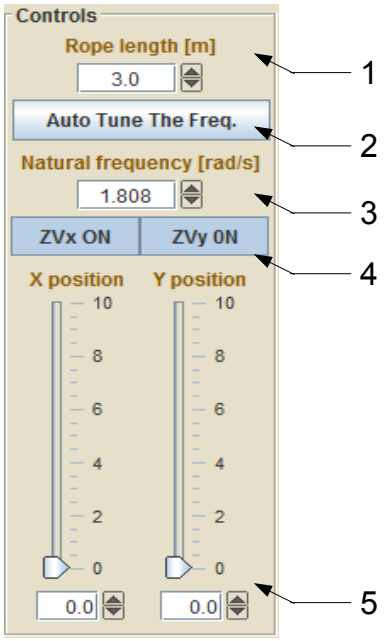
Tato třída slouží k propojení tříd Task, Canvas3D a panelu pro zobrazování trendů v jeden funkční celek. Navíc v ní jsou definovány panely pro ovládání simulace a panel obsahující interaktivní schéma zapojení. Tyto komponenty jsou inicializovány v metodě initComponents().

Panel této laboratoře je rozšířením třídy SingleTrendApplicationPanel a je doplněn o ActionListener kvůli přenastavování potlačované frekvence v blocích ZV při zmáčknutí tlačítka.

Vzhledem k množství prvků v panelu je laboratoř nastavena tak, aby se otevírala v okně 1020x650 px. Poměry jednotlivých panelů se dají samozřejmě uživatelem měnit, nicméně k plnému zobrazení je nutné minimálně takovéto rozlišení obrazovky.

5.10.1 Ovládací panel

První věcí, kterou může uživatel měnit, je délka provazu (kyvadla). Ke změnám slouží vstupní panel (číslo 1 na obrázku 5.5). Jak bylo napsáno již dříve, délka kyvadla může mít jakoukoli hodnotu v intervalu <0.1;10> m. Tato hodnota se může měnit i v průběhu simulace. S touto hodnotou se samozřejmě interaktivně mění i délka kyvadla v 3D vizualizaci a příslušná hodnota v Tasku.



Obrázek 5.5: Ovládací panel

Pod vstupním panelem s délkou kyvadla se nachází tlačítko pro možnost automatického výpočtu tlumené frekvence v závislosti na aktuálním rozměru kyvadla a nastavení této frekvence do IS filtrů (č. 2). Tuto frekvenci může nastavovat i uživatel ručně v intervalu $< 0; 10^{10} >$ rad/s. K ručnímu nastavení frekvence slouží vstupní pole číslo 3. Dvojice tlačítek níže (č. 4) slouží k vypínání a zapínání filtrů, aby měl uživatel možnost porovnat způsob řízení s filtrem a bez filtru. Pro každou osu je určeno jedno tlačítko. K definici cílové polohy vozíku slouží dvojice posuvníků (č. 5), kde se dá hodnota nastavit buď přímo pomocí políčka pod posuvníkem, či si uživatel může vyzkoušet řízení ručně, aby mohl porovnat své výsledky s generovaným řízením filtru. Rozsahy posuvníků jsou pevně stanoveny na interval <0;10> m, aby vozík nevyjel ze svého uložení.

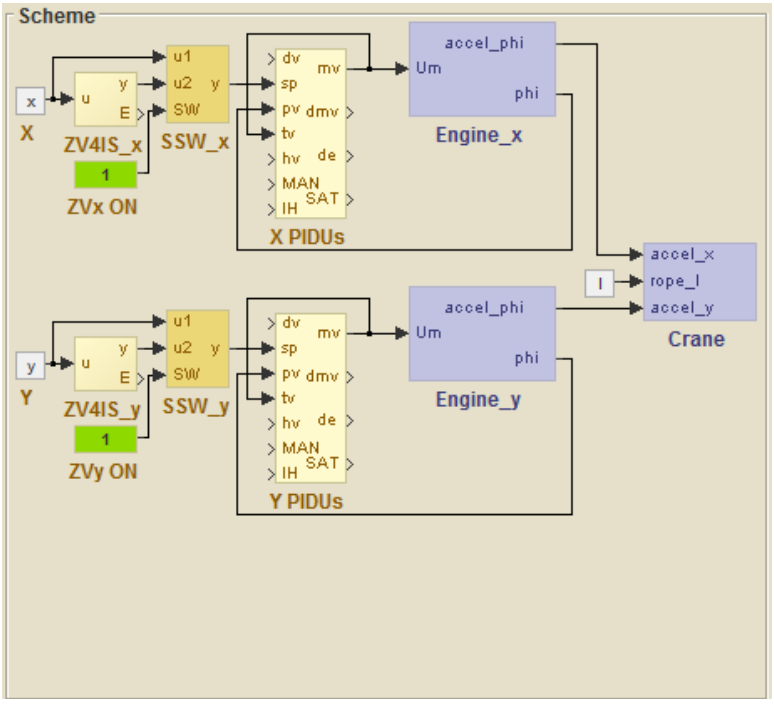
5.10.2 Panel obsahující interaktivní schéma zapojení

Pro lepší pochopení fungování simulace modelu je v laboratoři zakomponován panel se zjednodušeným schéma zapojení. Schéma je zjednodušené v tom smyslu, že jsou zde modely motorů a jeřábu nahrazeny symbolickými bloky. Dále jsme bloky určené k provádění druhé derivace polohy vozíku integrovány do bloku motoru, kterému byl přidán výstup zrychlení vozíku. Bloky kaskádní regulace byly nahrazeny jedním symbolickým blokem. Tato opatření bylo nutné provést kvůli grafické rozsáhlosti schématu.

Blok reprezentující model motoru je ve třídě XGEngine a blok modelu portálového jeřábu reprezentuje třída XGCrane. Obě tyto třídy jsou rozšířením třídy XGConnectableObject a jsou vytvořené na míru našim požadavkům (mají stejné pojmenování vstupů a výstupů jako v modelu REXu). Tyto třídy jsou v projektu umístěny v balíku xGUI.

Ve schématu je možné některé bloky otevřít a nastavit jejich parametry. Tato možnost je ale povolena pouze u bloků filtrů a bloků logické hodnoty, která obsluhuje stav přepínače pro odpojení filtrů. Cílem této laboratoře je prezentovat techniky tlumení a proto uživatel nebude mít přístup k nastavení bloků regulace nebo parametrů motoru, aby bylo zajištěno jejich konstantní chování.

Schéma je umisťováno do instance třídy XGPanel pomocí objektů XGComponent, které jsou k dispozici pro tvorbu laboratoří v rozhraní JavaREX v balíku xGUI. Bloky jsou postupně definovány a umisťovány do panelu a poté propojovány signálovými čarami, čímž je vytvářeno potřebné schéma. Výsledné zapojení je vidět na obrázku 5.6.



Obrázek 5.6: Panel s interaktivním schéma zapojení

Přidávání jednotlivých bloků je velmi jednoduché a lze ho rozdělit do několika základních kroků. Tyto kroky jsou níže podrobně popsány a jsou přidány i ukázky zdrojových kódů.

1. Vytvoření instance bloku
   Krom v této práci vytvořených tříd pro prezentaci motorů a jeřábu je potřeba definovat objekty tříd ConstBlock, CNBBlock a XGBlock. Konstruktor tříd je několikanásobně přetížený, nicméně nejčastěji jsou používány tři typy konstruktorů popsané níže. Jsou ukázány konstruktory třídy XGBlock. Konstruktory CNBBlock a ConstBlock však umožňují předávat stejné parametry.

   Konstruktory bloků

   ```
   XGBlock(Block block, int left, int top)
   XGBlock(Block block, int left, int top, int width, int height)
   XGBlock(Block block, int left, int top, int width, int height,
           int orientation)
   ```

   Block je reference na blok v Tasku, který má instance reprezentovat. Následuje pozice levého rohu bloku v pixelech vzhledem k počátečním souřadnicím panelu. Další hodnotou je pozice horního rohu a může následovat šířka, výška a orientace bloku. V současné době jsou k dispozici orientace zleva doprava (hodnota 0) a zprava doleva (hodnota 1). Příklad vytvoření instance je níže.

   Vytvoření bloku přepínače

   ```
   SSW_xBlock=new XGBlock((SSW) ((Task) task).getBlock("SSW_x"),
           SCHEMA_LEFT_OFFSET+90 , SCHEMA_TOP_OFFSET+46);
   ```

2. Přidání bloku do panelu
   Nyní když je blok vytvořený, lze ho přidat do panelu. To je provedeno pomocí metody XGPanel.add(), které se předává reference na přidávaný blok, jak je uvedeno níže.

   Přidání bloku do panelu

   ```
   schemePNL.add(SSW_xBlock);
   ```

3. Propojení bloků
   K propojení bloků slouží třída SignalLine, které se v konstruktoru předávají pouze reference na spojované bloky a řetězce s názvy portů, které se mají spojit. Často se ale stává, že čára se vygeneruje nějak nevhodně a je potřeba ji nějak upravit. To umožňuje metoda setPoints(), které se předávají dvě celočíselná pole. V prvním jsou souřadnice X všech bodů, ve který čára mění směr. V druhé jsou obdobně umístěny souřadnice Y. Metoda setDots() umožňuje v uzlech vytvořit spojovací tečky. Čáru poté opět přidáme do panelu.

   Vytvoření čáry a její přidání do panelu

   ```
   SignalLine xBlock2SSW_xBlock = new SignalLine(xBlock, "y",
           SSW_xBlock, "u1");
   xBlock2SSW_xBlock.setPoints(new int[]{SCHEMA_LEFT_OFFSET+30,
           SCHEMA_LEFT_OFFSET+30,SCHEMA_LEFT_OFFSET+90},
           new int[]{88,63,63});
   xBlock2SSW_xBlock.setDots(new boolean[]{true,false,false});
   schemePNL.add(xBlock2SSW_xBlock);
   ```

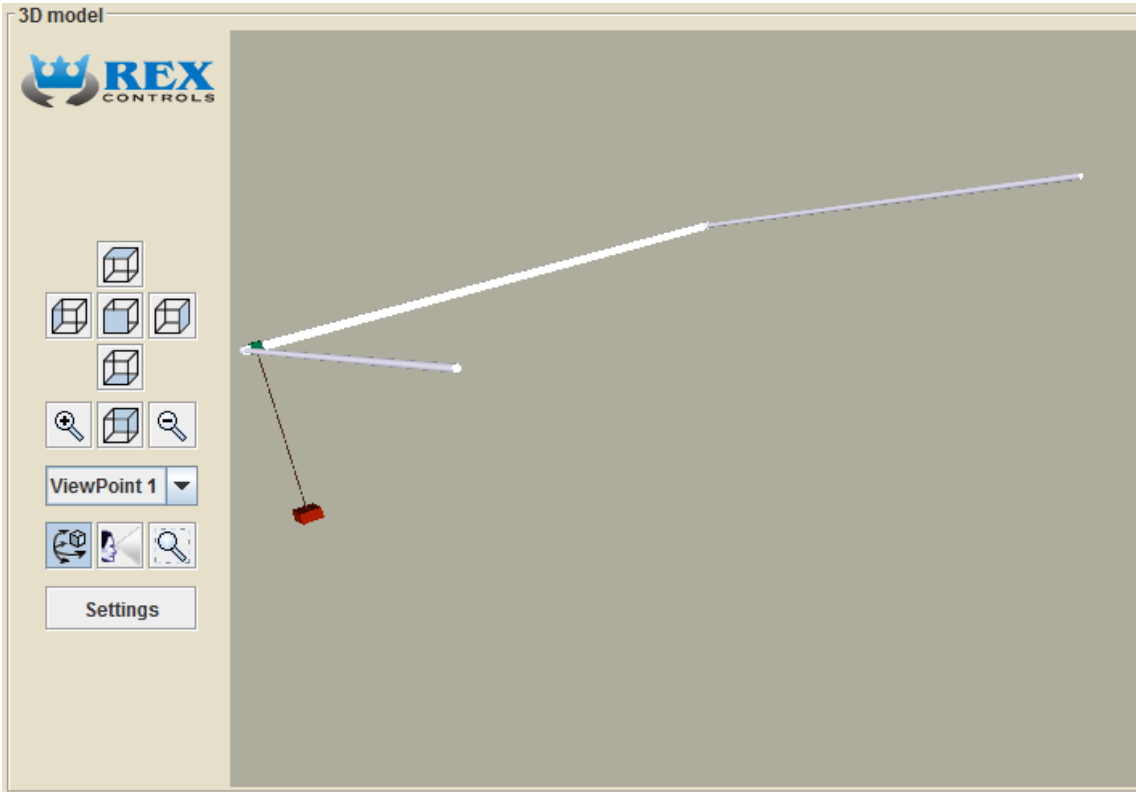
5.10.3 Panel pro vizualizaci 3D modelu

Přidání vizualizace systému do virtuální laboratoře je velmi jednoduché. V podstatě stačí vytvořit instanci třídy Canvas3D a tu pak přidat do panelu určeného k zobrazování této vizualizace. Při volání konstruktoru je pouze nutné zadat veličiny ve správném pořadí, jak je uvedeno v tabulce v subsekci 5.9.2. Vytváření instance je vidět v kódu níže.

Vytváření instance Canvas3D

```
canvas3D = new PendulumCanvas3D(new String[]{
            "l:ycn","SINT_7:y","SINT_10:y","q12:u1","q12:u3"
            }, trendPanel);
```

V předchůdci plátna, který je součástí rozhraní JavaREX, je již vytvořeno základní ovládání pro manipulaci s modelem. Ovládání je na levé straně plátna (viz obrázek 5.7) a obsahuje například tlačítka pro zobrazení modelu ze všech bočních pohledů nebo pro jeho přibližování/oddalování. Jelikož je ale model vytvořen s jinak orientovanými souřadnicemi, než je v Java3D zvykem, jsou pohledy ze stran pod jinými tlačítky, než by měly být. Proto je boční panel skryt příkazem canvas3D.remove(1);. Pohyb v modelu lze realizovat i pomocí šipek na klávesnici a myši, proto je postranní panel bez správných pohledů zbytečný.



Obrázek 5.7: Panel pro vizualizaci 3D modelu

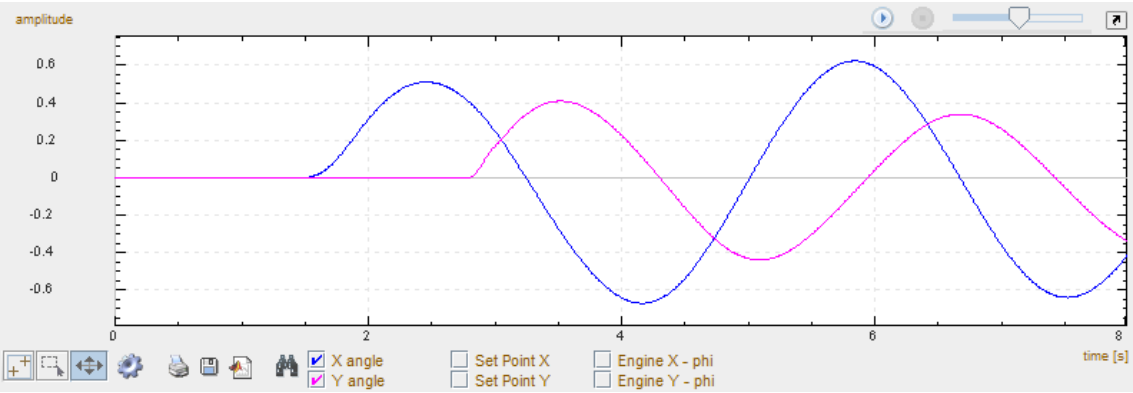
5.10.4 Panel pro zobrazování trendů

Poslední součástí laboratoře je panel pro zobrazování trendů. Tento panel je unifikován pro všechny laboratoře vytvářené pomocí JavaREX, a proto je jeho konstrukce obsažena již v předchůdci třídy Panel. V konkrétní laboratoři stačí pouze v konstruktoru třídy Panel předat reference veličin, jejichž průběhy mají být vykreslovány, a případně pozměnit nastavení vlastností panelu trendů.

V tomto konkrétním panelu trendů se budou zobrazovat úhly kyvadla, výstupy ZV filtrů a výstup motorů. K jejich pojmenování je využit zdrojový soubor s názvy v aktuálně používaném jazyce (viz kód níže). Panel trendů obsahuje samozřejmě menu pro ovládání panelu a pro export dat například do Matlabu (viz obrázek 5.8). Důležitou komponentou v panelu trendů je také posuvník pro zrychlování, či zpomalování simulace, který je umístěn v pravé horní části.

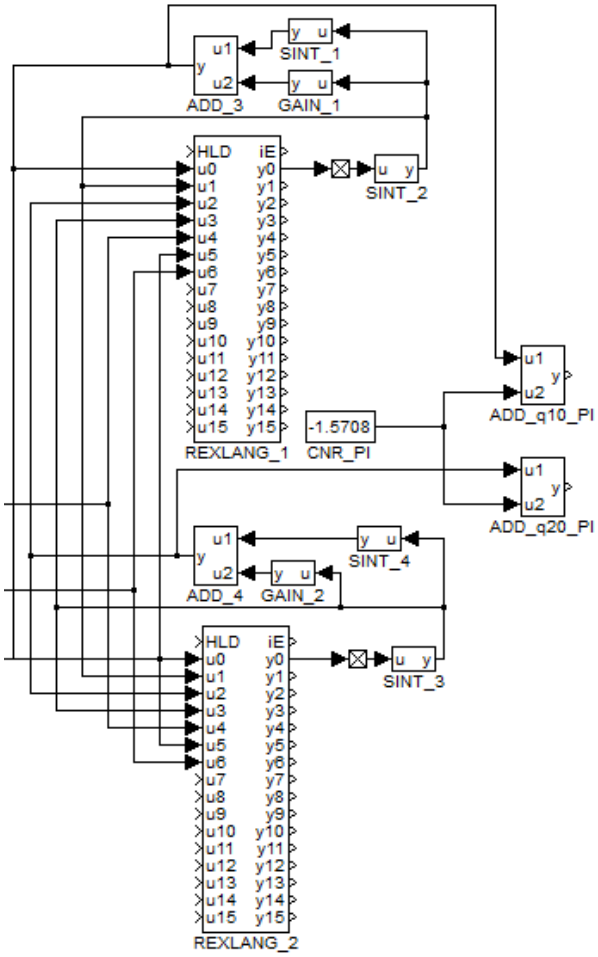
Využití zdrojových souborů k pojmenování proměnných

```
trendPanel.trendCHBPanel.getCheckBoxAt(0).setText(
        PENDULUM_RESOURCES.getString(
                "PendulumPanel.PHI_ANGLE_TITLE"));
trendPanel.trendCHBPanel.getCheckBoxAt(1).setText(
        PENDULUM_RESOURCES.getString(
                "PendulumPanel.PSI_ANGLE_TITLE"));
```



Obrázek 5.8: Panel pro zobrazení průběhů veličin

Rovnovážné polohy úhlů jsou sice na hodnotě $\frac{\pi}{2}$ rad, pro uživatele to ale může být matoucí a nepřehledné. To je důvod, proč jsou hodnoty při vykreslování trendů posunuty tak, aby rovnovážná poloha kyvadla byla pro oba úhly 0 rad. Toto posunutí bylo provedeno v modelu v REXu tím, že byl použiti blok ADD, ve kterém se sčítá aktuální hodnota úhlu a konstanta $-\frac{\pi}{2}$ (viz obrázek 5.9). Panelu trendu jsou předávány výstupy těchto bloků ADD.



Obrázek 5.9: Posunutí výchylek úhlů o $-\frac{\pi}{2}$ rad

Kapitola 6

Závěr

V této práci byla vytvořena interaktivní virtuální laboratoř prezentující techniky tlumení vibrací u mechanických systémů. K prezentaci byl jako vhodný příklad systému vybrán portálový jeřáb (sekce 2.1), na kterém byly předvedeny techniky tlumení vibrací pomocí tvarování vstupu (viz sekce 2.4). Byl odvozen nelineární model systému, navržen vhodný model motorů k řízení systému (více sekce 2.2) a podle [13] určeny vhodné parametry regulátorů k regulaci motorů (sekce 2.3). Systémy byly nasimulovány v řídicím systému REX a následně odzkoušeny v kapitole 3.

Výsledný simulační model byl převeden na jádro virtuální laboratoře (viz sekce 5.5). V kapitole 4 je popsána příprava interaktivního trojrozměrného modelu portálového jeřábu, který byl následně umístěn do laboratoře. Tato laboratoř byla vytvořena na základě již předem definované kostry laboratoří v rozhraní JavaREX (viz kapitola 5.1).

Z hlediska zadání práce odpovídá prvnímu bodu zadání sekce 2.4. Druhému bodu odpovídají sekce 2.1, 2.2 a 2.3. Vypracování třetího bodu reprezentuje sekce 5.5. Grafické uživatelské rozhraní požadované ve čtvrtém bodě je vytvořeno v kapitolách 4 a 5.1.

V laboratoři může uživatel nastavovat parametry bloku tlumení vibrací a v závislosti na kmitání nákladu jeřábu sledovat účinnost tohoto tlumení. Uživatel může nastavovat potlačovanou frekvenci a typ tlumícího filtru. Blok tlumení lze též úplně odstavit. Potlačovanou frekvenci lze automaticky nastavit tak, aby byla vzhledem k aktuální délce kyvadla optimální a náklad jeřábu tak kmital co nejméně. Průběh je zobrazován v interaktivním modelu a v panelu trendů. Při nastavení optimální frekvence dochází k výraznému útlumu vibrací, což je vidět v sekci 3.3.

Matematický model řízeného systému nebyl vytvářen s předpokladem, že by se kyvadlo mohlo přetočit. Z toho důvodu se při přetočení občas stává, že kyvadlo zůstává kmitat v horní polorovině. To je způsobeno problémem popsaným v kapitole 2.1.2, konkrétně zjednodušením rovnice 2.4.

Nastavování cílové polohy vozíku bylo koncipováno tak, aby si uživatel mohl vyzkoušet sám řízení a porovnat své výsledky s řízením generovaným tvarovacím filtrem.

Cílem práce bylo vytvořit názorný nástroj spustitelný přes internet ve webovém prohlížeči, který prezentuje techniky tlumení pomocí tvarování vstupu systému v systému REX. Tento cíl byl splněn.

Úloha by mohla být doplněna o nějakou přesnější metodu řešení obyčejných diferenciálních rovnic, než je Eulerova metoda, čímž by bylo dosaženo přesnějších odhadů proměnných v simulačním modelu. Dala by se využít například metoda Runge- Kutta čtvrtého řádu. V REXu by však musel být přidán další task pro počítání hodnot veličiny v mezikrocích a musel by se vyřešit problém převodu na jádro virtuální laboratoře, které může obsahovat pouze jeden task.

Další možností rozšíření laboratoře je doplnění možnosti řízení reálného systému, až bude k dispozici. Tím by laboratoř mohla fungovat buď jako virtuální, či jako vzdálená, přičemž by vše bylo ovládáno ze stejného HMI.

Literatura

[1] *Balda Pavel, Čech Martin* **Java interface to REX control system**
2006

[2] *Balda Pavel, Čech Martin* **A new technique for automatic generation of Java applets for web-based control education**
Strbske Pleso 2009

[3] *Čech M., Balda P., Schlegel M., Severa O.* **New virtual internet laboratories presenting advanced control schemes**
Eger, Hungary 2010

[4] *REX Controls s.r.o* **PID Controller Laboratory**
http://test1.rexcontrols.cz/en/home

[5] *Ondřej Severa, Martin Čech, Pavel Balda* **New tools for 3D HMI development in Java**
2011

[6] *Goubej M., Škarda R., Schlegel M.* **Input shaping filgers for the control of electrical drive with flexible load**
2009

[7] *Wikipedia:* **Lagrangeovská formulace mechaniky**
http://cs.wikipedia.org/wiki/Lagrangeova_pohybová_rovnice#Lagrangeovy_rovnice
11. 4. 2012

[8] *Schlegel, M.:* **Mechanické systémy**
7.2.2007

[9] *REX Controls s.r.o.:* **Začínáme se systémem REX**
Verze 2.03 (revize 2033),
Plzeň 15.9.2011

[10] *Reitinger, J.:* **Vizualizace průmyslových procesů v Java ME**
Plzeň 2010

[11] *REX Controls s.r.o.:* **Funkční bloky systému REX- Referenční příručka**
Verze 2.03 (revize 2033),
Plzeň 15.9.2011

[12] *cs.wikipedia.org:* **Gravitační zrychlení**
http://cs.wikipedia.org/wiki/Gravitační_zrychlení
Poslední modifikace 19.1.2012

[13] *Krejčí, Alois:* **Regulace servopohonů v mechatronických systémech**
Plzeň 2010

[14] *Kupka, Libor:* **Matlab & Simulink- studijní materiály pro předmět Základy kybernetiky**
Liberec 2008

[15] *Schlegel M., Goubej M.:* **Feature-based Parametrization of Input Shaping Filters with Time Delays**
Liberec 2008

[16] *Tišnovský, Pavel:* **VRML: Jazyk pro popis virtuální reality**
http://www.root.cz/clanky/vrml-jazyk-pro-popis-virtualni-reality/
8. 11. 2007

[17] *Tišnovský, Pavel:* **Tvoříme trojrozměrné scény v jazyce VRML2**
http://www.root.cz/clanky/tvorime-trojrozmerne-sceny-v-jazyce-vrml-2/
15. 11. 2007

[18] *Rousel, David:* **VRML materials**
http://vrmlstuff.free.fr/materials/
15. 11. 2007

[19] *Crispen, Bob:* **comp.lang.vrml FAQ Answers 5. Techniques**
http://vrmlworks.crispen.org/faq/faq5.html
11. 6. 1998

[20] *ZČU, Katedra kybernetiky:* **Přednášky k předmětu Modelování a simulace 1**

[21] *Wikipedia:* **Java (programovací jazyk)**
http://cs.wikipedia.org/wiki/Java_(programovací_jazyk)

[22] *12 Demo Source and Support:* **Internetové stránky java2s.com**
http://www.java2s.com/Code/Jar/j/Downloadj3dvrml97jar.htm

[23] *Oracle:* **Internetové stránky java.net**
http://java3d.java.net/binary-builds.html

[24] *Dredwerkz:* **J3DT_1 - Začínáme**
http://www.dredwerkz.cz/java_3d_tutorial/J3DT_1.htmll

[25] *java3d.org:* **Tutorial**
http://www.java3d.org/tutorial.html

[26] *Studenti Univerzity Palackého v Olomouci:* **Geometrie/Afinní transformace souřadnic**
http://cs.wikibooks.org/wiki/Geometrie/Afinní_transformace_souřadnic
8. 12. 2011

[27] *Petr Minařík:* **Teorie 3D prostoru**
http://programovani.net-mag.cz/?action=art&num=446

[28] *Oracle:* **Class Vector3d**
http://docs.oracle.com/cd/E17802_01/j2se/javase/technologies/desktop/java3d/forDevelopers/J3D_1_3_API/j3dapi/javax/vecmath/Vector3d.html

[29] *Richard G. Baldwin:* **Understanding Transforms in Java 3D**
http://www.developer.com/java/other/article.php/3717101/Understanding-Transforms-in-Java-3D.htm
18.12.2007

[30] *REX Controls s.r.o* **Řídicí systém REX**
http://www.rexcontrols.cz/rex

Seznam obrázků