VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH TRAJEKTORIE A ŘÍZENÍ LINEÁRNÍHO JEŘÁBU
LINEAR CRANE TRAJECTORY DESIGN AND CONTROL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jozef Krakovský**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Michal Bastl**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Jozef Krakovský**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**Ing. Michal Bastl**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Návrh trajektorie a řízení lineárního jeřábu

Stručná charakteristika problematiky úkolu:

Úkolem práce je navrhnout vhodný popis systému lineárního jeřábu. Dále se předpokládá vytvoření rozsáhlé rešerše na možnosti návrhu trajektorie a řízení, tak aby nedocházelo k velkým výkyvům přemisťovaného břemene. Student implementuje simulační modely k ověření možností různých přístupů. Předpokládá se využití řešení okrajového problému, genetické nebo Monte–Carlo metody a prediktivní řízení. Nakonec provede experimenty i prakticky na reálném laboratorním modelu lineárního jeřábu.

Cíle diplomové práce:

1) Popište problém a proveďte rešerši v podobě průzkumu literárních zdrojů v oblasti zájmu.
2) Vytvořte vhodný matematický model lineárního jeřábu. Na základě rešerše vyberte tři metody, které popište důkladně.
3) Vytvořte simulace vybraných metod. Předpokládá se použití nástrojů MATLAB/Simulink. Modely a algoritmy důkladně popište a vyzkoušejte různá nastavení a parametry simulací.
4) Proveďte experimenty na testovacím laboratorním zařízení, které bude dostatečně reprezentovat lineární jeřáb. Experimenty zhodnoťte a vyvoďte z nich závěry.

Seznam doporučené literatury:

TAKÁCS, Gergely a Martin GULAN. Základy prediktívneho riadenia. Bratislava: Slovenská technická univerzita v Bratislave, 2018. ISBN 9788022748261.

FRANKLIN, Gene F., J. David POWELL a Michael L. WORKMAN. Digital control of dynamic systems. 3rd ed. Menlo Park, Calif.: Addison-Wesley, c1998. ISBN 0201331535.

ASTROM, Karl J. a Richard M. MURRAY. Feedback systems: an introduction for scientists and engineers. 2008. Princeton: Princeton University Press, c2008. ISBN 0691135762.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Táto práca sa zaoberá problematikou riadenia lineárnych mostových žeriavov vybranými tromi metódami. V teoretickej časti oboznamuje so základnými štruktúrami jednotlivých algoritmov a základnými vzťahmi. V jadre práce sú riadiace algoritmy odsimulované v prostredí MATLAB pomocou vytvorených simulačných programov. V závere práce sú následne algoritmy aplikované na laboratórnom modely, ktorý dostatočne reprezentuje lineárny žeriav a algoritmy sú zhodnotené na základe dosiahnutých výsledkov

Kľúčové slová

Genetický algoritmus, Problém okrajových hodnôt, Prediktívne riadenie, GA, BVP, MPC, Lineárny žeriav

Abstract

This thesis deals with control of linear bridge cranes using three selected methods. In theoretical part, it gives information about basic structure of each selected algorithm and basic mathematical relations. In the middle, control of algorithms is simulated using created simulation programs in MATLAB. After that, the algorithms are applied on laboratory model of linear crane and in the end all of them are evaluated according to achieved results.

Key words

Genetic algorithm, Boundary value problem, model predictive control, GA, BVP, MPC, linear crane

Bibliografická citácia

KRAKOVSKÝ, Jozef. *Návrh trajektorie a řízení lineárního jeřábu.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/125125. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Michal Bastl.

Čestné prehlásenie

Prehlasujem, že som diplomovú prácu na tému *Návrh trajektorie a řízení lineárního jeřábu* vypracoval samostatne pod vedením vedúceho diplomovej práce a s využitím uvedených podkladov a odbornej literatúry.

V Brne dňa 26.6.2020 ……

Bc. Jozef Krakovský

Poďakovanie

Týmto by som rád poďakoval vedúcemu diplomovej práce Ing. Michalovi Bastlovi za odbornú pomoc a cenné rady pri riešení tejto práce.

Obsah

Zoznam obrázkov **7**
Zoznam tabuliek **8**
1. Úvod **9**
2. Formulácia problému **10**
2.1 Mostový žeriav 10
2.2 Anti-swing 10
2.3 Popis dynamiky systému 10
2.4 Pohybové rovnice 11
2.5. Linearizácia 13
3. Metódy riadenia **14**
3.1 Genetický algoritmus 14
3.1.1 Inicializačná generácia 14
3.1.2 Hodnotiaca funkcia 15
3.1.3 Selekcia 15
3.1.4 Reprodukcia 15
3.1.5 Mutácia 17
3.2 Problém okrajových hodnôt 18
3.2.1 MATLAB a problém okrajových hodnôt – bvp4c 18
3.3 Prediktívne riadenie 21
3.3.1 Predikčné matice 23
3.3.2 Účelová funkcia 24
3.3.3 Penalizačné matice 25
3.3.4 Kompaktná účelová funkcia 26
3.3.5 Procesné obmedzenia 27
3.3.6 Kvadratické programovanie 28
4. Aplikácia algoritmov v MATLABe **30**
4.1 Návrh genetického algoritmu 33
4.1.1 Výsledky simulácie 35
4.2 Návrh algoritmu problému okrajových hodnôt 37
4.2.2 Výsledky simulácie 39
4.3 Návrh algoritmu prediktívneho riadenia 42
4.3.1 Výsledky simulácie 44
5. Experiment **46**
5.1 Laboratórny model lineárneho žeriava 46
5.1.1 Konštrukcia modelu 47
5.2 Úprava pohybových rovníc pre laboratórny model 49
5.3 Odhad parametrov 51

5.4 Riadenie silou a rýchlosťou 53
5.5 Aplikácia problému okrajových hodnôt 54
5.5.1 Výsledky riadenia 55
5.6 Aplikácia prediktívneho riadenia 57
5.6.1 Výsledky riadenia 58
5.7 Aplikácia genetického algoritmu 61
5.8 Zhodnotenie algoritmov 62
6. Záver 64
Literatúra 65
Elektronické prílohy 67

Zoznam obrázkov

Obr. 2. 1 Schéma lineárneho žeriava 11
Obr. 3. 1 Genetický algoritmus, reprezentácia populácie v binárnom zápise [8] 14
Obr. 3. 2 Genetický algoritmus, bod kríženia [8] 16
Obr. 3. 3 Genetický algoritmus, kríženie génov [8] 16
Obr. 3. 4 Genetický algoritmus, mutácia [8] 17
Obr. 3. 5 MPC – Pohyblivý horizont [10] 22
Obr. 3. 6 MPC – Kriteriálna funkcia a kvalita [10] 22
Obr. 3. 7 MPC – Vplyv posledného členu účelovej funkcie na jej hodnotu a argument [10] . 27
Obr. 4. 1 Nelineárny model lineárneho žeriava v Simulinku 30
Obr. 4. 2 Lineárny model lineárneho žeriava v Simulinku 31
Obr. 4. 3 Porovnanie odozvy lineárneho a nelineárneho matematického modelu 31
Obr. 4. 4 Fyzikálny model lineárneho žeriavu vytvorený v Simscape 32
Obr. 4. 5 Riešenie simulácie genetického algoritmu 36
Obr. 4. 6 Nestabilita riešenia pri zadaní nesprávnej kombinácie xf a tf 40
Obr. 4. 7 Stabilné riešenie algoritmu BVP 41
Obr. 4. 8 Riešenie simulácie MPC riadenia 45
Obr. 5. 1 Laboratórny model lineárneho žeriava 46
Obr. 5. 2 Schéma experimentálneho modelu 48
Obr. 5. 3 Vstup do systému pre odhad parametrov 51
Obr. 5. 4 Nameraná a simulovaná odozva systému s odhadnutými parametrami 52
Obr. 5. 5 Schéma PID regulátora [19] 53
Obr. 5. 6 Riadenie lineárneho žeriava rýchlosťou pomocou PID regulácie 54
Obr. 5. 7 Riešenie BVP pri voľbe krátkeho časového vektora 55
Obr. 5. 8 Výsledok riadenia anti-swing využitím BVP 56
Obr. 5. 9 MPC riadenie v Simulinku 58
Obr. 5. 10 Výsledok MCP riadenia lineárneho žeriava, $xf = 0.3m$, $tf = 3s$ 59
Obr. 5. 11 Výsledok MPC riadenia lineárneho žeriava, $xf = 0.7m$, $tf = 5s$ 60
Obr. 5. 12 Porovnanie riešení počítačových simulácií jednotlivých algoritmov 63

Zoznam tabuliek

Tab. 3. 1 Ukončovacie hlásenia programu quadprog a ich význam..... 29
Tab. 4. 1 Nastavenie parametrov systému pre simuláciu.....35
Tab. 4. 2 Pole premenných štruktúry riešenia `sol` 38
Tab. 5. 1 Parametre motora.....47
Tab. 5. 2 Odhadnuté parametre systému 52
Tab. 5. 3 Hodnoty zosilnení PID regulátora..... 54
Tab. 5. 4 Hodnoty penalizačných matíc MPC algoritmu 58
Tab. 5. 5 Porovnanie časovej náročnosti algoritmov 62

1. Úvod

V pokrokovej dobe akou je 21. storočie je najviac badateľný pokrok technológií viac ako čohokoľvek iného. Cieľom ľudí bolo odjakživa zjednodušovanie si každodenných často namáhavých činností. Nie je tomu inak ani v prepravnom priemysle, kde je samozrejmé, že ľudia potrebujú pomoc s pohybom ťažkých predmetov z miesta na miesto. Takým miestom môže byť napríklad sklad, či lodný prístav, kde sa v dnešnej dobe najviac využívajú mostové žeriavy. Môžu byť jedno alebo dvojnosníkové (štvornosníkové) vo vyhotovení s kladkostrojom alebo žeriavovým vozíkom – mačkou. Mostové žeriavy zabezpečujú spoľahlivú manipuláciu s bremenami až do 120 ton. Jednonosníkové zabezpečujú bezpečný a optimálny presun materiálov vo výrobných halách prípadne skladoch, kde nemá žeriavový systém dostatok miesta. Dvojnosníkové mostové žeriavy umožňujú napríklad jednoduchú realizáciu vyšších rýchlostí žeriavu, prípadne doplnenie ďalšieho pomocného zdvihu. Pohybové schopnosti žeriavov umožňujú pohybovať bremenom podľa konštrukcie žeriavov po priamke, v rovine aj v priestore. Disponujú veľkým rozpätím, zabezpečujú akúkoľvek prepravu ťažkého a veľkého materiálu. Síce mostové žeriavy sú veľmi dobrým pomocníkom, narábanie s nimi nie je až tak jednoduché ako by sa mohlo zdať. Jedným z najväčších problémov je nežiaduce kývanie predmetov zavesených na lane. Existujú viaceré technológie potlačujúce toto správanie [18].

Cieľom tejto práce je popis možností plánovania trajektórie jednoduchých mostových lineárnych žeriavov. Plánovanie trajektórie prebieha v samotnom programe pred začatím pohybu, alebo pri sofistikovanejších programoch možno plánovať dokonca aj počas pohybu. Rýchly rozvoj výpočtovej techniky umožňuje riešiť aj také úlohy, ktoré boli kedysi takmer neriešiteľné. Pre klasické metódy počítania (najmä maticové) je dobrým pomocníkom MATLAB, ktorý je obľúbený a populárny pre svoju jednoduchosť. V práci budú popísané tri algoritmy a to algoritmus problému okrajových podmienok (BVP), genetický algoritmus (GA) a algoritmus prediktívneho riadenia (MPC). Pomocou týchto algoritmov je možné riešiť plánovanie trajektórie lineárnych žeriavov. Okrem popisu základnej funkcionality algoritmov bude vytvorený matematický model lineárneho žeriava v MATLAB Simulinku. Na tomto modely budú odsimulované a otestované jednotlivé algoritmy. Po potvrdení ich funkčnosti autor prejde na aplikáciu algoritmov na reálny model lineárneho žeriava, ktorý sa nachádza v mechatronickom laboratóriu *Mechlab*. Pri reálnom experimente sa počíta, že neznáme parametre systému budú musieť byť identifikované, bude aplikovaný vhodný model trenia a tiež aj niektoré časti algoritmov budú pozmenené pre reálny model. V samom závere práce budú výsledky z reálnych simulácií porovnané a zhodnotí sa, ktorý algoritmus je najvhodnejší pre riadenie daného systému. Predpokladá sa, že najzložitejší algoritmus (*Prediktívne riadenie*) bude zároveň aj ten s najlepšími výsledkami.

2. Formulácia problému

2.1 Mostový žeriav

Na úvod je potrebné si predstaviť systém mostového žeriava, ktorého riadenie bude hlavným predmetom tejto práce. Mostový žeriav je zariadenie, ktoré sa využíva na premiestňovanie zväčša ťažkých nákladov, či bremien. Ich rozmery a výkonnosť môže dosahovať rôznych hodnôt, od malých zdvihákov, ktorých účelom môže byť dvíhať motory z automobilov až po mohutné stavby, ktoré dokážu dvihnúť jedny z najťažších nákladov na svete, ktoré sa využívajú hlavne vo veľkých lodných prístavoch [1].

Vo svete existuje veľa druhov mostových žeriavov čo sa týka nie len rozmerov, ale aj ich pohybových vlastností. Najrozšírenejším typom je plný mostový žeriav. Pod týmto pojmom sa rozumie systém, kde okrem pojazdného vozíku, z ktorého visí lano, na ktoré sa zavesí náklad, pozostáva aj z dvoch pojazdných vzpier, ktoré sú uložené na koľaji a teda ich pohyb je orientovaný v jednej línii, zväčša kolmej na orientáciu pohybu vozíka. Ostatné systémy sú len istou variáciou plného mostového žeriava [1]. V tejto práci sa bude autor zaoberať systémom, kde celý žeriav je nehybný okrem pojazdného vozíka, na ktorom bude zavesené závažie, teda jedná sa o systém s dvomi stupňami voľnosti [1].

2.2 Anti-swing

V priemysle existuje veľa príkladov, kde pri narábaní s materiálmi či tovarom je pohyb veľmi obmedzený napr. priestormi v sklade a žeriavnik má v takýchto situáciách problém s manévrovaním, lebo prirodzene pri náhlych rozjazdoch a brzdeniach dochádza ku kývaniu nákladu. Problémov pri takomto nežiaducom oscilačnom pohybe môže byť viacero. Nie len zdržanie prevádzky, kým sa čaká na ustálenie oscilačného pohybu nákladu, ale aj bezpečnosť nákladu a najmä pracujúceho personálu je takto ohrozená [2].

Kývanie je veľkým problémom aj pre efektívnosť prevádzky mostových žeriavov. Riešením zo začiatku boli rôzne mechanické vylepšenia žeriavov, ako napríklad špeciálna zdvihákova kotva. V dnešnej dobe sa používa anti-swing riadenie takýchto systémov, ktoré zaručuje po dojazde na miesto určenia statickú polohu nákladu bez kývania. Anti-swing riadenie má za cieľ predchádzaniu vzniku kmitov, ale nerieši prípad, keď takéto kmity vzniknú napr. elimináciu kmitov spôsobených vetrom, prípadne kolíziou s prekážkou.

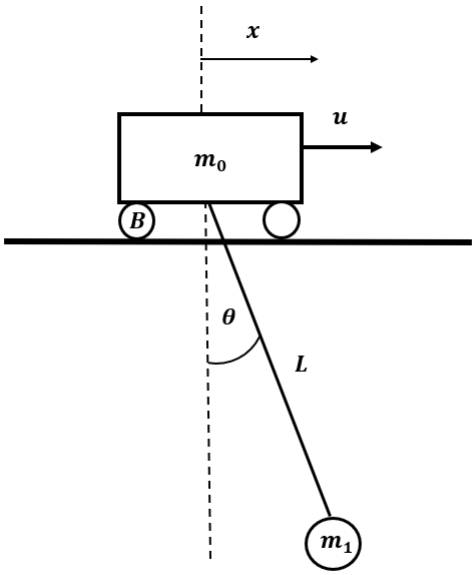
2.3 Popis dynamiky systému

Problematika lineárneho žeriavu sa dá označiť ako SIMO (single input, multiple output) systém, kde jedným vstupom, teda silou pôsobiacou na vozík riadime polohu vozíka, ale taktiež aj výchylku závažia na lane. Systém má dva stupne voľnosti a jeden aktuátor, teda sústava je kinematicky neurčitá. Popisovaný systém musí splňovať viaceré podmienky, ktoré treba zohľadniť pri riadení. Ideálne by sa závažie a teda výchylka lana mala pohybovať v okolí lineárnej polohy, čo je pri polohe lana zvislo dolu. Keďže ide o model, kde je závažie zavesené na lane, tak nesmie dochádzať k veľkým uhlovým rýchlostiam a zrýchleniam

v rotačnej väzbe uchytenia lana na vozík, pretože je potrebné zabezpečiť neustále napätie lana počas celého trvania pohybu, inak môže dôjsť k chaotickému pohybu závažia.

2.4 Pohybové rovnice

Pohybová rovnica vyjadruje vzťah medzi hnacím členom a pracovným mechanizmom. Je matematickým vyjadrením vzťahov medzi momentmi a silami pôsobiacimi na mechanický subsystém pohonu a kinematickými veličinami určujúcimi vzniknutý pohyb. Pomocou schémy lineárneho žeriava na obrázku 2.1. budú odvodené pohybové rovnice systému. Závažie zavesené na lane je považované za hmotný bod, tuhosť a hmotnosť lana sú zanedbané. Dĺžka lana sa v priemysle zachováva konštantná z bezpečnostných dôvodov.



Obr. 2. 1 Schéma lineárneho žeriava

Pohybové rovnice sú odvodené pomocou Lagrangeových rovníc. Energie systému sú vyjadrené takto [2] [3].

$$E_k = \frac{1}{2}m_0\dot{x}^2 + \frac{1}{2}m_1\left(\dot{x} + L\dot{\theta}\cos(\theta)\right)^2 \tag{2.1}$$

$$E_p = m_1 g(L - Lcos\theta) \tag{2.2}$$

Lagrangián má tvar [3]

$$L = E_k - E_p = \frac{1}{2}m_0\dot{x}^2 + \frac{1}{2}m_1\left(\dot{x} + L\dot{\theta}\cos(\theta)\right)^2 - m_1 g(L - Lcos\theta) \tag{2.3}$$

Derivovaním Lagrangiánu podľa premenných $x, \dot{x}, \theta$ a $\dot{\theta}$ podľa rovnice (2.4) a (2.5) vznikne systém rovníc popisujúci dynamiku lineárneho žeriava (2.6) a (2.7) [6].

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = u \tag{2.4}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0 \tag{2.5}$$

kde *u* je akčný člen, teda sila pôsobiaca na vozík.

$$(m_0 + m_1)\ddot{x} + m_1 L\ddot{\theta}\cos\theta - m_1 L\dot{\theta}^2 sin\theta = u \tag{2.6}$$

$$m_1 L^2\ddot{\theta} + m_1 L\ddot{x}cos\theta + m_1 gLsin\theta = 0 \tag{2.7}$$

Po pár matematických operáciách možno z rovníc (2.6) a (2.7) vytvoriť nelineárny stavový model systému (2.8), kde $X_1 = x, X_2 = \dot{x}, X_3 = \theta$ a $X_4 = \dot{\theta}$ [6].

$$\frac{d}{dt}\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} = \begin{pmatrix} X_2 \\ \dfrac{u + m_1 L{X_4}^2 \sin(X_3) + g m_1 \sin(X_3)\cos(X_3)}{m_0 + m_1(1 - cos^2(X_3))} \\ X_4 \\ \dfrac{ucos(X_3) + m_1 L{X_4}^2 \sin(X_3)\cos(X_3) + g(m_0 + m_1)\sin(X_3)}{m_1 Lcos^2(X_3) - (m_0 + m_1)L} \end{pmatrix} \tag{2.8}$$

Medzi hlavné výhody stavovej reprezentácie systému proti tradičným opisom patrí najmä jednoduchosť na implementáciu, transformácia diferenciálnych rovníc *n*-tého rádu na *n* diferenciálnych rovníc prvého rádu, prístup k stavom, zohľadnenie počiatočných podmienok, riešiteľnosť priamo v časovej oblasti a je použiteľná aj na zložité nelineárne systémy s viacerými vstupmi a výstupmi s časovou premenlivosťou.

2.5. Linearizácia

V okolí pracovného bodu je možné nájsť lineárnu aproximáciu nelineárneho systému. Výhodou lineárneho systému je jednoduchšie riadenie viacerými spôsobmi. Linearizácií môže byť viacero, napr. linearizácia okolo jedného pracovného bodu, okolo viacerých pracovných bodov alebo spätnoväzobná linearizácia. V prípade mostového žeriava bude využitá linearizácia okolo jedného pracovného bodu a to okolo stabilnej rovnovážnej polohe závažia visiaceho zvislo dolu [2] [3].

Lineárnou oblasťou možno rozumieť oblasť, kde podľa *Obr. 2.1* uhol θ nadobúda hodnôt blízkych nule. V praxi do lineárnej oblasti spadajú uhly zväčša v rozmedzí $\theta = \pm 5°$. V tom prípade možno zaviesť aproximáciu $cos\theta = 1$, $sin\theta = \theta$. Nelineárny stavový model systému (2.8) sa dá potom prepísať na

$$\frac{d}{dt}\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix} = \begin{pmatrix} X_2 \\ \dfrac{u + m_1 L {X_4}^2 X_3 + g m_1 X_3}{m_0} \\ X_4 \\ \dfrac{u + m_1 L {X_4}^2 X_3 + g(m_0 + m_1) X_3}{m_1 L - (m_0 + m_1) L} \end{pmatrix} \tag{2.9}$$

Pri porovnaní rovníc (2.8) a (2.9) je ešte možné vyvodiť ďalšiu výhodu lineárnych modelov, a to rýchlosť výpočtu. Síce pri moderných počítačoch rozdiel rýchlosti výpočtu riadenia týchto dvoch modelov možno nebude až tak razantný, ale ak by šlo o riadenie v reálnom čase, tak pri vyšších vzorkovacích frekvenciách by už dĺžka výpočtu mohla hrať dôležitú rolu.

3. Metódy riadenia

3.1 Genetický algoritmus

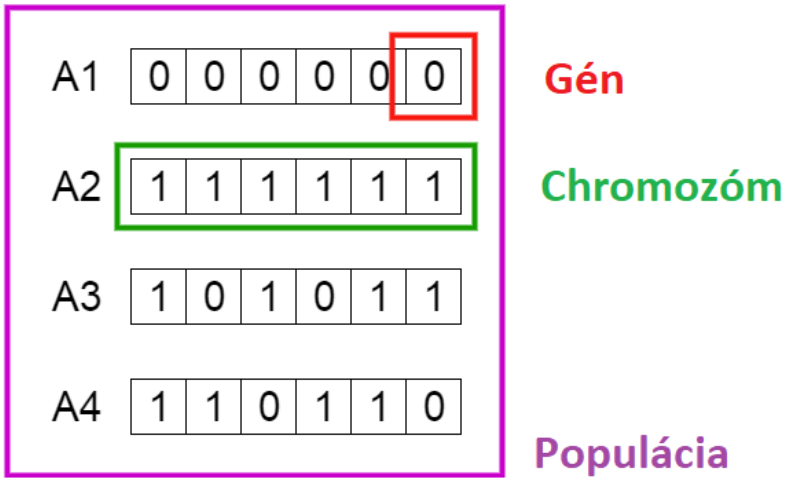
V optimalizačných úlohách sa veľmi často využíva evolučných algoritmov inšpirovaných Darwinovou teóriou prírodnej evolúcie. Tento algoritmus odráža proces prirodzenej selekcie, kde podľa hodnotiacej črty najlepší jedinci sú vybraní k reprodukcii pre vytvorenie novej lepšej generácie [9]. Proces selekcie spočíva vo vybraní najlepších jedincov, ktorých charakteristiky sa predajú do novej generácie potomkov. V ideálnom prípade sa prenesú na potomkov dobré charakteristiky a vzniknú tak lepší jedinci. Tento proces sa opakuje kým nie sú nájdení najlepší jedinci populácie [9].

Evolučné algoritmy sa využívajú najmä na problém hľadania. Uvažuje sa súbor riešení problému a vyberú sa z nich tie najlepšie. Genetický algoritmus sa skladá z piatich základných častí [8]:

1. Inicializačná generácia
2. Hodnotiaca funkcia
3. Selekcia
4. Reprodukcia
5. Mutácia

3.1.1 Inicializačná generácia

Celý proces je sústredený na jednotlivcov, ktorí tvoria populáciu. Každý jedinec je riešením problému, ktorý je charakterizovaný súborom parametrov – génov, ktoré sú spojené do reťazca – chromozóm. Na reprezentáciu génov sa často využíva binárneho zápisu čísiel. Potom reprezentácia populácie môže byť nasledovná [8]:



Obr. 3. 1 Genetický algoritmus, reprezentácia populácie v binárnom zápise [8]

Inicializačná populácia sa často vytvára náhodne alebo s prihliadnutím na problém je možné náhodné generovanie trochu potlačiť. Pri zložitejších úlohách záleží aj na inicializačnej populácii, pretože je možné, že celé riešenie skonverguje do lokálneho extrému a v ďalších fázach algoritmu sa z neho už nedostane.

3.1.2 Hodnotiaca funkcia

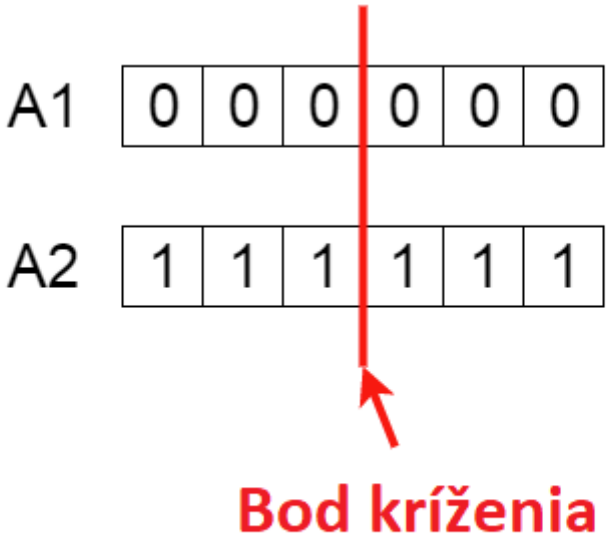
Hodnotiť môžeme jedincov alebo aj populáciu. Hodnotenie prebieha pomocou hodnoty jednej fitness funkcie u obyčajného algoritmu, avšak u miltikriteriánych úloh, keď je fitness funkcií väčší počet sa používajú rôzne sofistikované techniky hodnotenia. Hodnotiaca funkcia udáva kvalitu riešenia. Určuje ako veľmi je daný jedinec vhodný súperiť s ostatnými jedincami o partnera s najlepšími génmi. Každému jedincovi je teda pridelené skóre, podľa ktorého sú následne jedinci zoradení od najlepšieho po najhoršieho [8] [9].

3.1.3 Selekcia

Myšlienkou selekcie je vybrať najlepších jedincov v závislosti na hodnotiacej funkcii a nechať ich predať svoje gény ďalšej generácii. Vyberú sa dvaja jedinci – rodičia, v závislosti na ich hodnotení. Jedinci s vyšším hodnotením majú vyššiu pravdepodobnosť dostať sa do ďalšej fázy algoritmu. Väčšinou platí, že k reprodukcii sa dostane iba určité percento najlepších jedincov z populácie, ktorí sú popárovaní na základe ich hodnotiacej funkcie. Tvorba párov môže mať rôzne podoby, ale jeden zo základných spôsobov je postupne párovať jedincov od najlepších smerom k horším [8].

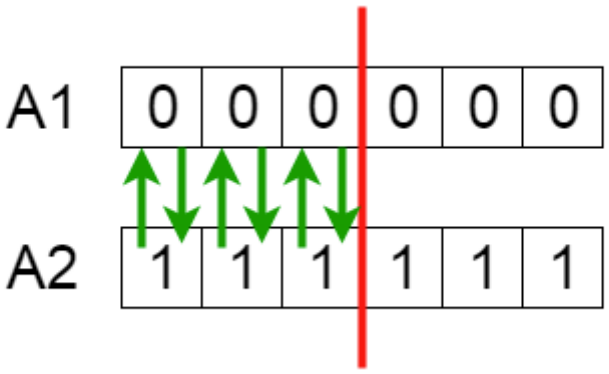
3.1.4 Reprodukcia

Najdôležitejšou časťou celého algoritmu je práve reprodukcia. Tento proces má jedincom zabezpečiť získanie vlastností, ktoré mu práve môžu chýbať k lepšej hodnote v procese hodnotenia. Z páru vybraných rodičov sa náhodne vyberie bod kríženia génov. Je to pomyselný bod, ktorý rozdelí gény rodičov na dve alebo viac častí a tie sa medzi sebou vymenia a tak vzniknú dvaja noví potomkovia. Tento proces v binárnej podobe je zobrazený na nasledovných obrázkoch [8].



Obr. 3. 2 Genetický algoritmus, bod kríženia [8]

Následne z týchto rodičov a takto usporiadaných génov vzniknú potomkovia.



Obr. 3. 3 Genetický algoritmus, kríženie génov [8]

Títo potomkovia sú ďalej pridaní do populácie, pričom ich rodičia v populácii zostávajú. Pri pokročilejších algoritmoch je možné vytvoriť aj isté počítanie veku, čo znamená, že jedinci, ktorí sú v populácii nejaký počet iterácií, tak sa dostanú do fázy úmrtia, teda sú z populácie vymazaní. Takáto funkcia môže algoritmu pomôcť prekonať lokálne extrémy [9].

3.1.5 Mutácia

U novo vzniknutých jedincov sa s istou pravdepodobnosťou, ktorá je označovaná ako *mutation rate* (miera mutácie), môže objaviť mutácia. Miera mutácie vyjadruje aké percento génov z celého chromozómu zmení svoju hodnotu. Mutácia slúži na udržanie rozdielnosti jedincov, pomáha získavaniu nových vlastností a tým bráni predčasnej konvergencii k nevhodnému riešeniu. V binárnom zápise to znamená, že niektoré bity chromozómu môžu byť prevrátené [8].

Obr. 3. 4 Genetický algoritmus, mutácia [8]

3.2 Problém okrajových hodnôt

Metóda problému okrajových podmienok pre danú diferenciálnu rovnicu spočíva v nájdení riešenia diferenciálnej rovnice pre daný súbor okrajových podmienok. „*Okrajová podmienka je predpis kombinácii hodnôt neznámeho riešenia a jeho derivácií vo viac ako jednom bode*“ [17]. Správanie problému okrajových hodnôt nie je celkom ako problém počiatočných hodnôt. U problémov okrajových hodnôt často pre dané okrajové podmienky riešenie buď neexistuje alebo ak existuje, tak ich môže byť viac, väčšinou nekonečne mnoho.

3.2.1 MATLAB a problém okrajových hodnôt – bvp4c

Keďže riešenie systémov nelineárnych rovníc je celkom náročná matematická úloha, tak MATLAB poskytuje vhodnú zabudovanú funkciu *bvp4c*, pomocou ktorej sa dajú riešiť značne sofistikované problémy. Rieši systém diferenciálnych rovníc prvého rádu v tvare [12]

$$y' = f(x, y, p), \qquad a \leq x \leq b \tag{3.1}$$

vzhľadom k okrajovým podmienkam [12]

$$g(y(a), y(b), p) = 0 \tag{3.2}$$

kde *p* označuje vektor voľných parametrov algoritmu.

Konkrétne, bvp4c je konečne diferencovateľný program, ktorý implementuje trojstupňovú Lobato IIIa rovnicu zahŕňajúcu Runge-Kutta metódu. Uvažuje sa, že funkcie $f(x, y, p)$ a $g(u, v, p)$ sú dostatočne hladké, spojité a splňujú Lipschitzovu podmienku v ose y. Podľa [13] ide o spájanie po častiach kubickej polynomickej funkcie $S(x)$. $S(x)$ spĺňa okrajové podmienky a každý subinterval $[x_i, x_{i+1}]$ siete $a = x_0 < x_1 < \ldots < x_N = b$ je preložený kubickým polynómom cez dané body na okrajoch subintervalu a v jeho strede. Funkcia je spojitá na okrajoch subintervalov a spĺňa podmienku $S(x) \in C^1[a, b]$. Ak sa aplikuje predchádzajúce na kvadratúrny problém, zredukuje sa riešenie na Simpsonovu rovnicu. Pre $y_i = S(x_i) \approx y(x_i)$ a $h_i = x_{i+1} - x_i$ platí vzťah

$$\begin{aligned} y_{i+1} &= y_i + \frac{h_i}{6}\big(f(x_i, y_i) + f(x_{i+1}, y_{i+1})\big) \\ &+ \frac{2h_i}{3} f\left(x_i + \frac{h_i}{2}, \frac{y_i + y_{i+1}}{2} - \frac{h_i}{8}\big(f(x_{i+1}, y_{i+1}) - f(x_i, y_i)\big)\right) \end{aligned} \tag{3.3}$$

Táto rovnica sa používa pre viaceré programy pre riešenie problémov okrajových hodnôt. Častou otázkou pri riešení BVP je definícia siete, na ktorej sa problém rieši. MATLAB túto úlohu rieši riadením zvyškov, lebo Simpsonova rovnica je na toto riešenie prispôsobená [13].

Zvyškom funkcie $S(x)$ v diferenciálnych rovniciach je $r(x) = S'(x) - f\big(x, S(x)\big)$ a zvyšok v okrajových podmienkach je $g\big(S(a), S(b)\big)$. Inak povedané, $S(x)$ je riešením BVP

$$\begin{aligned} y' &= f(x,y) + r(x) \\ g\big(y(a), y(b)\big) &= g\big(S(a), S(b)\big) \end{aligned} \tag{3.4}$$

Z pohľadu spätnej analýzy chýb, $S(x)$ je vhodným riešením iba ak sú zvyšky malé. Zvyšok z odvodenia Lipschitzovej podmienky pre Simpsonovu rovnicu má tvar [13]

$$r(x) = S'(x) - y'(x) + f(x_i, y_i) - f\big(x_i, y(x_i)\big) \tag{3.5}$$

Z tejto rovnice vidno, že zvyšok je možné vypočítať pre akékoľvek x a je možné vytvoriť vierohodný odhad zvyšku aj keď je sieť príliš hrubá. Preto je algoritmus do určitej miery odolný aj proti nevhodne zvolenej sieti a nevhodne vytvorenom odhade riešenia užívateľom a nájde riešenie, keď je evidentné asymptotické chovanie [13].

Simpsonova metóda aplikovaná na rovnice (3.1) a (3.2) s použitím siete $a = x_0 < x_1 < \ldots < x_N = b$ je vyhodnotená riešením algebraických rovníc

$$\Phi(X, Y) = 0 \tag{3.6}$$

kde

$$X = [x_0, x_1, \ldots, x_N]^T$$

$$Y = [y_0, y_1, \ldots, y_N]^T$$

$$\Phi_0(X, Y) = g(y_0, y_N, p)$$

$$\Phi_{\mathrm{i}}(X, Y) = \mathrm{y_i} - y_{i-1} - \frac{1}{6} h_{i-1} \left(f_{i-1} + 4 f_{i-\frac{1}{2}} + f_i \right)$$

pre $i = 1,2,\ldots,N$, a

$$f_i = f(x_i, y_i, p)$$

$$f_{i-\frac{1}{2}} = f\left(x_{i-1} + \frac{h_{i-1}}{2}, \frac{y_{i-1} + y_i}{2} - \frac{h_{i-1}}{8} (f_i - f_{i-1}), p \right)$$

Rovnica (3.6) sa rieši ďalej zjednodušenou Newtonovou metódou, teda je potrebný Jakobián $\frac{\partial \Phi}{\partial Y}$. Pri jemne zvolenej sieti a v dôsledku počítania veľkého systému lineárnych rovníc v každej iterácii vzniká kľúčová úloha počítania Jakobiánu vzhľadom na efektivitu a pamäť. Ak v algoritme nevystupujú žiadne neznáme parametre a sú oddelené okrajové

podmienky, tak Jakobiho matica má tvar schodovej matice, pre ktoré existuje viacero efektívnych spôsobov ako s nimi počítať a ako ich ukladať. To je dôvod, prečo mnohé známe programy na riešenie BVP vylučujú z algoritmov neznáme parametre a neseparované okrajové podmienky, čo má za následok to, že celé toto bremeno je presunuté na užívateľa tieto kroky pre program pripraviť. Výhodou pre MATLAB v tomto prípade je, že obsahuje technológiu pre narábanie s riedkymi maticami a teda Jakobiho maticu nie je potrebné počítať ako schodovú. Celý algoritmus počítania Jakobiánu je dostatočne robustný. Obsahuje informáciu o váhe odhadu jedného Jakobiánu, ktorá slúži ako pomoc pri určení prírastkov pre nasledujúci. Pri počítaní jedného stĺpca Jakobiánu, zaznamenáva zmenu funkčnej hodnoty a prepočíta stĺpec s inou hodnotou prírastku ak to je potrebné pre vytvorenie lepšieho odhadu. Funkcie *bvp4c* testuje či platí [13]

$$\| J_i - J_{i-1} \|_1 \leq 0{,}25 \cdot (\| J_i \|_1 + \| J_{i-1} \|_1) \tag{3.7}$$

Ak áno, Jakobián v strede subintervalu $J_{i-\frac{1}{2}}$ je aproximovaný priemerom hodnôt J_i a J_{i-1}, inak musí byť vypočítaný priamo. Takáto aproximácia má za následok, že celý algoritmus pri dlhších simuláciách dokáže znížiť čas simulácie až o 20% [13].

Všeobecný prístup tvorby siete prerozdeľuje body siete globálnou stratégiou založenou na inverznej interpolácii. Tá však predpokladá, že funkcia *f* je hladká a sieť jemná. Takáto stratégia je neprijateľná práve ak je sieť hrubá a funkcia *f* iba po častiach hladká. Preto vývojári programu MATLAB vytvorili lokálnu stratégiu prerozdeľovania bodov siete. Ak je norma zvyšku väčšia než tolerancia, tak do subintervalu sú pridané rovnomerne rozložené body siete, nikdy nie však viac ako dva. Ak je norma zvyšku väčšia než stonásobok tolerancie, pridané sú dva body, naopak iba jeden. Je možné počet bodov aj redukovať, ak je evidentné asymptotické chovanie zvyšku. Jednou zo stratégií použitých v *bvp4c* je nahradenie troch po sebe idúcich subintervalov za dva [13].

3.3 Prediktívne riadenie

„*Základná myšlienka prediktívneho riadenia je prepočítavanie najlepšieho možného – optimálneho riadiaceho zákona počas riadeni*a.“ [10, s. 38] Teda princípom prediktívneho riadenia na základe modelu (MPC) je v každej vzorkovacej perióde vypočítať optimálny akčný zásah na základe situácie, t.j. informácií o stavoch systému. Oproti napríklad LQ riadeniu má pri aplikácii procesných obmedzení lepší výkon, pričom LQ riadenie je oveľa pokročilejšie ako klasické integračno-derivačné (PID) riadenie. Hoci prediktívne riadenie možno aplikovať na systémy, ktoré klasickými metódami iba ťažko riadiť, jeho zložitosť je vyššia a praktická implementácia je taktiež obťažnejšia [14].

Pre využitie prediktívneho riadenia na základe modelu je samozrejme potrebné vytvoriť model systému. Vhodným popisom systému pre využitie MPC bude stavový model. Obecný zápis stavového modelu je uvedený v rovniciach (3.8) a (3.9) [14].

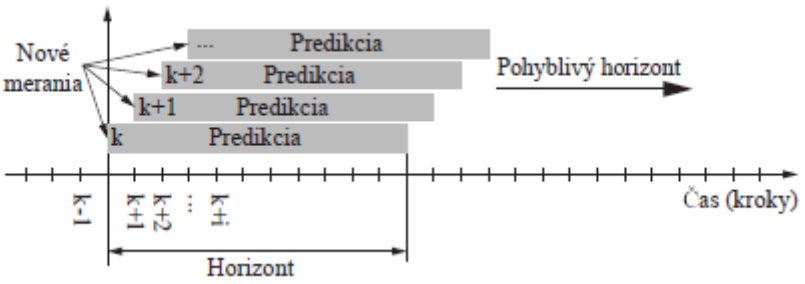
$$\frac{dx(t)}{dt} = Ax(t) + Bu(t) \tag{3.8}$$

$$y(t) = Cx(t) + Du(t) \tag{3.9}$$

Časová premenná $x(t)$ je stĺpcový vektor popisujúci stavy systému, $u(t)$ je stĺpcový vektor vstupov, matica A popisuje vnútorné väzby systému (matica dynamiky), B popisuje väzby systému na vstup (matica riadenia), matica C je väzba výstupu na stav a matica D je väzba vstupu na výstup. Tento model je možné vytvoriť vyčíslením Jakobiánu vytvoreného zo systému diferenciálnych rovníc prvého rádu popisujúcich dynamiku systému (2.8) v lineárnom bode [20].

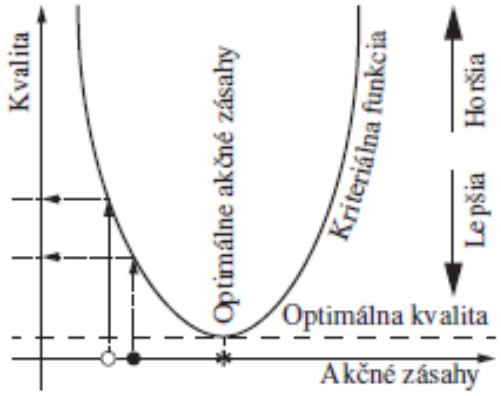
Samotný proces predikcie možno popísať ako simuláciu v každej vzorkovacej perióde, pričom jej účelom je zistiť ako sa systém bude správať v budúcnosti, ak sa nachádza v aktuálnom stave a do systému vstupujú aktuálne a následne predikované vstupy na základe známej stratégie. Nutné je poznamenať, že predikcia predpovedá správanie systému v každom kroku iba do určitého okamihu, teda na istom horizonte, ktorý nesmie byť nekonečne dlhý ako tomu je pri LQ riadení. Z tohto vyplýva, že čas je nutné previesť do diskrétneho tvaru. Spojitý čas sa rozdelí na rovnomerne rozdelené úseky trvajúce dĺžky Ts. Simulácia predikcie je vykonaná v diskrétnych okamihoch $t = kTs$, pričom k je celé číslo označujúce koľko vzorkovacích periód do budúcnosti predikujeme. Číslo k nazývame horizontom predikcie [10].

Ďalšou dôležitou vlastnosťou horizontu predikcie je to, že nie je statický. Po vykonaní predikcie v časovej vzorke k sa posúva do nasledujúcej časovej vzorky $k+1$. Takýto horizont sa nazýva pohyblivý horizont. Správanie pohyblivého horizontu je zobrazené na Obr. 3.1 [10].



Obr. 3. 5 MPC – Pohyblivý horizont [10]

Pri automatickom riadení je potrebné nielen aby stavy systému spĺňali predpísané hodnoty, ale tiež je dôležité aj zohľadniť vplyv akčných zásahov pre zohľadnenie energetickej náročnosti procesu dodržiavania predpísaných hodnôt stavov. Tieto informácie vyjadruje kriteriálna funkcia *J*, ktorá vyjadruje kvalitu riadenia, t.j. ako ďaleko sme od dosiahnutia definovaných cieľov riadenia. Ideálne je vyjadriť kriteriálnu funkciu tak, že nižšia hodnota indikuje kvalitnejšie riadenie ako vyššia hodnota, pretože celá úloha sa presmeruje na minimalizačnú úlohu kriteriálnej funkcie. Cennou vlastnosťou predikcie je, že je možné predikovať vplyv vstupov na dynamiku riadeného systému, preto je možné predikovať aj kriteriálnu funkciu a to nielen o krok dopredu ale až po koniec predikčného horizontu. Kriteriálna funkcia a hodnotenie kvality je zobrazené na Obr. 3.6 [10] [14].



Obr. 3. 6 MPC – Kriteriálna funkcia a kvalita [10]

Mnohé praktické aplikácie však nemôžu aplikovať akčný zásah z celého rozsahu do nekonečna, ale musia byť často obmedzené v istom intervale. Tieto obmedzenia, vzniknuté či z technologického alebo bezpečnostného hľadiska, musia byť dodržané. Jednoduchý príklad obmedzenia je to, keď napríklad v prípade žeriava je motor schopný vyvinúť moment len do istej hodnoty a túto hodnotu z technologického hľadiska nemôže presiahnuť. Bezpečnostné hľadisko môže súvisieť napríklad s rýchlosťou pohybu nákladu [10] [14].

3.3.1 Predikčné matice

Po prevedení rovnice (3.8) do diskrétnej časovej roviny možno získať rovnicu predstavujúcu jednokrokovú predikciu (3.10) [10].

$$\mathrm{x_{k+1}} = \mathrm{Ax_k} + \mathrm{Bu_k} \tag{3.10}$$

To znamená, že ak je známy vstup a aktuálny stav v čase *k*, je možné vypočítať z modelu stavy o jeden krok dopredu. Ak sa aplikuje tento zákon viackrát, je možné získať predikciu až po koniec horizontu. Táto predikcia je zobrazená v rovnici (3.11) [10].

$$\begin{aligned}
k \quad & x_k = x_k \\
k+1 \quad & x_k = Ax_k + Bu_k \\
k+2 \quad & x_k = Ax_{k+1} + Bu_{k+1} = A^2x_k + ABu_k + Bu_{k+1} \\
\vdots \quad & \vdots \\
k+i \quad & x_i = Ax_{k+i+1} + Bu_{k+i} \\
\vdots \quad & \vdots \\
k+n_p \quad & x_{n_p} = A^{n_p}x_k + A^{n_p-1}Bu_k + \ldots + ABu_{k+n_p} + Bu_{k+n_p-1}
\end{aligned} \tag{3.11}$$

Za predpokladu, že skúmaný systém je autonómny, teda nemá vstupy, je možné odstrániť vynútenú časť, $B = 0$. Potom možno odvodiť predikčnú maticu dynamiky *M*, ktorá predikuje chovanie systému na základe aktuálnych stavov, to znamená, že každý nasledujúci stav je závislý iba na aktuálnom stave, nie na akčných zásahoch [10].

$$\overrightarrow{x_k} = \begin{bmatrix} x_{k+1} \\ x_{k+2} \\ \vdots \\ x_{k+n\_p} \end{bmatrix} x_k = Mx_k \tag{3.12}$$

$$M = \begin{bmatrix} A^1 \\ A^2 \\ \vdots \\ A^{n_p} \end{bmatrix} \tag{3.13}$$

Keďže pri riadení systémov je nutné zohľadniť nielen predikciu dynamiky, ale aj predikciu odozvy na vstupy, tak rovnako ako pri odvodení predikčnej matice dynamiky M z rovnice (3.11) je možné odvodiť predikčnú maticu vynútenej odozvy N, ktorá má však už komplikovanejší tvar ako predikčná matica dynamiky [10].

$$N = \begin{bmatrix} B & 0 & 0 & \cdots & 0 & 0 \\ AB & B & 0 & \cdots & 0 & 0 \\ A^2B & AB & B & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A^{n_p-2}B & A^{n_p-3}B & A^{n_p-4}B & \cdots & B & 0 \\ A^{n_p-1}B & A^{n_p-2}B & A^{n_p-3}B & \cdots & AB & B \end{bmatrix} \tag{3.14}$$

3.3.2 Účelová funkcia

Ako už bolo spomenuté, hodnota účelovej (kriteriálnej) funkcie znázorňuje kvalitu riadenia. Hoci v algoritme MPC vystupujú takmer výlučne matice a maticové vzťahy, výstupom účelovej funkcie je skalár. Konkrétny tvar účelovej funkcie nie je stanovený, záleží to od optimalizovaného problému. Hlavnou myšlienkou MPC riadenia je optimalizačnou úlohou, t.j. hľadanie extrémov funkcie, konkrétne hľadanie minima. Pri označení optimálnej hodnoty argumentu (u^*) má optimalizačná úloha nasledujúci tvar [10].

$$J(u^*) = \arg\min J(u) \tag{3.15}$$

Argument značí, že je hľadaná optimalizačná premenná funkcie pri minime, $J(u^*)$ označuje hodnotu účelovej funkcie pri hodnote premennej u^*. Keďže stavy sú jednoznačne dané stavovým modelom, tak optimalizačná úloha je závislá iba na akčných zásahoch, pričom účelová funkcia je závislá ako na akčných zásahoch, tak aj na stavoch [10].

$$J(u^*) = \min_u J(u, x) \tag{3.16}$$

Minimum funkcie znamená nájsť bod, v ktorom funkcia ani nestúpa ani neklesá. To znamená, že v minime funkcie je derivácia nulová [10].

$$\frac{\mathrm{d}J}{\mathrm{d}u} = 0 \tag{3.17}$$

Ak by do systému vstupovalo viacero vstupov, tak optimalizačná úloha nadobudne tvar (3.18) [10].

$$\nabla J(u) = \left(\frac{\partial J}{\partial u_1}, \frac{\partial J}{\partial u_2}, \ldots, \frac{\partial J}{\partial u_N}\right) = 0 \tag{3.18}$$

Keďže ide o prediktívne riadenie, je možné predikovať aj účelovú funkciu. Ak j_k označuje kvalitu riadenia v jednom kroku a j_{k+i} označuje kvalitu riadenia v kroku $k+i$ pre $i = 1,2,\ldots,n_p-1$, potom v rámci trvania horizontu možno vyjadriť účelovú funkciu J_k, ktorá už nesie informáciu aj o kvalite riadenia v budúcnosti [10].

$$\mathrm{J_k} = \sum_{i=0}^{n_p-1} j_{k+i} \tag{3.19}$$

3.3.3 Penalizačné matice

Aj u prediktívneho riadenia je možné kontrolovať kvalitu riadenia odchýlkou ako je tomu napríklad u PID riadenia. Odchýlka je daná rozdielom požadovanej od aktuálnej hodnoty, čo je možné položiť rovné kvalite riadenia [10].

$$j_k = e_k = r - y_k \tag{3.20}$$

Pri odvodzovaní nasledovných vzťahov je referenčná hodnota $r = 0$. Pretože jednoduchý lineárny žeriav má iba jeden vstup a je žiaduce, aby účelová funkcia zohľadňovala kladnú aj zápornú odchýlku rovnako, tak je výhodnejšie počítať s kvadrátom výstupov [10].

$$j_k = e_k^2 = y_k^2 \tag{3.21}$$

Keďže výstupy sú vyjadrené maticou, tak tvar (3.21) je možné prepísať do tvaru nasledujúceho [10].

$$j_k = y_k^T y_k = x_k^T C^T C x_k = x_k^T Q x_k \tag{3.22}$$

kde matica *Q* je váhovacia (penalizačná) matica stavu. Táto matica musí byť symetrická, kladne semidefinitná, teda väčšia alebo rovná nule. Analogicky pri zavedení vstupov do účelovej funkcie je možné odvodiť druhú penalizačnú maticu *R* – penalizačná matica vstupov. Táto matica musí byť štvorcová podľa počtu vstupov a musí byť kladne definitná, teda väčšia ako nula a to preto, lebo pri zadaní matice *R* nula, akčné zásahy budú násobené nulou, teda do systému nebudú vstupovať žiadne akčné zásahy a riadenie nemá zmysel [10].

$$j_k = x_k^T Q x_k + u_k^T R u_k \tag{3.23}$$

V rovnici (3.20) bola účelová hodnota počítaná iba do kroku $n_p - 1$ a to preto, že koncový stav je váhovaný samostatne. Potom prepísaním tejto rovnice s váhovacími maticami vznikne vzťah (3.24) [10].

$$J_k = \sum_{i=0}^{n_p-1} (x_{k+i}^T \boldsymbol{Q} x_{k+i} + u_{k+i}^T \boldsymbol{R} u_{k+i}) + x_{k+n_p}^T \boldsymbol{P} x_{k+n_p} \tag{3.24}$$

kde matica *P* je váhovacou (penalizačnou) maticou koncového stavu [10].

3.3.4 Kompaktná účelová funkcia

Po odvodení účelovej funkcie do kompaktného stavu vznikne finálny zápis účelovej funkcie na celom horizonte [10].

$$J_k = \vec{u}_k^T \mathbf{H} \vec{u}_k + 2x_k^T \mathbf{G}^T \vec{u}_k^T + x_k^T \mathbf{F} x_k \tag{3.25}$$

kde $\overrightarrow{u_k}$ značí celú postupnosť vstupov do budúcnosti. Pre Hessián $\boldsymbol{H}$ platí

$$H = \sum_{i=0}^{n_p-1} N_i^T Q N_i + N_{n_p}^T P N_{n_p} + diag(R) \tag{3.26}$$

ďalší člen $\boldsymbol{G^T}$ má tvar

$$G^T = \sum_{i=0}^{n_p-1} M_i^T Q N_i + M_{n_p}^T P N_{n_p} \tag{3.27}$$

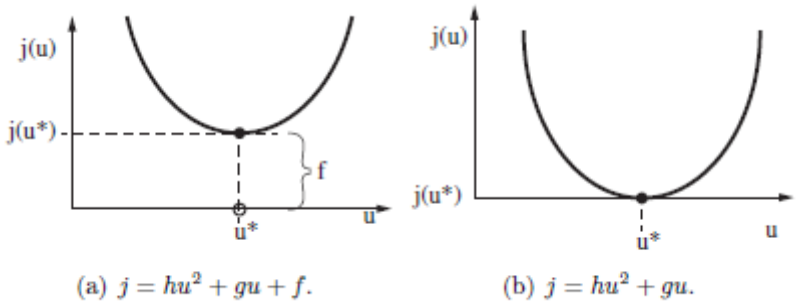
Pre posledný člen účelovej funkcie $\boldsymbol{F}$ má tvar

$$F = \sum_{i=0}^{n_p-1} M_i^T Q M_i + M_{n_p}^T P M_{n_p} \tag{3.28}$$

Potom optimalizačná úloha dostáva tvar

$$\vec{u}_k^* = \arg\min(\vec{u}_k^T \mathrm{H} \vec{u}_k + 2x_k^T \mathrm{G}^T \vec{u}_k^T + x_k^T \mathrm{F} x_k) \tag{3.29}$$

Pri optimalizačnej úlohe MPC je dôležité nájsť, kde sa minimum nachádza ale samotná funkčná hodnota účelovej hodnoty nie je dôležitá. Z Obr. 3.7 je viditeľné, že člen *f* účelovej funkcie neobsahuje optimalizačné premenné, teda vplýva na hodnotu účelovej funkcie ale nie na hodnotu argumentu a preto ho je možné z celého procesu optimalizovania vylúčiť [10].



Obr. 3. 7 MPC – Vplyv posledného členu účelovej funkcie na jej hodnotu a argument [10]

Z rovníc (3.24) – (3.28) je vidieť, že matica *P* je dôležitou súčasťou riešenia MPC. Tá sa vypočíta riešením Riccatiho rovnice (3.30) [10].

$$P - (A + BK)^T P(A + BK) = Q + K^T RK \tag{3.30}$$

$$K = (R + B^T PB)^{-1} B^T PA \tag{3.31}$$

Samotné riadenie prebieha v dvoch častiach, offline a online. V offline časti sú vypočítané všetky doterajšie časti algoritmu, ako je diskrétny stavový model, penalizačné matice, matice *H,G* a *F* poprípade aj matice obmedzení. Potom nasleduje online časť, kde sú vypočítané optimálne akčné zásahy v cykle na základe aktuálnych stavov. MPC riadiaci zákon bez procesných obmedzení má nasledujúci tvar [10].

$$\vec{u}_k = -H^{-1} G x_k \tag{3.32}$$

3.3.5 Procesné obmedzenia

Aplikácia obmedzení je celkom jednoduchá. Ide vlastne o saturáciu akčných zásahov alebo aj stavov na maximálnej alebo minimálnej povolenej hodnote. Obmedzenia sa uvádzajú v maticovom zápise [10].

$$A_c \vec{u}_k \leq b_0 \tag{3.33}$$

V prípade obmedzení na absolútnu hodnotu amplitúdy vstupov matice A_c a b_0 majú tvar

$$A_c = \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} \quad \text{a} \quad b_0 = \begin{bmatrix} \mathbf{1}\bar{u} \\ -\mathbf{1}\underline{u} \end{bmatrix} \tag{3.34}$$

kde matica ***I*** je štvorcová jednotková matica s rozmerom $dĺžka\ horizontu \cdot počet\ vstupov$, matica ***1*** je stĺpcový vektor jednotkových matíc *I*, $\bar{u}$ je horné ohraničenie a naopak $\bar{u}$ je dolné ohraničenie [10].
Pri obmedzovaní stavov je dôležité myslieť aj na obmedzení stavov v budúcnosti. To znamená, že obmedzenia musia platiť na celom predikčnom horizonte. Preto obmedzenia stavov vo všeobecnom stave majú tvar

$$A_c = \begin{bmatrix} \mathbf{N} \\ -\mathbf{N} \end{bmatrix}, \qquad b_0 = \begin{bmatrix} \mathbf{1}\bar{x} \\ -\mathbf{1}\underline{x} \end{bmatrix} \quad \text{a} \quad B_0 = \begin{bmatrix} -\mathbf{M} \\ \mathbf{M} \end{bmatrix} \tag{3.35}$$

a potom platí

$$A_c\vec{u}_k \leq b_0 + B_0 x_k \tag{3.36}$$

3.3.6 Kvadratické programovanie

Doterajšie informácie o MPC je možné zhrnúť do nasledujúcej rovnice [10].

$$u^* = \arg\min_u \left(\frac{1}{2}u^T H u + g^T u\right) \tag{3.37}$$

vzhľadom na podmienku

$$\begin{aligned} A_c u &\leq b_c \\ A_e u &= b_e \end{aligned} \tag{3.38}$$

Samotná minimalizácia kvadratickej účelovej funkcie s obmedzeniami nie je možné riešiť v uzavretom tvare, to znamená, že neexistuje explicitné riešenie. Je potrebné využiť numerické metódy. MATLAB pre riešenie minimalizačných úloh obsahuje v *Optimization Toolboxe* funkciu *quadprog* [15]. Pri zapísaní príkazu

```
x = quadprog(H,f,A,b)
```

MATLAB rieši minimalizačnú úlohu v tvare

$$\min_x \frac{1}{2} x^T H x + f^T x, \qquad \text{vzhľadom k} \qquad Ax \leq b \tag{3.39}$$

čo zodpovedá tvaru minimalizačnej úlohy MPC pre $x = u$. Keďže *quadprog* je numerická iteračná metóda, tak riešenie, ktoré nájde, nemusí byť vždy optimálne alebo ho nemusí nájsť vôbec. V programe *quadprog* je preddefinovaných niekoľko hlásení. Majú formu číslic a najdôležitejšie z nich a ich významy sú zobrazené v Tab. 3.1 [15].

Output	**Význam**
1	Nájdené zlučiteľné riešenie.
0	Prekročený počet krokov.
-2	Nenájdené zlučiteľné riešenie.
-3	Nekonečný problém.
-6	Nekonvexný problém.

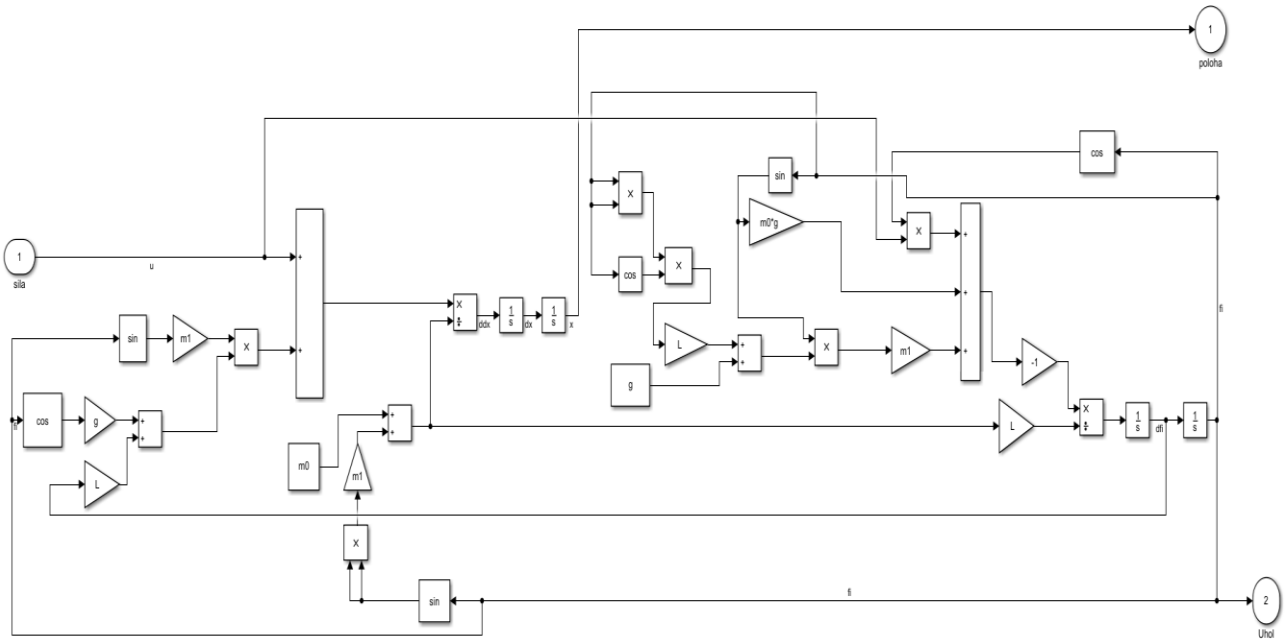
Tab. 3. 1 Ukončovacie hlásenia programu quadprog a ich význam

Význam prvých troch hlásení je jasný. Hlásenie *-3* udáva, že riešenie úlohy leží v nekonečne. Posledné hlásenie je použité, ak neplatí kladná definitnosť Hessiánu. Používateľ môže byť upozornený aj na nesymetrickosť Hessiánu, ktorú program automaticky vyrieši za užívateľa nasledovným spôsobom [10] [15].

$$H = \frac{H + H^T}{2} \tag{3.40}$$

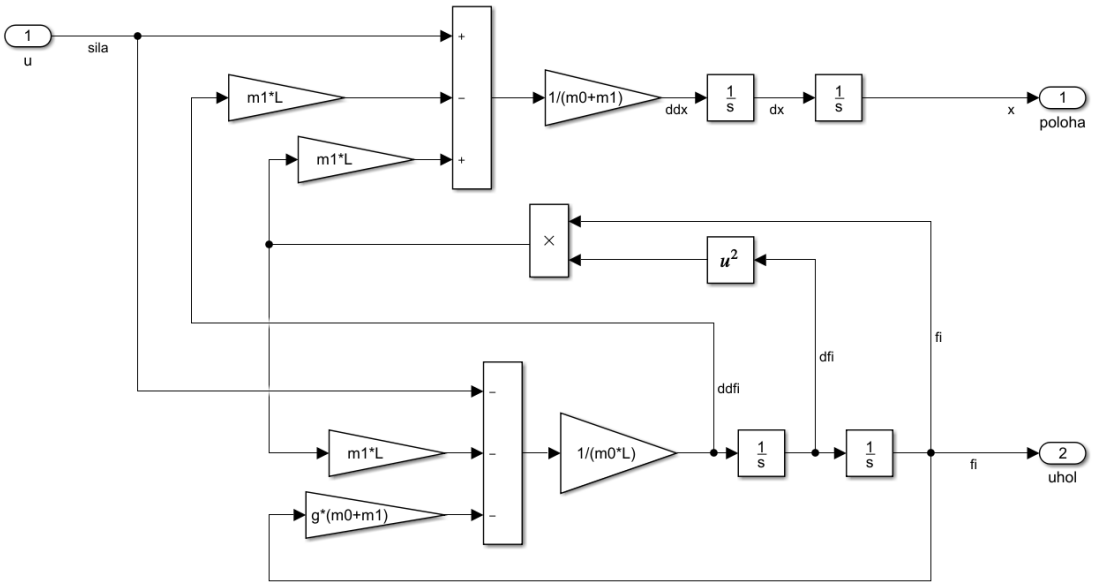
4. Aplikácia algoritmov v MATLABe

V tejto kapitole budú popísané algoritmy v prostredí MATLAB. Ide o numerické simulácie, ktoré budú ešte ďalej overované na reálnom modely a preto v prvom kroku bude potrebné vytvoriť matematický model lineárneho žeriavu v Simulinku. Tento model bol vytvorený podľa systému diferenciálnych rovníc (2.6) a (2.7).



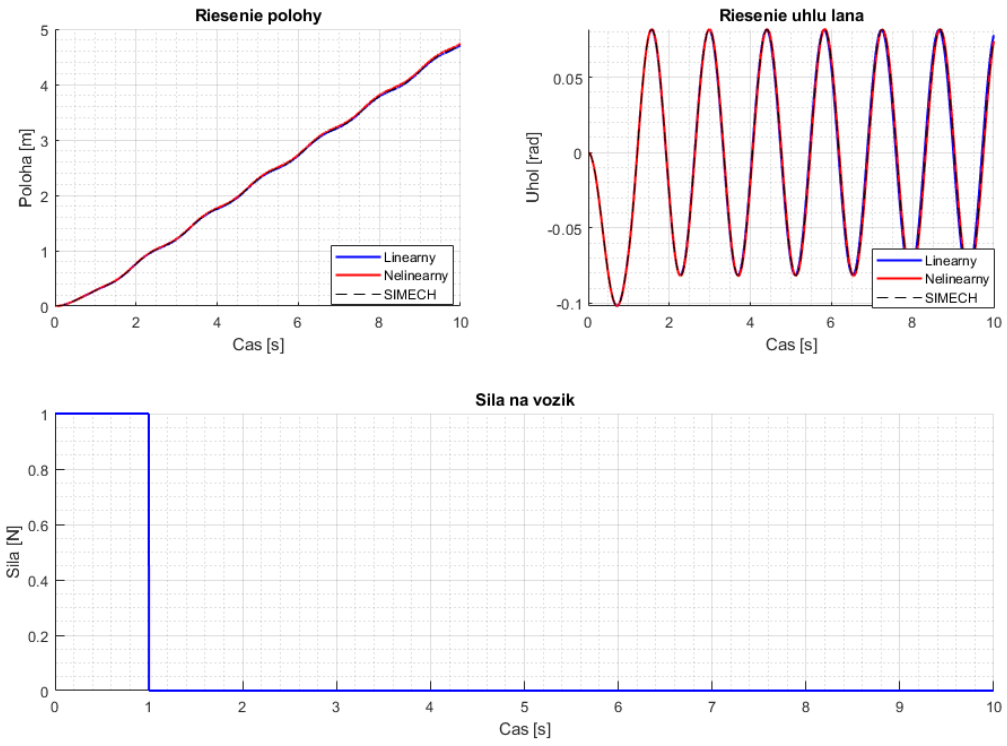
Obr. 4. 1 Nelineárny matematický model lineárneho žeriava v Simulinku

Z tohto modelu je odvodený lineárny model v Simulinku pomocou zavedenej linearizácie $cos\theta = 1, sin\theta = \theta$.



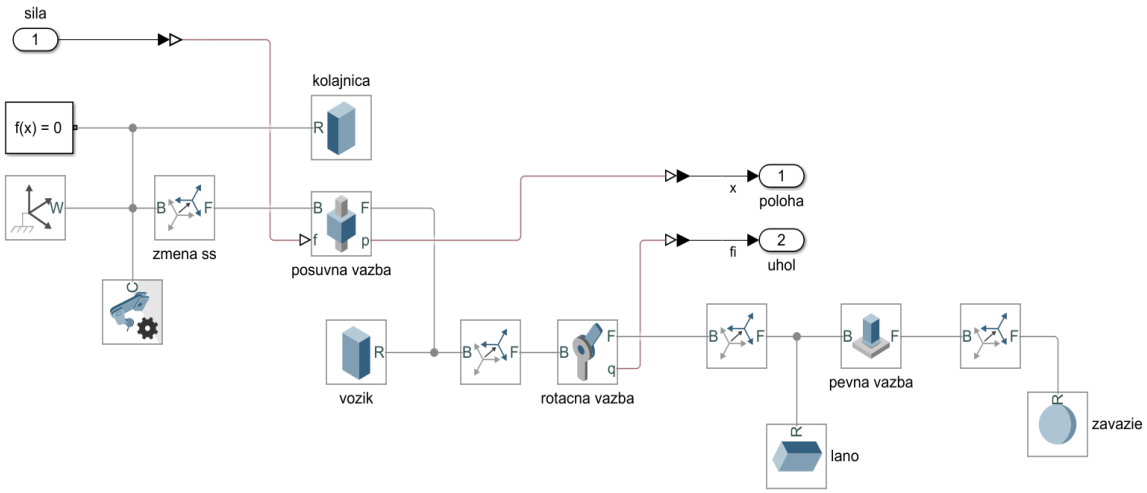
Obr. 4. 2 Lineárny model lineárneho žeriava v Simulinku

Z Obr. 4.1 a 4.2 je vidieť, že lineárny model je jednoduchší. Hlavný rozdiel je v tom, že lineárny model neobsahuje goniometrické členy. Na Obr. 4.3 je porovnanie chovania lineárneho a nelineárneho matematického modelu pri aplikácii rovnakého akčného zásahu. Je tu vidno, že linearizácia je prípustná, pretože chovanie je takmer totožné.



Obr. 4. 3 Porovnanie odozvy lineárneho a nelineárneho matematického modelu

Pre vývoj algoritmov je veľmi vhodné vytvoriť aj vizuálny model žeriavu pre lepšiu interpretáciu výsledkov. Taktiež tento vizualizačný model slúži ako potvrdenie odvodených rovníc systému z kapitoly 2. Model bol vytvorený v prostredí MATLAB pomocou rozšírenia Simscape. Toto rozšírenie slúži na modelovanie fyzikálnych sústav na základe definovania väzieb a telies systému. Model vytvorený v Simscape je zobrazený na Obr. 4.4.



Obr. 4. 4 Fyzikálny model lineárneho žeriavu vytvorený v Simscape

4.1 Návrh genetického algoritmu

Genetický algoritmus je nedeterministická metóda riešenia problému. Patrí medzi základné stochastické optimalizačné algoritmy s výraznými vývojovými črtami. Ako bolo spomenuté v kapitole 3.1, genetický algoritmus má 5 základných častí:

1. Inicializačná populácia
2. Hodnotiaca funkcia
3. Selekcia
4. Reprodukcia
5. Mutácia

Navrhnutý algoritmus sa bude držať tejto štruktúry s tým, že kroky 2 – 5 prebiehajú v cykle. Ešte pred samotným tvorením genetického algoritmu je potrebné zadefinovať parametre systému. Rovnako tomu bude aj pri všetkých ostatných algoritmoch. Ide konkrétne o hmotnosť vozíku m_0, hmotnosť nákladu m_1, dĺžku lana L, gravitačnú konštantu g a vzdialenosť, do ktorej sa má žeriav dostať x_f z počiatočnej polohy x_0. Okrem gravitačnej konštanty, ktorá bude uvažovaná pre všetky algoritmy a simulácie rovnaká $g = 9{,}81ms^{-2}$, budú parametre systému menené a pri jednotlivých výsledkoch konkrétne uvedené.

Je splnené, že stavy a sila pôsobiaca na vozík sú spojité alebo po častiach spojité a splnená je aj Dirichletova podmienka. Potom je možné akčný člen, silu pôsobiacu na vozík, predpísať v tvare Fourierovej rady [3].

$$y(t,p) = a_0 + a_1 \cos\left(\frac{\pi t}{T}\right) + b_1 sin\left(\frac{\pi t}{T}\right) + a_2 cos\left(\frac{2\pi t}{T}\right) + b_2 sin\left(\frac{2\pi t}{T}\right) + p_1 cos\left(\frac{3\pi t}{T}\right) + p_2 sin\left(\frac{3\pi t}{T}\right) + p_3 cos\left(\frac{4\pi t}{T}\right) + p_4 sin\left(\frac{4\pi t}{T}\right) \tag{4.1}$$

kde $p_1, \ldots, p_4$ sú voľné parametre a a_0, a_1, a_2, b_1, b_2 sú podmienené parametre.

$$\begin{aligned} a_1 &= -p_1 \\ a_2 &= -p_3 - a_0 \\ b_1 &= -3p_2 \\ b_2 &= -2p_4 \end{aligned} \tag{4.2}$$

Parameter a_0 je DC parameter, ktorý posúva celú funkciu o jednosmernú číselnú zložku kladným alebo záporným smerom.

V teoretickej časti bol jedinec popísaný ako binárny reťazec reprezentujúci nejaké číslo. Avšak pre problematiku návrhu trajektórie bude jedinec reprezentovaný ako vektor zložený zo štyroch parametrov $p_1, \ldots, p_4$.

$$jedinec = (p_1, p_2, p_3, p_4) \tag{4.3}$$

Genetické algoritmy nie vždy dokážu skonvergovať k požadovanému výsledku. Záleží to na programátorovej intuícii riešenia problému. Inicializačná generácia je vytváraná náhodne. V MATLABe tvorenie náhodných čísiel vyzerá nasledovne.

```
for i = 1:popsize
    pop(i,:) = (rand(1,paramNum) - 0.5)*10;
end
```

kde konštanta `popsize` predstavuje veľkosť populácie, `paramNum` počet voľných parametrov a do premennej `pop` sa zapisujú jedinci. Takto tvorené parametre sú v rozsahu (-5, 5), čo z testovania vyšlo ako dostačujúce.

Jednotliví jedinci sú postupne dosadení do rovnice (48), z ktorej sa vytvorí časový priebeh sily na vozík. Tá je následne použitá ako vstup do modelu lineárneho žeriava v Simulinku (Obr. 4.2). Jednotlivé stavy sú uložené do prostredia MATLAB a nasleduje počítanie hodnotiacej funkcie.

Hodnotiaca funkcia je vytvorená ako súčet všetkých stavov na konci simulácie, pričom sú sčítané absolútne hodnoty rozdielov od požadovaných koncových podmienok. Okrem polohy majú všetky stavy koncové podmienky položené v nule, takže rozdiel od koncovej podmienky je samotná hodnota daného stavu.

$$J = |x_{end} - x_f| + |\dot{x}_{end}| + |\theta_{end}| + |\dot{\theta}_{end}| \tag{4.4}$$

kde označenie *end* predstavuje poslednú hodnotu simulácie. Hodnotiaca funkcia je vypočítaná, pre každého jedinca a následne zoradená od najnižšej hodnoty po najvyššiu. Keďže ide o minimalizačnú úlohu, teda hodnotiacu funkciu je potreba minimalizovať, ideálne do nuly, tak hodnotiaca funkcia s najnižšou hodnotou predstavuje najlepšieho jedinca aktuálnej populácie.

Zo zoradenej hodnotiacej funkcie sa procesom selekcie vyberie isté percento najlepších jedincov, ktorí budú vybraní ako rodičia novej generácie. Toto percento je vybrané podľa úvahy programátora. Čím väčšie percento je zvolené, tým horší jedinci sa dostanú do procesu reprodukcie, takže je možné, že ich „zlé“ gény budú predané novým jedincom. Naopak nízke percento má za následok, že v ďalších krokoch bude faktor náhody hrať veľkú rolu vo vytváraní nových jedincov.

V procese reprodukcie sú z dvoch vybraných rodičov vytvorení dvaja potomkovia. Chromozómy predávané potomkom sú vybrané náhodne a tiež je zabezpečené, že druhý parameter rodiča môže byť u nového potomka napríklad parameter štvrtý. Tento proces je vytvorený funkciou `randperm` , ktorá vytvorí náhodný vektor permutácií $1{:}\,n_{parametrov}$. Po vytvorení potomkov je vytvorený vektor spojením rodičov a potomkov.

Počet voľných miest v populácii zostáva pre zmutovaných jedincov. Mutácia prebieha tak, že sa vyberie, koľko percent chromozómov zmutuje. Pri zvolení stupňa mutácie napríklad 50% zmutujú dva parametre zo štyroch. Samotná mutácia nemá analogicky daný tvar. Opäť závisí na intuícii programátora, ako proces mutácie vymyslí. V tomto prípade mutácia znamená

násobenie parametru náhodne vytvoreným číslom v intervale (-0.5 , 0.5) a ešte raz násobený celým číslom v rozsahu (0, 20).
Po tomto kroku nasleduje opäť hodnotenie jedincov rovnakým spôsobom ako u inicializačnej populácie, následne sú zoradení od najlepšieho po najmenšom a cyklus sa vracia do bodu selekcie.

Dôležité je spomenúť, že najlepší jedinci z každej generácie sú ukladaný, taktiež najlepšia hodnota hodnotiacej funkcie, ktorá je zároveň vypisovaná na terminál pre sledovanie priebehu algoritmu. Algoritmus je ukončený jednou z dvoch podmienok:

1. Počet generácií dosiahne maximálny počet, ktorý je zadaný na začiatku – týmto sa obmedzí nekonečný priebeh algoritmu, pri uviaznutí v lokálnom alebo globálnom extréme
2. Hodnotiaca funkcie klesne pod požadovanú hodnotu - hodnotiaca funkcia takmer nikdy neskonverguje do nuly, ak ide o komplexnejšiu úlohu. To znamená, že je potrebné vytvoriť istý kompromis, teda nájsť hodnotu hodnotiacej funkcie, ktorá je postačujúca pre riešenie problému.

4.1.1 Výsledky simulácie

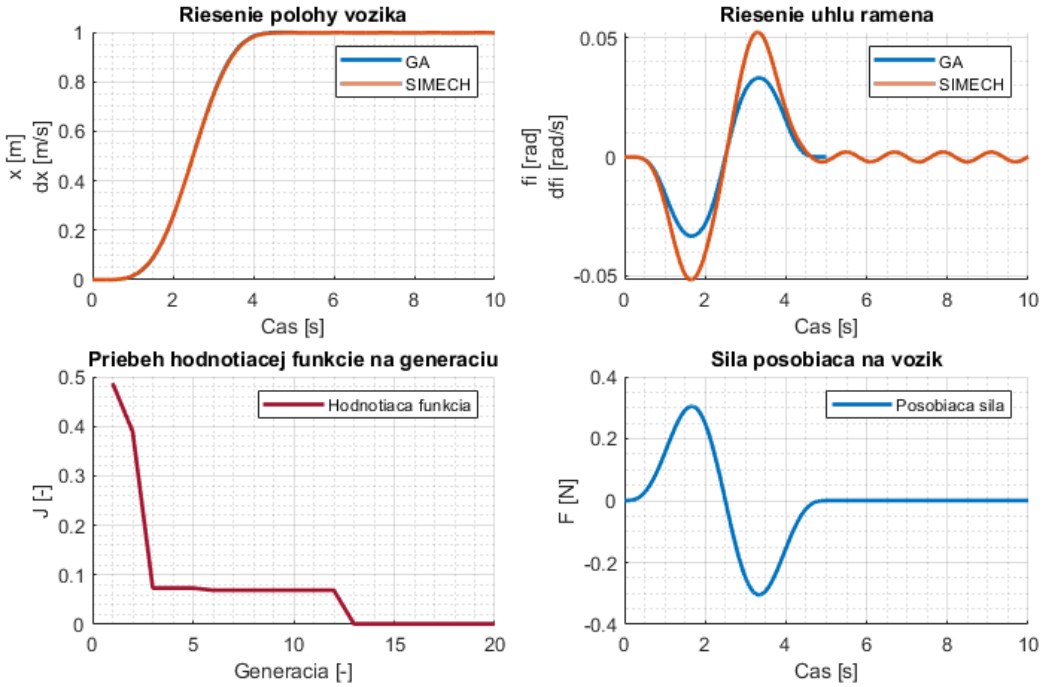
Genetický algoritmus je jeden z predstaviteľov algoritmov, v ktorých prevažuje faktor náhody. To znamená, že pri takýchto algoritmoch je potrebné zadávať väčšinu vecí intuitívne, keďže konvergencia je zaručená až v nekonečne. U všetkých algoritmov je kritické zadanie času, za ktorý sa má systém presunúť z miesta na miesto s dodržaním okrajových podmienok v závislosti na vzdialenosti o koľko sa má posunúť.

Pre zníženie časovej náročnosti na nájdenie riešenia bola zvolená dĺžka vzorkovacej periódy $Ts = 0.01$s. Na zobrazenie funkčnosti simulácie algoritmu bol zvolený dojazd $x_f = 1$m a čas, za ktorý sa má uskutočniť premiestnenie systému $t_f = 5$s. Parametre systému boli nastavené nasledovne.

Parameter	Hodnota
m_0	0.5 kg
m_1	0.2 kg
L	0.5 m

Tab. 4. 1 Nastavenie parametrov systému pre simuláciu

Riešenie genetického algoritmu pre vyššie uvedené nastavenia je zobrazené na nasledujúcom obrázku.



Obr. 4. 5 Riešenie simulácie genetického algoritmu

Ukončovacie podmienky algoritmu boli dve. Algoritmus ukončí hľadanie riešenia pokiaľ hodnotiaca funkcia klesne pod hodnotu $J = 0.001$ alebo počet generácií dosiahne hodnoty 20. Práve druhú podmienka bola dosiahnutá v hľadaní tohto riešenia, pričom hodnota hodnotiacej funkcie v poslednej generácii bola $J = 0.001176$.

Na obrázku vľavo hore je zobrazené riešenie polohy algoritmu aj zo Simechanics. Vpravo hore je riešenie uhlu ramena, kde možno vidieť slabé oscilovanie na konci simulácie, ale hodnota amplitúdy tejto oscilácie je $\theta = 0.002$ rad, čo odpovedá uhlu $\theta = 0.115°$. Vľavo dole je zobrazený priebeh hodnotiacej funkcie počas jednotlivých generácií a na poslednom grafe vpravo dole je zobrazený navrhnutý priebeh sily na vozík, ktorá bola aplikovaná v konečnej simulácií v Simechanics.

4.2 Návrh algoritmu problému okrajových hodnôt

Rovnako ako u predchádzajúcej metódy riadenia, aj tu je na začiatku potrebné zadať parametre systému. Tieto parametre sú následne uložené do globálnej štruktúry premenných, ktorá obsahuje tieto parametre. Voľba globálnych premenných vznikla preto, lebo algoritmus BVP je zložený z celkovo hlavného programu a troch vnorených funkcií, do ktorých vstupujú rovnaké premenné.

Po prepísaní definičnej rovnice BVP (3.1) pre riešený problém, vznikne nasledujúci predpis.

$$x' = f(t, x, p) \tag{4.5}$$

kde za pôvodnú *x*-ovú zložku bol dosadený čas a za *y*-ovú a jej deriváciu boli dosadené stavy systému. Voľné parametre zostali nezmenené. Z toho vyplýva, že na začiatku algoritmu je potrebné vytvoriť časový vektor, na ktorom je riešenie hľadané. Pre vytvorenie tohto vektoru je potrebné určiť dĺžku simulácie. Tá bude záležať na parametroch systému, pretože napríklad pre prípad dojazdu $x_f = 10\text{m}$ a váhe závažia $m_1 = 1000\text{kg}$ nie je možné, aby systém takéto závažie previezol danú vzdialenosť v ráde jednotiek sekúnd. Hoci by algoritmus našiel riešenie takéhoto problému, fyzikálne by bolo však nemožné ho splniť. To znamená, že opäť do algoritmu vstupuje intuícia užívateľa.

Nasleduje nastavenie možností funkcie *bvp4c*, ktoré sa zadávajú pomocou funkcie *bvpset*.

```
options = bvpset('RelTol',1e-6,'Nmax',1000,'stats',on);
```

V tejto funkcii je možné zadávať relatívnu alebo absolútnu odchýlku, ktorá sa počíta na základe zvyšku, maximálny počet bodov siete *Nmax* a tiež sa dá zapnúť či vypnúť zobrazovanie štatistické informácie. Tiež je možné zadať ukazovateľ na funkciu, ktorá analyticky počíta Jakobián, ale v tomto prípade nebolo potrebné meniť výpočet Jakobiánu algoritmom.

Po zadefinovaní možností algoritmu sa definuje počiatočný odhad riešenia. Počiatočné riešenie má samozrejme vplyv na konečné. Je možné ho zadať ako analytickú funkciu závislú na *x*-ovej zložke, ale tiež aj ako jedno číslo. Definuje sa pre všetky stavy, čiže je potrebný výstup vektoru o veľkosti (1 x počet stavov). Pre daný problém sa osvedčilo nezadávať analytické počiatočné riešenia, ale práve riešenie dané číslom. Najlepšie a najstabilnejšie výsledky algoritmus dával, ak boli všetky stavy na počiatku odhadnuté na hodnotu nula. Odhad definuje funkcia *bvpinit*.

```
solinit = bvpinit(time,@odeInit,[0 0 0 0]);
```

Vstupom do funkcie je jednak časový vektor, ale aj ukazovateľ na funkciu *odeInit*, v ktorej je zadané počiatočný odhad riešenia. Posledným vstupom do funkcie je počiatočný odhad voľných parametrov algoritmu. Keďže ich hodnota nemá analytický predpis ani konkrétnu hodnotu, tak všetky sú na začiatku položené rovno nule.

Nasleduje samotné definovanie funkcie *bvp4c*, ktorá už program posúva do časti samotného riešenia problému BVP. Predpis funkcie v MATLABe vyzerá nasledovne.

```
sol = bvp4c(@myODE, @odeBC, solinit, options);
```

Posledné dva vstupy ukazujú už na popisované veci, teda na možnosti (nastavenia) algoritmu a počiatočné odhady. Prvým vstupom vo funkcii je ukazovateľ na funkciu *myODE*, v ktorej je definovaný systém diferenciálnych rovníc systému. Element *sol* je štruktúra, ktorá v sebe obsahuje viaceré premenné:

x	Sieť vytvorená algoritmom. Väčšinou je zmenená oproti pôvodne nastavenej sieti užívateľom.
y	Aproximácia *y(x)* v bodoch siete *x*.
yp	Aproximácia *y'(x)* v bodoch siete *x*.
parameters	Finálne hodnoty voľných parametrov.
solver	Názov riešiča – 'bvp4c'
stats	Výpočtová štatistická hodnota – počet bodov siete, zvyšková chyba, počet volaní funkcií *myODE* a *odeBC*.

Tab. 4. 2 Pole premenných štruktúry riešenia `sol`

Keďže algoritmus BVP rieši úlohu pomocou voľných parametrov, tak sa využijú k počítaniu aktuálneho akčného zásahu rovnakým spôsobom pomocou Fourierovej rady ako tomu je pri genetickom algoritme podľa rovnice (48). Funkcia *myODE* má nasledovný predpis.

```
function dydt = myODE(t,y,p)
```

Vstupom je teda aktuálny čas a aktuálny vektor stavov. Funkcia beží v cykle od začiatku časového vektora po koniec a postupne aktualizuje stavy systému. V tejto funkcii sú zároveň definované diferenciálne rovnice prvého rádu popisujúce dynamiku systému, z ktorých sa počíta nasledujúci vektor stavov.

Ďalším vstupom do funkcie *bvp4c* je ukazovateľ na funkciu *odeBC*, ktorá v sebe obsahuje predpis okrajových podmienok. Predpis má nasledovný.

```
function res = odeBC(ya, yb, ~)
```

Vstupom je vektor stavov v bode A y_a, čo reprezentuje stavy na začiatku v čase $t = 0$ a vektor stavov v bode B y_b, čo reprezentuje stavy na konci časového vektoru $t = t_f$. Tretím vstupom je vektor voľných parametrov, ale v tomto prípade okrajové podmienku nie sú na týchto parametroch závislé. V tejto funkcii sú okrajové podmienky zapísané vyjadrením rovnosti.

$$y_{a(b)}(i) - bc_{a(b)}(i) = 0 \qquad i = 1, \ldots, počet\ stavov \tag{4.6}$$

To znamená, že súčet stavu v danom bode a okrajovej podmienky musí byť nula. Inak povedané, zapisuje sa to do funkcie *odeBC* tak, že sa napíše stav v danom bode (na začiatku alebo na konci) a za to hodnota okrajovej podmienky v danom bode s opačným znamienkom. Konkrétne v prípade lineárneho žeriava, kde je žiaduce, aby všetky stavy boli na začiatku rovné nule a na konci okrem polohy boli tiež rovné nule, sa to zapíše nasledovne.

$$\begin{aligned} BC = \; [\; & y_a(1) \\ & y_a(2) \\ & y_b(1) - x_f \\ & y_b(2) \\ & y_a(3) \\ & y_a(4) \\ & y_b(3) \\ & y_b(4)]\text{'}; \end{aligned}$$

Z toho vyplýva, že všetky stavy v bodoch A aj B sú rovné nule a tretím členom je okrajová podmienky polohy na konci časového vektora v tvare

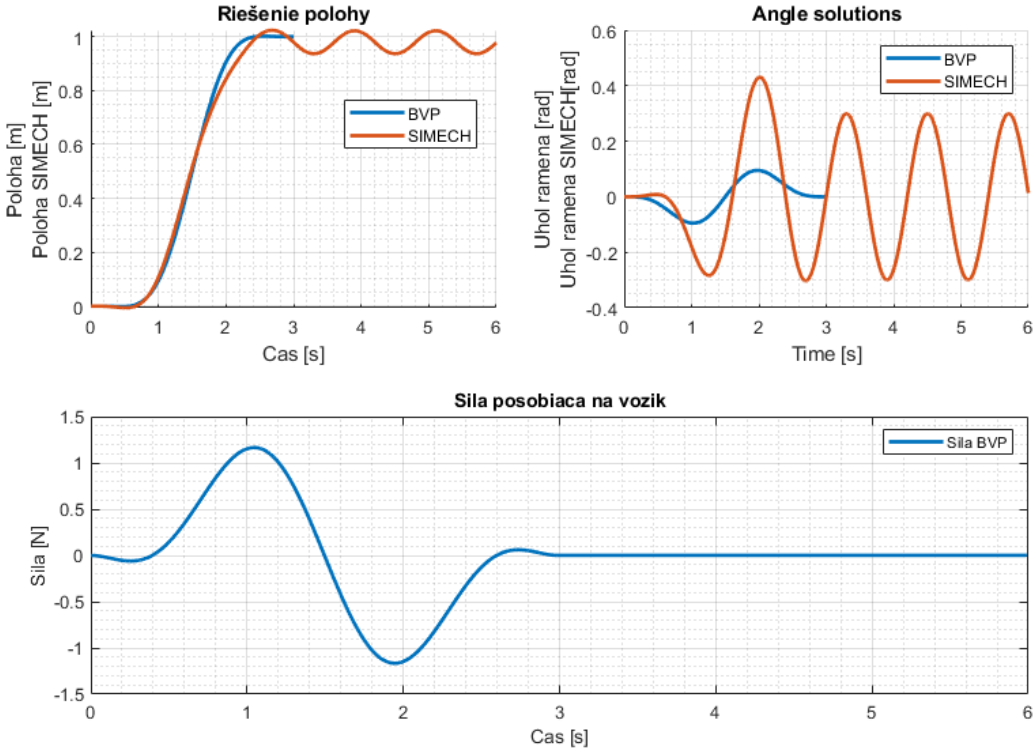
$$\begin{aligned} y_b - x_f &= 0 \\ y_b &= x_f \end{aligned} \tag{4.7}$$

Algoritmus teda najprv vypočíta počiatočný odhad na sieti bodov. Následne prvýkrát zavolá funkciu *myODE* pre vypočítanie stavov na začiatku a porovná ich s okrajovými podmienkami. Pokiaľ sa stavy zhodujú s podmienkami v zadanej tolerancii, tak pokračuje ďalej a počíta stavy až do konca časového vektora, kde na konci opäť porovnáva stavy s koncovými podmienkami. Ak všetko sedí, algoritmus končí, ak nie, nasleduje opakovanie výpočtu stavov, ale s menším rozptylom bodov siete.

4.2.2 Výsledky simulácie

Simuláciou je možné rozumieť zobrazenie správania reálneho modelu numerického výpočtu matematického modelu v časovej oblasti. Dôležité je realizovať vhodné simulácie, ktoré reálne môžu nastať na skutočnom zariadení, avšak počítač je schopný odsimulovať aj situácie, ktoré za bežného chodu nenastanú.

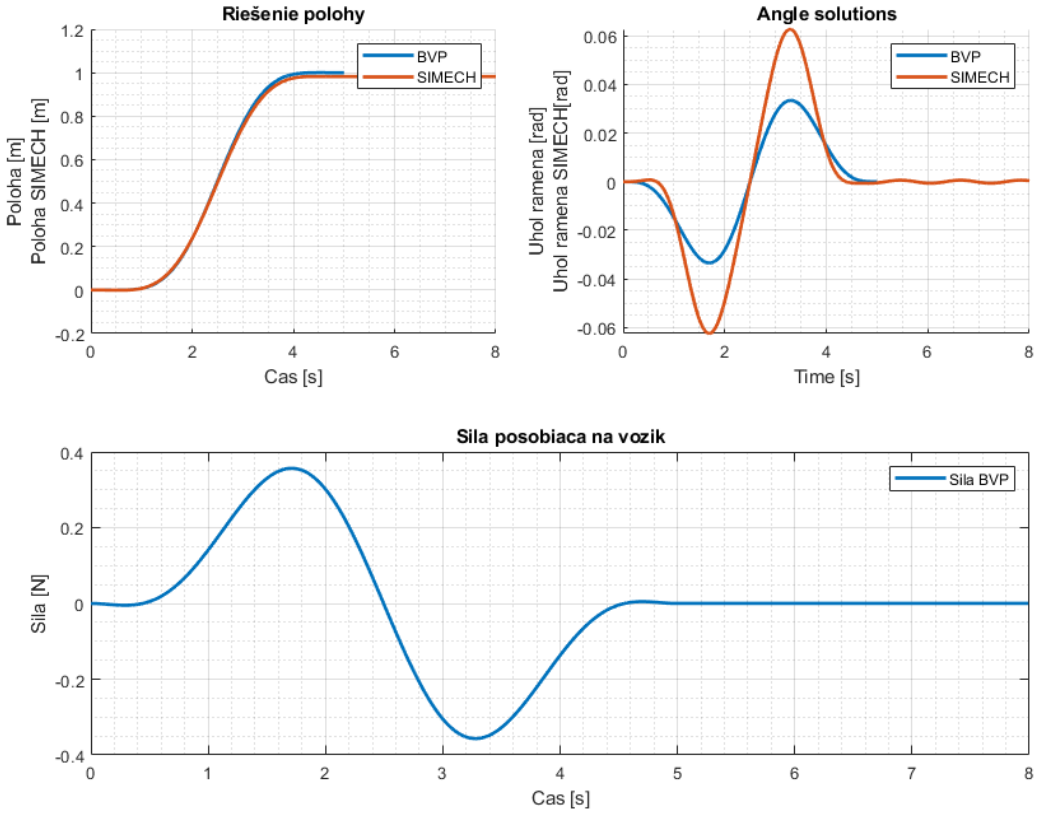
Pre vytvorenie vhodnej simulácie stačí zadať parametre modelu, konkrétne hmotnosti vozíku m_0 a závažia m_1, dĺžku lana L, dĺžku vzorkovacej periódy T_s, vzdialenosť, do akej sa má vozík presunúť x_f a v neposlednej rade aj čas, za ktorý má toto presunutie nastať t_f. Práve tento čas v spojení so vzdialenosťou x_f je kritickým parametrom. Pri zadávaní je potrebné zvážiť, či je fyzikálne možné presunúť vozík so závažím o vzdialenosť x_f za čas t_f. Ak táto podmienka nie je splnená, tak je systém nestabilný a výsledok z algoritmu BVP je nezmyselný a tým pádom irelevantný. Príklad takého správania je zobrazený na nasledujúcom obrázku.



Obr. 4. 6 Nestabilita riešenia pri zadaní nesprávnej kombinácie x_f a t_f

Na ľavej strane hore je zobrazené riešenie polohy algoritmom BVP a v Simechanics. Na pravej strane je riešenie uhlu ramena, kde je vidieť práve nestabilita riešenia. Na grafe dole je zobrazená navrhnutá sila pôsobiaca na vozík. V tomto prípade bol nastavený dojazd $x_f = 1\text{m}$ a čas $x_f = 3\text{s}$. Hoci premiestnenie na požadovanú vzdialenosť prebehlo, prebehlo takým spôsobom, že neboli splnené všetky okrajové podmienky na konci simulácie, konkrétne uhlová rýchlosť ramena, a teda v koncovej polohe nastal oscilačný pohyb závažia zaveseného na ramene.

Avšak ak sa trochu znížia časové požiadavky na systém a nastavíme rovnaký dojazd pri čase $t_f = 5\text{s}$ je premiestnenie stabilné. Táto simulácia je zobrazená na nasledujúcom obrázku.



Obr. 4. 7 Stabilné riešenie algoritmu BVP

4.3 Návrh algoritmu prediktívneho riadenia

Algoritmus opäť začína zadaním parametrov systému. Ďalej sa algoritmus líši od predchádzajúcich v tom, že je nutná linearizácia v okolí pracovného bodu. Na rozdiel od genetického algoritmu a BVP, ktoré buď pracovali s modelom v Simulinku alebo matematickým modelom v MATLABe, pre riadenie pomocou MPC je potrebné vytvoriť diskrétny linearizovaný stavový model. Keďže v MATLABe je možné zadať premenné a funkcie ako symboly, tak linearizácia je pomerne jednoduchá. Na zadanie symbolov sa použije príkaz

```
syms dxdt y x1 x2 x3 x4 u
```

čo znamená, že všetko za príkazom *syms* sú symbolické premenné. Následne je zapísaný stavový model pomocou symbolických premenných a tiež vektor výstupov $y = [x_1; x_3]$. Stavový model sa skladá z matíc A, B, C a D. Linearizované matice stavového modelu sa vypočítajú pomocou jakobiánu. Konkrétne v MATLABe to vyzerá nasledovne.

```
Alin = jacobian(dxdt,[x1 x2 x3 x4]);
Blin = jacobian(dxdt,u);
Clin = jacobian(y,[x1 x2 x3 x4]);
Dlin = jacobian(y,u);
```

Vstupom do funkcie jakobiánu je na prvom mieste funkcia, ktorú derivujem a následne vektor premenných, podľa ktorých derivujem. V premennej *dxdt* je definovaný stavový model a v premennej *y* sú výstupy. Po vyjadrení matíc systému sú za symbolické premenné dosadené hodnoty v okolí pracovného bodu. Pracovný bod bol zvolený nasledovne.

$$\mathrm{x_{lin}} = [0, 0, 0, 0];$$
$$\mathrm{u_{lin}} = 0;$$

Diskrétny stavový model sa v MATLABe vytvorí pomocou príkazu *c2dm*.

```
[A,B,C,D] = c2dm(Ac,Bc,Cc,Dc,Ts,'zoh');
```

Do príkazu vstupujú vyčíslené matice spojité systému Ac, Bc, Cc a Dc, vzorkovacia perióda Ts, podľa ktorej sa diskretizuje a posledným vstupom je metóda diskretizácie. Bola vybraná

zoh – zero order hold, čo znamená, že sa uvažujú vstupy, ktoré sa považujú počas trvania vzorkovacej periódy za konštantné.
Po úvodnej linearizácii a diskretizácii stavového modelu prichádza na rad definovanie samotných matíc algoritmu MPC. Prvé na rade sú penalizačné matice Q a R. Penalizačná matica stavov Q má inicializačný tvar

```
Q = C'*C*q;
```

kde pomocou násobku matíc C sa vytvorí štvorcová matica o veľkosti počtu stavov a násobok q slúži na nastavovanie samotnej hodnoty matice Q. Ak bolo požadované váhovanie jednotlivých stavov samostatne, musela by sa matica Q zadávať ručne celá.

Penalizačná matica vstupov R je v tvare skaláru, pretože do systému vstupuje iba jeden akčný člen. Táto hodnota sa volí ručne, ale obecne platí, že čím menšiu hodnotu má matica R, tým viac aplikujem akčný zásah do systému a naopak, čím väčšiu hodnotu má člen matice Q, tým dôležitejší je daný stav.

Ďalej sa volí dĺžka horizontu predikcie. Táto hodnota musí byť riadne zvážená, pretože malá hodnota nedokáže vytvoriť kvalitnú predikciu a naopak veľká hodnota značí náročnosť na výpočtový čas a kapacitu. Na rade je výpočet váhovacej matice koncového stavu P, ktorá sa počíta pomocou Ricattiho rovnice (3.30) a (3.31).

Predikčné matice M a N sa vytvoria podľa rovníc (3.13) a (3.14). Tieto matice poslúžia na vytvorenie matíc H,G a F. V teoretickej časti bolo vysvetlené, prečo matica F nie je nutná pre samotné MPC riadenie, preto je možné výpočet tejto matice vynechať. Aby program nemusel hlásiť zbytočné upozornenia, bola vyriešená aj nesymetrickosť Hessiánu nasledovným vzťahom.

```
H = (H+H')/2;
```

V tomto okamihu už je možné vytvoriť samotný cyklus minimalizačnej úlohy. Avšak, hoci ide len o počítačovú simuláciu, je vhodné vytvoriť aj matice obmedzení. Tie sa vytvoria pomocou vzťahov uvedených v teoretickej časti (3.34) a (3.35) s tým, že všetky stavy sa nechajú neobmedzené, čiže ich za ich horný limit sa zvolí ľubovoľné vysoké čísla, napr. milión, naopak dolný limit bude mať rovnaký tvar iba s opačným znamienkom. Obmedzenie sily pri simulácii nie je nutné, tak ako u stavov, ale keďže simulácie sa robia preto, aby čo najlepšie odrazili reálne chovanie, tak nie je žiadúce, aby do systému vstupovali sily, ktoré reálna sústava nie je schopná dodať. Čiže obmedzenie sily sa môže stanoviť napríklad na $u_{obmedzenie} = \pm 10N$. Rovnakou úvahou sa môžu obmedziť aj stavy, ale je postačujúce ak sa obmedzí iba poloha vozíka spolu so silou. Dolné obmedzenie vozíka môže byť nastavené napríklad na $\underline{x_1} = 0m$, tým je zabezpečené, že vozík sa nebude hýbať záporne zvoleným smerom od počiatočnej polohy a horné obmedzenie môže byť zvolené napríklad $\overline{x_1} = 2x_f m$. Zvolený je dvojnásobok práve preto, lebo pri riadení anti-swing systém občas prekročí zvolenú vzdialenosť dojazdu práve preto, aby zariadil nehybnosť závažia na konci pohybu.

Ešte pred samotným cyklom minimalizačnej úlohy je treba zadefinovať dĺžku trvania simulácie a tiež koncovú podmienku systému. V prípade anti-swing riadenia lineárneho žeriava sú všetky koncové podmienky stavov požadované rovné nule okrem stavu polohy vozíka. Takáto podmienka sa tým pádom v MPC riadení zadá pomocou násobku matíc Nx a r, pričom Nx má tvar jednostĺpcovej matice výstupov a r je číslo značiace vzdialenosť dojazdu.

```
Nx = [1; 0; 0; 0];
r = xf;
```

Ako bolo už spomenuté, minimalizačná úloha prebieha v cykle. Počet krokov cyklu závisí od zvolenej vzorkovacej periódy a zvolenej doby trvania simulácie. Čiže počet krokov simulácie sa vypočíta z nasledovného vzťahu.

$$i = \frac{t_{sim}}{Ts} \tag{4.8}$$

Pre hľadanie aktuálneho vstupu sa využije funkcia v MATLABe *quadprog*. Zadanie vstupov do tejto funkcie záleží od riešeného problému, čiže opäť sa odvíja od programátorovej intuície. Pre riešený systém lineárneho žeriava bola funkcia zadaná takto.

```
U(:,i) = quadprog(H,G*(X(:,i)-Nx*r),Ac,bc);
```

Takýto zápis znamená, že do *i*-teho stĺpca premennej *U* sa ukladajú vstupy predikované na celej dĺžke horizontu. Z teoretickej časti sú vstupy funkcie zrejmé. Prvým vstupom je matica *H*, druhým však nie je samotná matica *G*, ale keďže ide o riadenie do koncového stavu, tak matica *G* je násobená aktuálnymi stavmi *X*, od ktorých je odčítaná hodnota koncovej podmienky `Nx*r`. Tretí a štvrtý vstup do funkcie sú matice obmedzení.
Takto nájdený vektor vstupov následne vstupuje do časti simulácie systému cyklu, ktorá vyzerá takto.

```
X(:,i+1) = A*X(:,i)+B*U(1,i);
```

Počítajú sa stavy v budúcom kroku zo stavovej rovnice podľa rovnice (3.10), pričom do tejto rovnice vstupuje iba prvý člen vstupu, čo predstavuje aktuálny vstup v danom kroku.

4.3.1 Výsledky simulácie

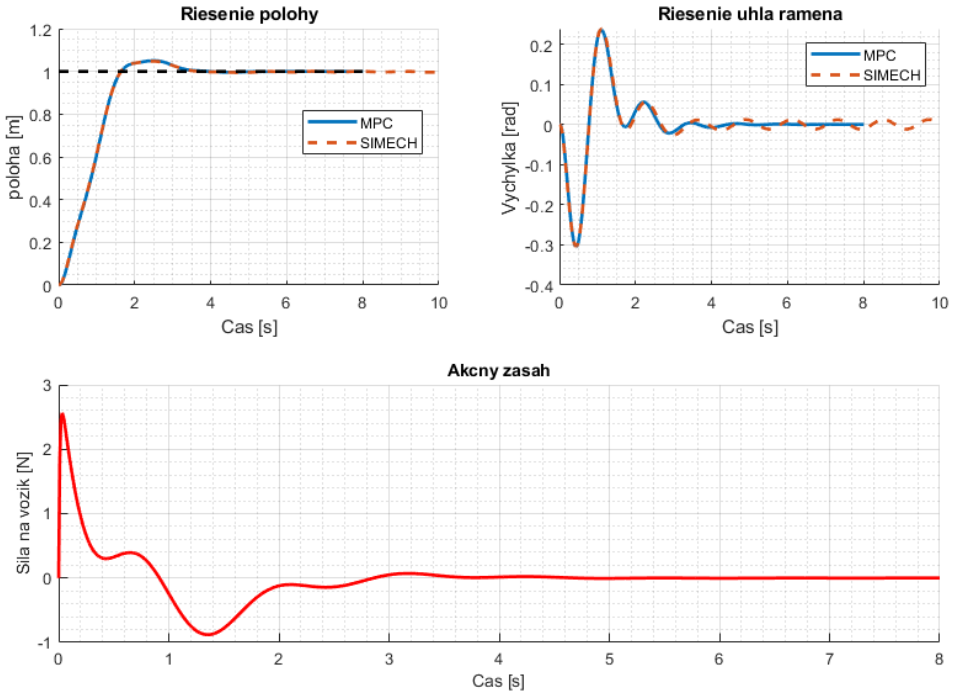
Kompletný algoritmus prediktívneho riadenia bol vytvorený podľa uvedenej predlohy z kapitoly 4.3 a celý proces riadenia je zameraný len na nastavenie penalizačných matíc *Q* a *R* a dĺžky predikčného horizontu n_p. Dĺžka horizontu je volená podľa toho, koľko chceme aby systém videl do budúcnosti. Ak je zvolená dĺžka horizontu $n_p = 1$, tak systém nevidí do budúcnosti a počíta priamo akčný zásah iba pomocou aktuálnych stavov z kroku *k* a nevidí, aké stavy nastanú v budúcnosti. Riadenie samozrejme prebehne aj pri takom nastavení, ale jeho presnosť je zredukovaná.

Po otestovaní algoritmu bolo zistené, že voľba vhodnej dĺžky horizontu je na intervale 50-100 krokov. Po zadaní dlhšieho predikčného horizontu nie je badateľná zvýšená presnosť, iba zvýšená výpočtová náročnosť a teda dĺžka simulácie.

Proces nastavovania matíc *Q* a *R* je obdobný ako napríklad nastavovanie zosilnení u PID regulátorov. Je to iteračný proces hľadania vhodnej kombinácie zosilnení penalizačných matíc. Všeobecne platí, že čím je nižšia hodnota zosilnenia penalizačnej matice

vstupov R, tým viac sú aplikované vstupy a opačne. Pre penalizačnú maticu stavov to platí naopak, teda čím vyššia hodnota zosilnenia matica má, tým viac je dbané na dodržanie predpísaných stavov, resp. okrajových podmienok.

Tak ako u predchádzajúcich algoritmov aj u MPC riadenia je dôležité dbať na voľbu časového vektoru, na ktorom je počítaná simulácia. Pre obdobné nastavenie ako u predchádzajúcich algoritmov, dojazd $x_f = 1\text{m}$ a čas simulácie $t_f = 8\text{s}$ je riešenie zobrazené na nasledujúcom obrázku.



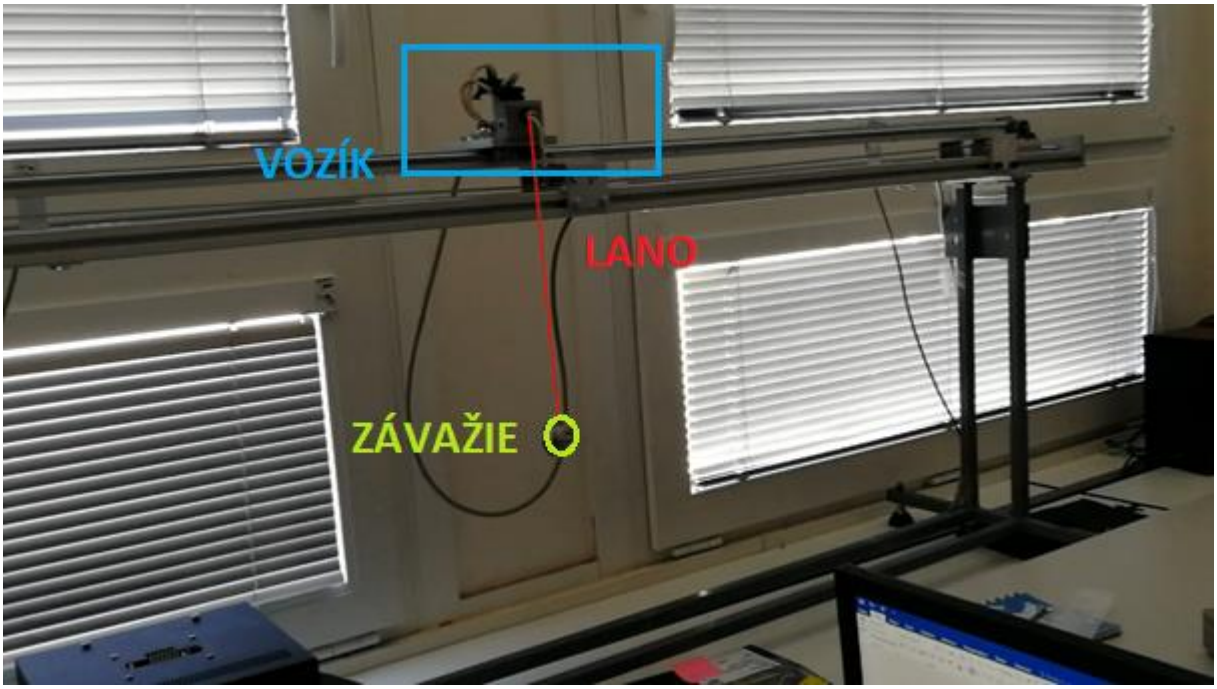
Obr. 4. 8 Riešenie simulácie MPC riadenia

5. Experiment

Riadenie systémov v počítačových simuláciách a reálnych systémov sa podstatne líši a v tejto kapitole budú algoritmy popísané v predošlých kapitolách aplikované na reálnom laboratórnom modely. Keďže riadiace algoritmy nie sú väčšinou tvorené univerzálne na rôzne systémy, tak v prvej časti bude popísaný samotný laboratórny model lineárneho žeriavu. Bude tiež popísaná odlišnosť reálneho systému od uvedeného teoretického systému, potrebné bude aj upraviť pohybové rovnice. Následne budú algoritmy aplikované na model, popíšu sa potrebné úpravy algoritmov a uvedú sa dosiahnuté výsledky.

5.1 Laboratórny model lineárneho žeriava

Model bol vytvorený v rámci realizácií bakalárskych a diplomových prác. Posledné úpravy na modely v rámci diplomovej práce realizoval Ing. Vít Slabý, ktorý upravoval model na dvojité kyvadlo a menil časť elektroniky. Keďže paralelne s touto prácou je realizovaná diplomová práca Bc. Tomáša Kirchnera, tak model bol od poslednej dostupnej dokumentácie trochu pozmenený. Pre riadenie lineárneho žeriava budú použité kyvadlá odmontované a nahradené závažím zaveseným na lane s nízkou hmotnosťou a tuhosťou, ktoré bude považované za bezhmotné.



Obr. 5. 1 Laboratórny model lineárneho žeriava

5.1.1 Konštrukcia modelu

Samotný model sa skladá z rámu s koľajnicou. Ide o konštrukciu z štvorcových oceľových profilov. Na vrchu je pripevnená koľajnica – lineárne vedenie ALUROL o dĺžke 1,6m. Rovnobežne s ňou je vedený ozubený remeň, ktorý je uprostred podoprený kvôli minimalizácii prehnutia. Na koncoch koľajnice sú z bezpečnostných dôvodov osadené skrutky, ktoré majú za úlohu zabrániť nekontrolovateľnému vyleteniu vozíka z koľajnice. Za rovnakým účelom boli 20 cm od samotného konca koľajnice osadené bezkontaktné indukčné snímače, ktoré pri detekcii vozíka vypnú simuláciu.

Vozík tvorí oceľová doska, ku ktorej sú zospodu priskrutkované štyri kolieska. Na vozíku je pripevnené skriňa s ložiskami, ktoré podporujú oceľovú osku, na ktorú sú pripevnené ramená kyvadla, v prípade lineárneho žeriava na ňu bolo pripevnené lanko so závažím. Taktiež sa na vozíku nachádza enkodér snímajúci natočenie osky, ktoré je priamo rovné natočeniu ramena kyvadla.

Systém je poháňaný motorom Transmotec PD4266-24-4, ktorý je cez spojku prepojený s ozubeným kolom, ktoré poháňa ozubenú remenicu pripevnenú na vozík. Parametre motora sú uvedené v Tab. 5.1.

Motor Transmotec PD4266-24-4		
Menovitý moment	M_{jm}	0,18 Nm
Menovité otáčky	n_{jm}	1445 min^{-1}
Konštanta motora	$c\phi$	0,0324 NmA^{-1}
Menovitý prúd	I_{jm}	2,1 A
Maximálny prúd	I_{max}	13 A

Tab. 5. 1 Parametre motora

Pretože systém lineárneho žeriava je prevažne riadený vstupnou silou na vozík, tak je potrebné uviesť vzťah prepočtu sily na prúd, resp. napätie. Prepočet sily pôsobiacej na vozík na moment motoru je nasledujúci.

$$M_m = r \cdot F_{vozík} = 0.0185 \cdot F_{vozík} \tag{5.1}$$

Konštanta *r* je polomer ozubeného kola, cez ktoré je motor prepojený s ozubenou remenicou, ktorá poháňa vozík. Následne prúd motora možno vypočítať takto.

$$I_m = \frac{M_m}{c\phi} = \frac{M_m}{0.0324} \tag{5.2}$$

Vzťah medzi prúdom a napätím je lineárne závislý na konštante. Napätiu 10V odpovedá prúd 6A, to isté platí pre záporné hodnoty. To znamená, že napätie sa z prúdu vypočíta jednoduchým vynásobením konštantou.

$$U_m = \frac{10}{6} I_m \tag{5.3}$$

Pre ovládanie motora je model osadený riadiacou jednotkou ESCON 70/10, ktorá obsahuje generátor PWM.

Súčasťou motora je aj enkodér, čo je zariadenie snímajúce natočenie pomocou logických signálov. Motor obsahuje inkrementálny enkodér, čo znamená, že natočenie sníma pomocou dvoch logických signálov. Hlavný parameter enkodéru v kvadratúrnom móde je CPR (*Counts per Revolution*), čo odkazuje na počet dekódovaných stavov, ktoré sa nachádzajú v jednej otáčke. Použitý enkodér má CPR = 76. Prevodový pomer motora je 4:1, takže na jednej otáčke pastorka odpovedá 304 čítaní enkodéru. Potom sa z počtu tikov enkodéru f_0 dá vypočítať rozlíšenie polohy na koľajnici s_0.

$$s_0 = \frac{2\pi r}{76 \cdot 4} f_0 \tag{5.4}$$

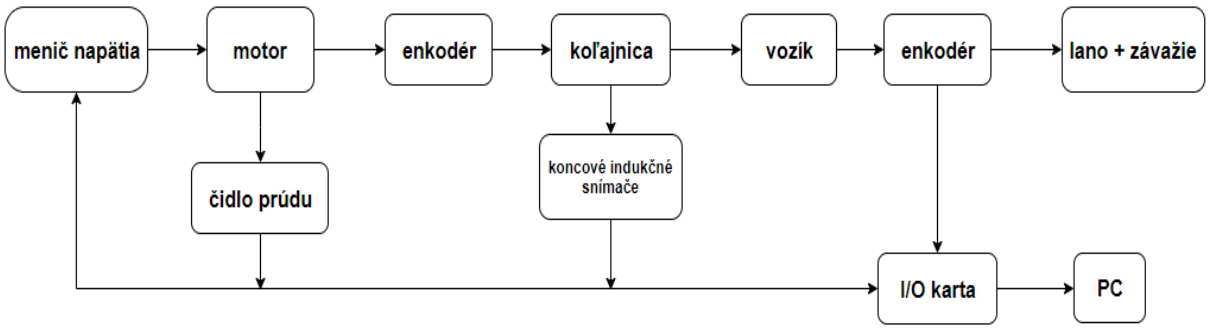
Pre $f_0 = 1$ tik je rozlíšenie polohy na koľajnici $s_0 = 0{,}36$ mm, čo je dostačujúce pre riešenie anti-swing riadenia.

Pre snímanie natočenia polohy kyvadla je použitý enkodér Broadcom/Avago HEDL-5540-I13 s dvoma výstupom i a s napájaním 5V. Hodnota CPR je 2048. Rozlíšenie natočenia ramena s_1 z počtu tikov f_1 je dané obdobným vzťahom ako u polohy.

$$s_1 = \frac{2\pi}{2048} f_1 \tag{5.5}$$

Opäť rozlíšenie natočenia tak ako aj u polohy je dostatočné. Pre $f_1 = 1$tik je rozlíšenie natočenia $s_1 = 0{,}00307$ rad.

Experimenty v tejto práci budú vykonané na popísanom laboratórnom modely. Na nasledujúcom obr. je pre ilustráciu zobrazená schéma celého testovacieho modelu lineárneho žeriava.



Obr. 5. 2 Schéma experimentálneho modelu

5.2 Úprava pohybových rovníc pre laboratórny model

Pohybové rovnice z kapitoly 2 sú nedostačujúce pre aplikáciu algoritmov na reálny model. Do spomenutého dynamického popisu nebolo zahrnuté trenie ani zotrvačnosť závažia na lane. Pri malých váhach je možné moment zotrvačnosti závažia zanedbať, ale ak by algoritmy boli aplikované pre reálne zariadenia v praxi, tak tam zotrvačnosť dosahuje oveľa väčších rozmerov a zanedbať ju nemožno. Pre moment zotrvačnosti platí vzťah ako pre jednoduché kyvadlo.

$$I = m_1 L^2 \tag{5.6}$$

Pohybové rovnice treba znova odvodiť rovnako ako v kapitole 2. Lagrangián má tvar

$$L = \frac{1}{2} m_0 \dot{x}^2 + \frac{1}{2} m_1 \left(\dot{x} + L\dot{\theta}\cos(\theta)\right)^2 + \frac{1}{2}(I + m_1 L^2)\dot{\theta}^2 - mgL\cos(\theta) \tag{5.7}$$

kde *I* predstavuje moment zotrvačnosti závažia na lane. Opäť deriváciami podľa rovníc (2.4) a (2.5) vznikne systém rovníc (5.8).

$$\begin{aligned} (m_0 + m_1)\ddot{x} + m_1 L\ddot{\theta}\cos\theta - m_1 L\dot{\theta}^2 sin\theta &= F - b\dot{x} - b_s \\ (I + m_1 L^2)\ddot{\theta} + m_1 L\ddot{x}cos\theta + m_1 gLsin\theta &= 0 \end{aligned} \tag{5.8}$$

kde v prvej rovnici členy na pravej strane b predstavujú vytvorený model trenia. Konkrétne b predstavuje dynamické trenie a b_s suché trenie.

Zo systému rovníc (5.8) je možné odvodiť stavový systém rovníc, ktorý však na prvý pohľad už nemá taký jednoduchý tvar ako v kapitole 2 pre ideálny model.

$$\begin{aligned} \dot{X_1} &= X_2 \\ \dot{X_2} &= \frac{m_1 L \frac{m_1 L\dot{\theta}^2 sin\theta cos\theta - b\dot{x}cos\theta - b_s + Fcos\theta + (m_0 + m_1)gsin\theta}{\left(\frac{(I + m_1 L^2)(m_0 + m_1)}{m_1 L} - m_1 L\cos^2\theta\right)}\cos\theta + m_1 L\dot{\theta}^2 sin\theta - b\dot{x} - b_s + F}{(m_0 + m_1)} \\ \dot{\theta_1} &= \theta_2 \\ \dot{\theta_2} &= \frac{-m_1 L\dot{\theta}^2 sin\theta cos\theta + b\dot{x}cos\theta + b_s cos\theta - Fcos\theta - (m_0 + m_1)gsin\theta}{\left(\frac{(I + m_1 L^2)(m_0 + m_1)}{m_1 L} - m_1 L\cos^2\theta\right)} \end{aligned} \tag{5.9}$$

Takýto stavový systém rovníc už naznačuje, že jeho riešenie si vyžaduje istú výpočtovú kapacitu. Je teda vhodné vytvoriť aj linearizovaný stavový model rovníc, ktorý výpočtovú náročnosť opäť posunie na nižšiu úroveň.

$$\dot{X_1} = X_2$$

$$\dot{X_2} = \frac{m_1 L \frac{m_1 L \dot{\theta^2} \theta - b\dot{x} - b_s + F + (m_0 + m_1) g\theta}{\left(\frac{(I + m_1 L^2)(m_0 + m_1)}{m_1 L} - m_1 L\right)} + m_1 L \dot{\theta^2} \theta - b\dot{x} - b_s + F}{(m_0 + m_1)} \tag{5.10}$$

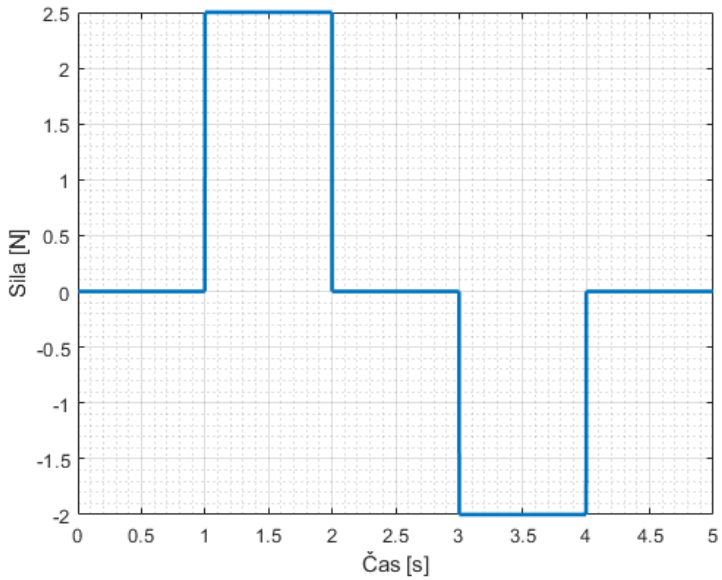
$$\dot{\theta_1} = \theta_2$$

$$\dot{\theta_2} = \frac{-m_1 L \dot{\theta^2} \theta + b\dot{x} + b_S - F - (m_0 + m_1) g\theta}{\left(\frac{(I + m_1 L^2)(m_0 + m_1)}{m_1 L} - m_1 L\right)}$$

5.3 Odhad parametrov

Ako už bolo vysvetlené, experimentálny model sa pomerne dosť líši od simulačného modelu. Všetky parametre reálneho systému ale však nie je možné zmerať a pre presné riadenie sú ich hodnoty kľúčové. V takýchto situáciách sa využíva nástrojov pre odhad parametrov. Aj MATLAB má takýto nástroj, ktorý sa volá Parameter Estimation. Ide o vbudovanú aplikáciu, ktorá má za úlohu pomocou numerických metód a opakovaných simulácií meniť hodnoty parametrov, dokiaľ nenájde taký súbor parametrov, ktoré odpovedajú reálnemu správaniu systému.

Dôležitou časťou pri odhadovaní parametrov je meranie. Ako prvé je potrebné zaznamenať správanie systému pri aplikácii určitého akčného zásahu. Toto meranie nemá predpísaný tvar. Bol teda zvolený istý tvar vstupu do systému, teda sila na vozík a následne bude zaznamenaná odozva z vytvoreného modelu.



Obr. 5. 3 Vstup do systému pre odhad parametrov

Zo stavového systému rovníc (5.9) sa vytvorí model v Simulinku, ktorý bude slúžiť ako simulačný model pre odhad parametrov. Síce parametre ako hmotnosti a dĺžka lana je možné zmerať, tak budú odhadované aj tie spolu s koeficientami modelu trenia a momentom zotrvačnosti, lebo v systéme rovníc (5.9) nie sú zahrnuté parametre zvyšku systému, ako napríklad zotrvačnosť a trenie v motore, remenici, ozubených kolách, celkové straty energie a pod., ktoré ovplyvňujú správanie modelu.

Po odhadnutí parametrov však odsimulovaná odozva systému stále úplne nezodpovedala nameraným výstupom a preto bol ešte pridaný lineárny koeficient redukovanej hmotnosti, ktorá zohľadňuje všetky straty po ceste od momentu z motora po silu pôsobiacu na vozík.

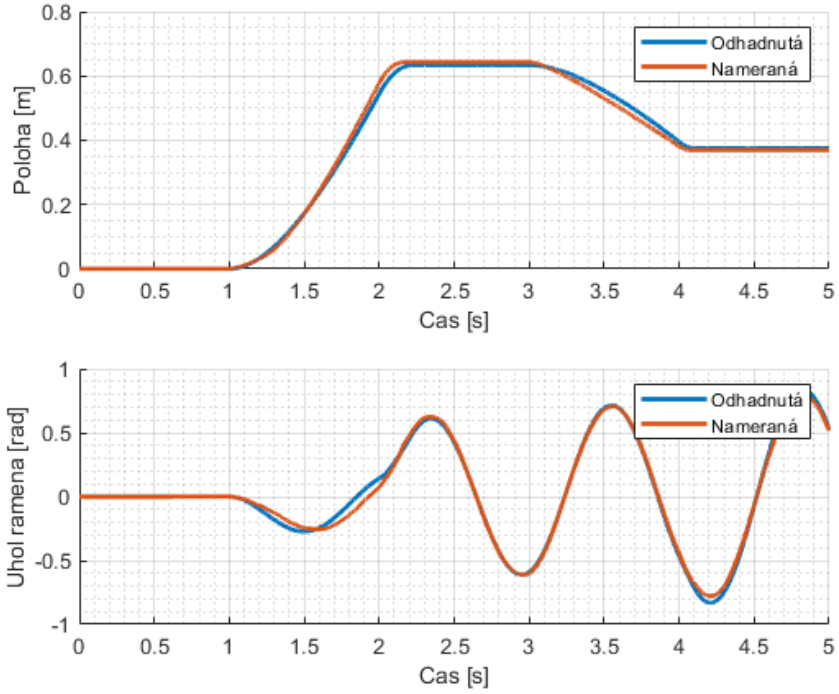
$$\ddot{x}_v = M_{red} \cdot \ddot{x} \tag{5.11}$$

Hodnoty odhadnutých parametrov sú v Tab. 5.2.

Parameter	Hodnota
I	$0.71994\ e^{-3}\ kgm^2$
L	$0.06597\ m$
m_0	$0.369\ kg$
b	0.87644
b_s	1.6439
m_1	$0.03938\ kg$
M_{red}	0.80028

Tab. 5. 2 Odhadnuté parametre systému

Priebeh nameranej odozvy systému na vstup z Obr. 5.3 a simulovanej s odhadnutými parametrami sú zobrazené na Obr. 5.4.



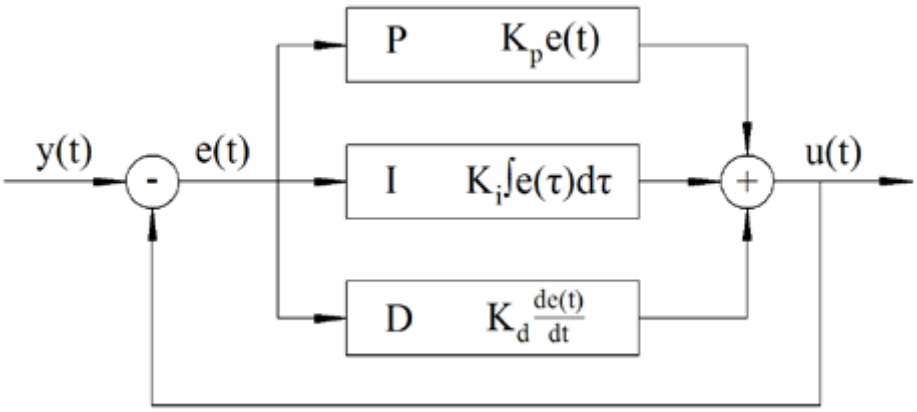
Obr. 5. 4 Nameraná a simulovaná odozva systému s odhadnutými parametrami

5.4 Riadenie silou a rýchlosťou

Vytvorené algoritmy majú za úlohu navrhnúť časový priebeh sily, ktorá premiestni vozík so závažím z počiatočného miesta do koncového bez veľkých výchyliek závažia. Pri implementácii riadenia silou na reálny model došlo na problém, že vozík sa nesprával v každom mieste koľajnice rovnako, pravdepodobne trenie pôsobiace proti pohybu vozíka nie je všade rovnaké a tiež záleží aj do ktorej strany sa vozík pohybuje. Takéto správanie nie je možné v algoritme vystihnúť a pre riadenie v otvorenej slučke, teda bez spätnej väzby, má kritické následky. V skratke, vozík sa nesprával tak ako algoritmus predpokladal a navrhol.

Pre riadiace algoritmy v otvorenej slučke riešenie ale existuje. Je to riadenie rýchlosťou a uzavretie riadiacej slučky. Pre takýto typ riadenia je nutné, aby algoritmus navrhol nielen časový priebeh sily, ale aj časový priebeh rýchlosti vozíka. Keďže silu pôsobiacu na vozík nejde jednoducho merať, ale jeho rýchlosť áno, tak je možné vytvoriť uzavretú slučku so spätnou väzbou a je možné správanie vozíka regulovať. Vstupom do riadiacej slučky teda nebude sila pôsobiaca na vozík, ale požadovaný časový priebeh rýchlosti vozíka, ktorá bude v spätnej väzbe regulovaná regulátorom. Je možné použiť najjednoduchší typ regulácie, t.j. regulácia PID regulátorom.

PID regulátor je všestranne použiteľný nástroj. Ide o typ riadenia so spätnou väzbou, ktorý má za úlohu nulovať odchýlku požadovanej hodnoty veličiny od aktuálnej hodnoty veličiny. Schéma PID regulátoru je na Obr. 5.5.



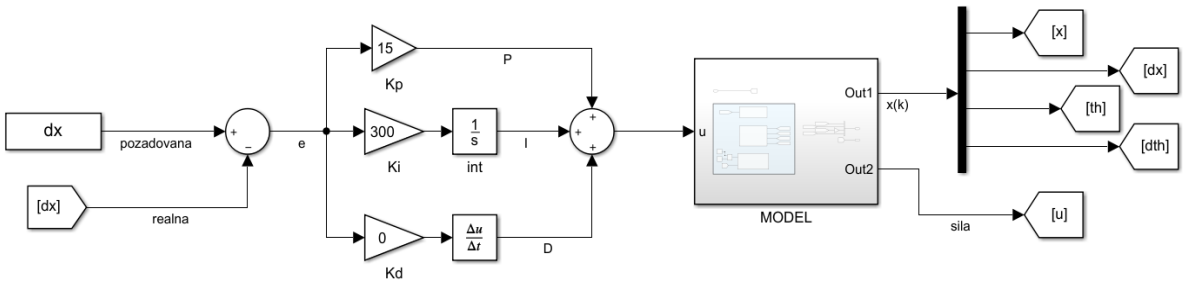
Obr. 5. 5 Schéma PID regulátora [19]

Najzložitejšia vec na PID regulácii je nastavenie zosilnení jednotlivých členov regulátora K_p, K_i a K_D. Premenná $y(t)$ je požadovaná aktuálna hodnota veličiny, $u(t)$ je vypočítaný aktuálny akčný zásah, ktorého rozdielom od požadovanej hodnoty $y(t)$ vznikne aktuálna odchýlka $e(t)$. Jednotlivé členy regulátora majú význam P – *proporcionálnej* (zosilňujúcej) *zložky,* I – *integračnej zložky* a D – *derivačnej zložky.*

5.5 Aplikácia problému okrajových hodnôt

Algoritmus BVP sa pri aplikácii na model lineárneho žeriava moc nemenil. Zmenený bol systém diferenciálnych rovníc pridaním modelu trenia. Po tom ako algoritmus vytvorí časový priebeh sily spolu s časovým priebehom všetkých stavov systému, tak je potrebné aplikovať výsledok na model.
Ako bolo spomenuté v predošlej kapitole, tak model nemá konštantné trenie po celej dĺžke koľajnice a tiež parametre trenia záležia aj na smere pohybu. Toto správanie sa v algoritme v doterajšom stave nedá vystihnúť a preto je potrebné vytvoriť riadenie so spätnou väzbou.

BVP algoritmus vytvorí časový priebeh rýchlosti vozíku, ktorý v uzavretej slučke poslúži ako vstup do PID regulátora. Model riadenia pomocou PID regulátora je zobrazený na nasledujúcom obrázku.



Obr. 5. 6 Riadenie lineárneho žeriava rýchlosťou pomocou PID regulácie

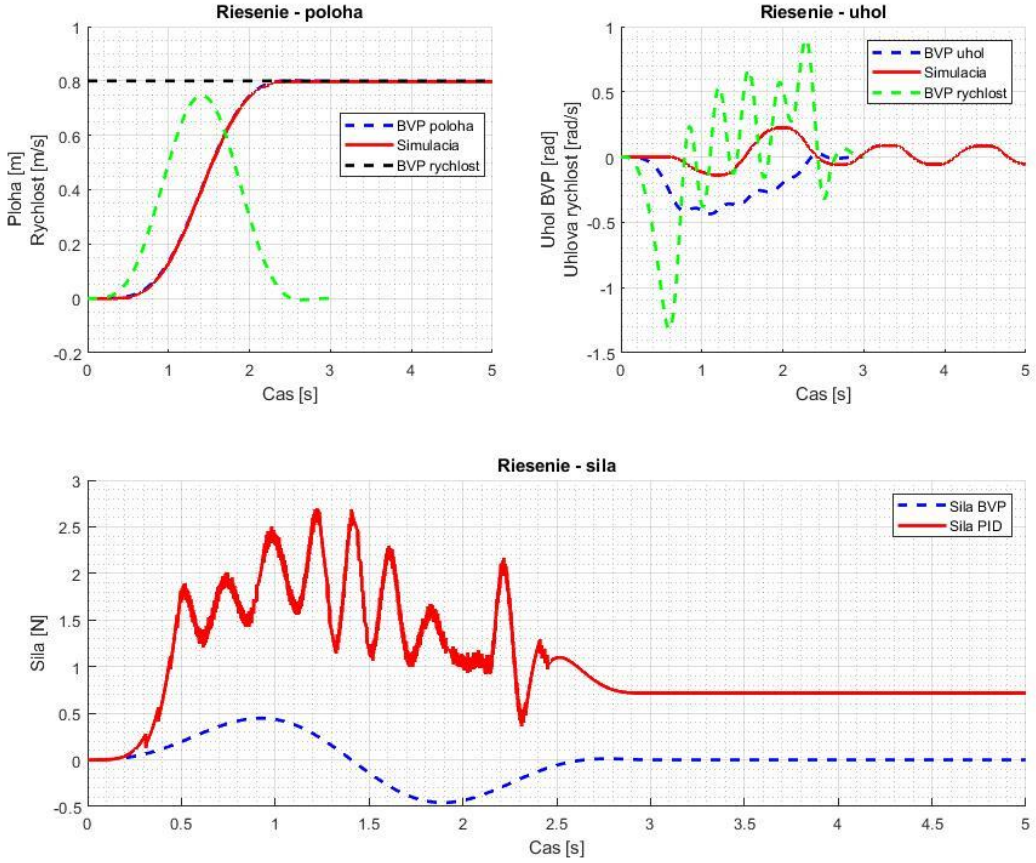
Na ľavej strane je počítaná odchýlka požadovanej rýchlosti od reálnej rýchlosti vozíku, odchýlka prejde PI regulátorom (derivačné zosilnenie K_D je nulové) a výstup z regulátora je opäť aplikovaný do modelu ako akčný zásah na vozík.
Poslednou úlohou ostáva ladenie PID regulátora, konkrétne jednotlivých zosilnení. Výsledný PID regulátor má nasledovné parametre:

Zosilnenie	**Hodnota**
K_P	15
K_I	300
K_D	0

Tab. 5. 3 Hodnoty zosilnení PID regulátora

5.5.1 Výsledky riadenia

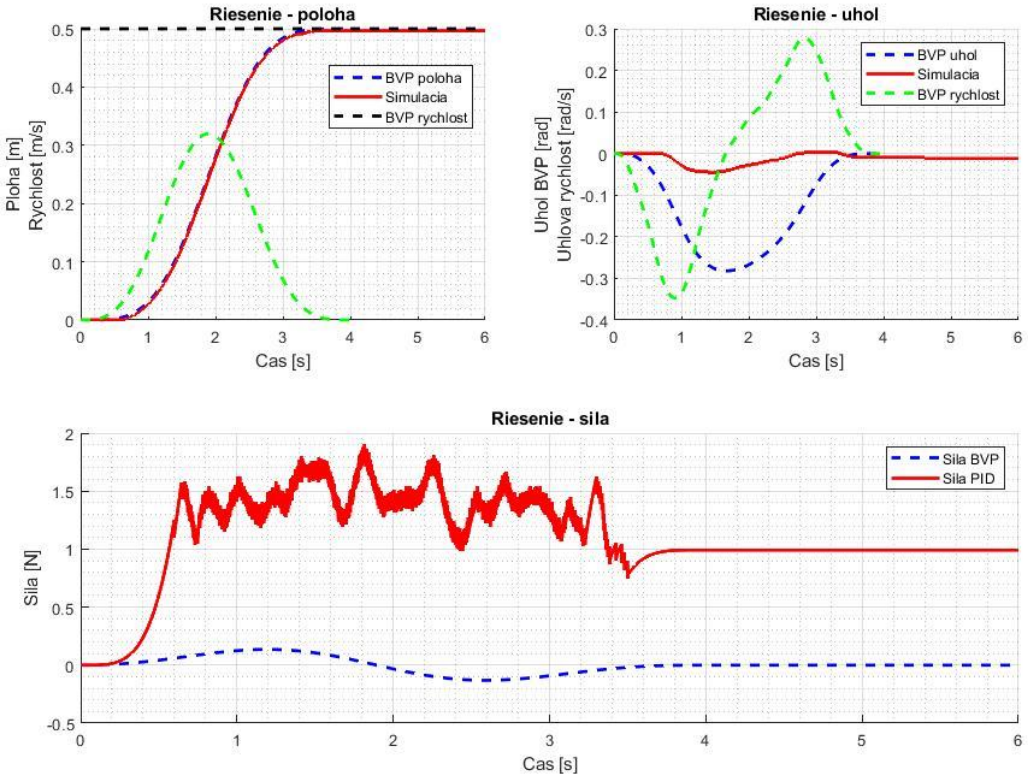
Efektivita riadenia sa odvíja od zvoleného časového vektoru. Ten je zadávaný na začiatku algoritmu v podobe zvolenia koncového času a vzorkovacej periódy. Koncový čas je potrebné voliť s ohľadom na vzdialenosť akú má vozík prejsť. Pri voľbe vysokého času nie sú následky riadenia tak zlé ako pri opačnej situácii. Anti-swing riadenie bude splnené, iba bude potrebné čakať na vykonanie pohybu žeriava. Pri voľbe príliš nízkeho času už riadenie anti-swing nemusí byť dodržané v požadovanej miere. Taký prípad je napríklad zobrazený na nasledujúcom obrázku.



Obr. 5. 7 Riešenie BVP pri voľbe krátkeho časového vektora

Okrajové podmienky polohy a rýchlosti vozíka (vľavo hore) sú splnené, na druhej strane uhol ramena a uhlová rýchlosť závažia (vpravo hore) nesplňujú konečnú podmienku, konkrétne uhol závažia osciluje okolo rovnovážnej polohe v rozsahu $x_3 = \pm 0.1$ rad $\approx$ 5,73°. V dolnej polovici na obrázku je zobrazený časový priebeh sily. Modrá čiarkovaná čiara predstavuje výsledok z algoritmu BVP proti červenej plnej čiare, ktorá predstavuje reálny akčný zásah z riadenia rýchlosťou PID regulátorom.

Pri voľbe správneho časového vektoru je riadenie anti-swing dodržané. Príklad takej simulácie je zobrazený na nasledujúcom obrázku.



Obr. 5. 8 Výsledok riadenia anti-swing využitím BVP

Pre požadovanú koncovú vzdialenosť od začiatku $x_f = 0.5\text{m}$ a koncový čas $t_f = 4\text{s}$ vyzerá riešenie anti-swing riadenia využitím BVP ako na Obr. 5.8. Všetky okrajové podmienky boli dodržané, v tomto prípade uhol závažia na lane je ustálený v $x_3 = 0.005\,\text{rad} \approx 0.287°$. Keďže v laboratórnom modeli je použitý inkrementálny snímač polohy, tak všetky hodnoty polohy sa odvíjajú od počiatočnej hodnoty. To znamená, že ak nebolo závažie na lane v úplnom pokoji na začiatku simulácie, tak je pravdepodobné, že inkrementálny snímač polohy definoval nulový uhol závažia na lane nenulovej skutočnej hodnote. Ak by šlo o reálnu aplikáciu, bolo by vhodnejšie použiť namiesto inkrementálnych snímačov polohy absolútny snímač polohy, ktorý v každom momente vie určiť skutočnú hodnotu polohy.

5.6 Aplikácia prediktívneho riadenia

Rozšírenie rovníc o model trenia bol uplatnený aj v aplikácii MPC riadenia. Okrem samotných diferenciálnych rovníc boli rozšírené aj matice systému *A, B* a *C* za účelom vytvorenia riadenia inkrementálnym akčným zásahom. To znamená, že predikované akčné zásahy nebudú v absolútnom tvare, ale v prírastkovom tvare závislom na predchádzajúcej hodnote. Takéto riadenie môže dopomôcť lepšej dynamike systému a stabilnejším rozbehom.

Matice systému *A, B* a *C* budú počítané nasledovným spôsobom.

$$A_{ink} = \begin{bmatrix} \mathrm{eye}(n_u) & \mathrm{zeros}(n_u, n_x) \\ B & A \end{bmatrix} \tag{5.12}$$

$$B_{ink} = [\mathrm{eye}(n_u) \quad \mathrm{zeros}(n_u, n_x)] \tag{5.13}$$

$$C_{ink} = [\mathrm{zeros}(n_u, n_x) \quad \mathrm{C}] \tag{5.14}$$

kde príkaz *eye* vytvorí nulovú maticu s jednotkami na hlavnej diagonále o veľkosti počtu vstupov n_u, príkaz *zeros* vytvorí nulovú maticu o veľkosti n_u x n_x, kde n_x je počet stavov. Zmena nastane ešte vo výpočte penalizačných matíc.

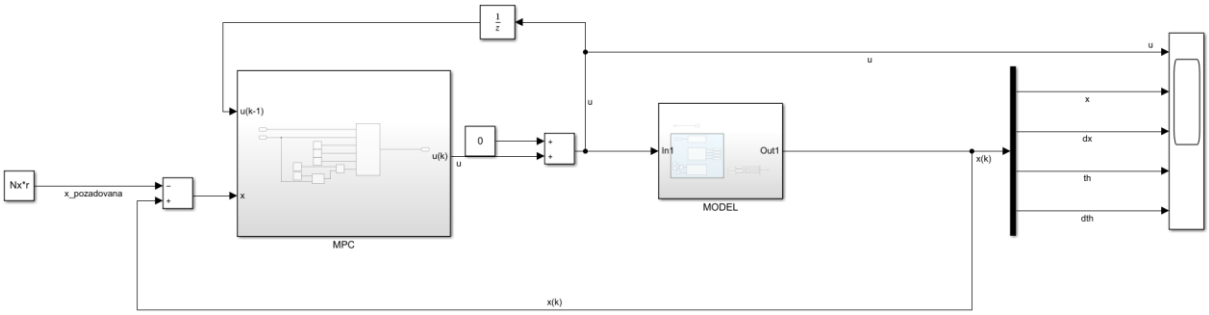
$$Q_{ink} = \begin{bmatrix} R & \mathrm{zeros}(n_u, n_x) \\ \mathrm{zeros}(n_x, n_u) & Q \end{bmatrix} \tag{5.15}$$

$$R_{ink} = 1 \tag{5.16}$$

Ďalšia zmena je už len v samotnej simulácii vo výpočte akčného zásahu. V minimalizačnej úlohe nie je počítaný priamo akčný zásah ale jeho prírastok Δu. Následne akčný zásah je dopočítaný z jednoduchej rovnice

$$u(k+1) = u(k) + \Delta u \tag{5.17}$$

Pri aplikácii na model je možné použiť riadenie v online uzavretej slučke, čo znamená, že vypočítaný akčný zásah je hneď v kroku *k* aplikovaný na systém a v kroku *k+1* sa z aktuálnych stavov počíta ďalší akčný zásah. V Simulinku teda model MPC riadenia vyzerá nasledovne.



Obr. 5. 9 MPC riadenie v Simulinku

Minimalizačná úloha je v Simulinka zadaná cez blok *Matlab Function* použitím funkcie *quadprog*. Vypočítaný akčný zásah vstupuje ako sila pôsobiaca na vozík do subsystému *MODEL*, v ktorom je uložené všetko potrebné pre riadenie modelu žeriava. Stavy z reálneho systému idú cez spätnú väzbu a slúžia na výpočet ďalšieho akčného zásahu.

5.6.1 Výsledky riadenia

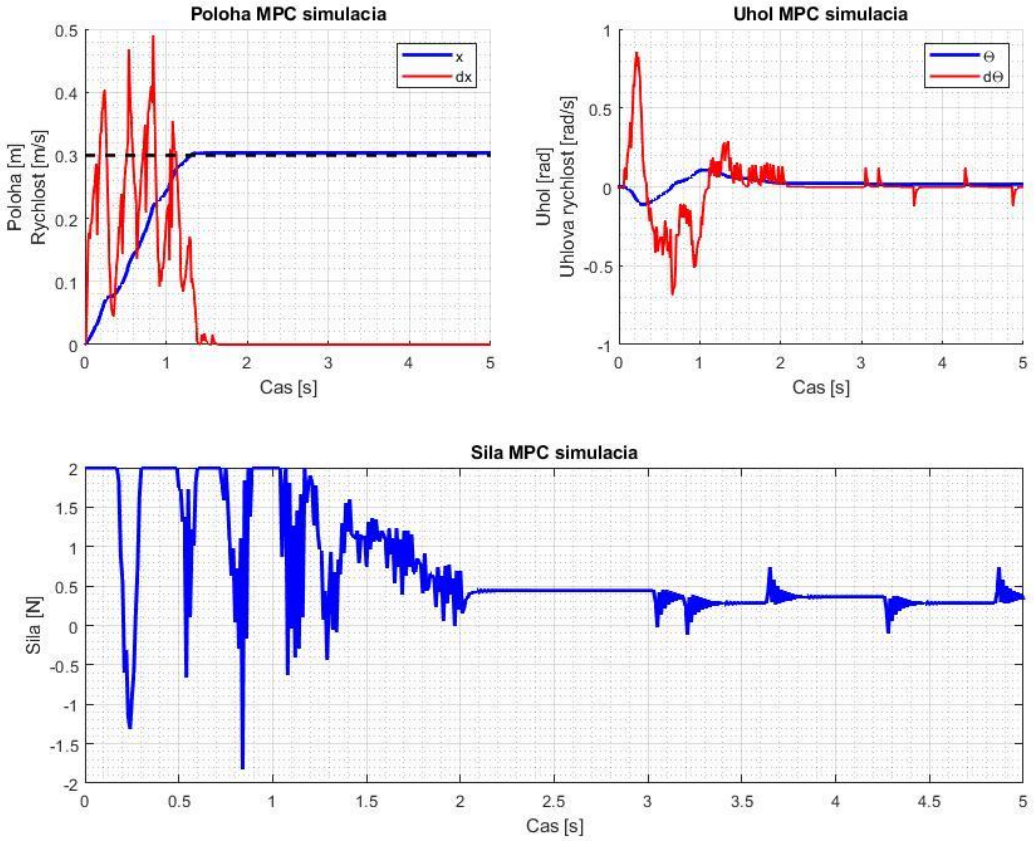
Na rozdiel od riadenia BVP, riadenie MPC sa neodvíja len od zvoleného časového vektoru, ale aj od nastavenia penalizačných matíc a zvolenej dĺžky predikčného horizontu. Aby bolo možné realizovať riadenie v online uzavretej slučke v Simulinku bolo potrebné zvoliť dĺžku vzorkovacej periódy $Ts = 0.01\text{s}$. Pre nižšie hodnoty počítač nestíhal počítať akčný zásah pre nasledujúci krok v tak krátkom čase. Voľba dĺžky horizontu je vyslovene úlohou programátora, ale z realizovaných pokusov je odporúčaná aspoň hodnota $n = 20 - 50$ krokov. Pre simulácie bola však zvolená hodnota $n = 150$ krokov, čo odpovedá predikcii 1,5 sekundy dopredu pri dĺžke vzorkovacej periódy $Ts = 0.01\text{s}$.

Penalizačné matice majú veľmi rozdielne hodnoty pre riadenie v inkrementálnom tvare alebo v priamom tvare. Simulácie boli realizované riadením v inkrementálnom tvare a hodnoty penalizačných matíc boli nastavené nasledovne.

Nastavenie hodnôt penalizačných matíc	
matica	**hodnota**
Q	4500
R	0.5

Tab. 5. 4 Hodnoty penalizačných matíc MPC algoritmu

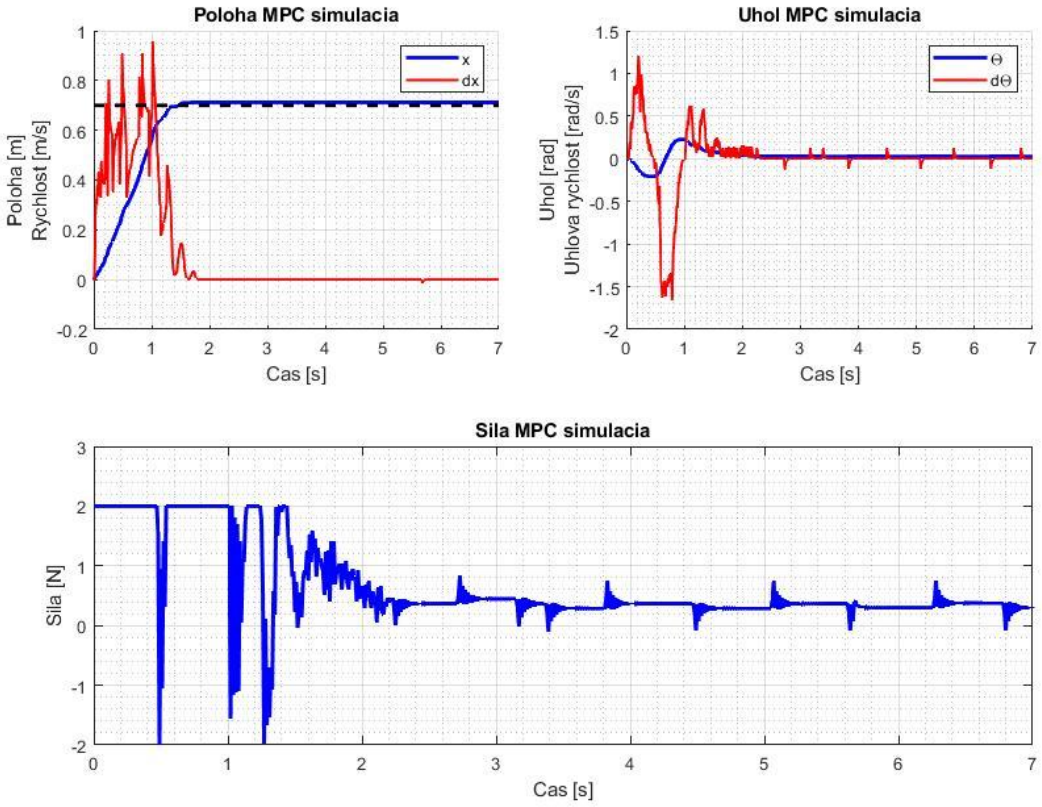
Algoritmus nemal problém nájsť vhodné riešenie pre krátky či dlhý dojazd, ako aj pre krátky či dlhý zvolený časový vektor. Napríklad pre dojazd $x_f = 0.3\text{m}$ a koncový čas $t_f = 3\text{s}$ riadenia MPC algoritmom vyzeralo nasledovne.



Obr. 5. 10 Výsledok MCP riadenia lineárneho žeriava, $x_f = 0.3m$, $t_f = 3s$

Je možné si všimnúť, že sila bola obmedzená na intervale $F = \pm 2\text{N}$. Keďže sa jedná o online riadenie, tak je vidieť, že pri odchýlke od koncových okrajových podmienok algoritmus stále dopočítava akčný zásah pre dorovnanie týchto nerovností. Keďže sú ale veľmi malé a trenie vysoké, tak sú viditeľné iba v grafickej podobe, nie však na reálnej sústave v podobe trhania.

Na nasledujúcom obrázku je uvedený príklad MPC riadenia pre dlhší dojazd $x_f = 0.7\text{m}$ a časový vektor $t_f = 5\text{s}$.



Obr. 5. 11 Výsledok MPC riadenia lineárneho žeriava, $x_f = 0.7m$, $t_f = 5s$

Nastavenie matíc ovplyvňuje aj čas, v ktorom dosiahnu koncových okrajových podmienok. Je vidieť, že aj pri zvolenom čase simulácie $t_f = 5\text{s}$ bol žeriav ustálený už pri polovičnom čase. Zo simulácií bolo zistené, že MPC algoritmus nemá taký hladký chod ako ostatné dva algoritmy BVP a GA. Aj zo samotných grafov je vidieť, že hlavne uhlová rýchlosť závažia na lane je oveľa väčšia, približne 4-5 násobná proti algoritmu BVP. V pôvodných experimentoch však šlo hlavne o čo najrýchlejší presun závažia z bodu A do bodu B.

5.7 Aplikácia genetického algoritmu

Genetický algoritmus, ktorý bol vybraný k preskúmaniu v tejto práci sa síce javí ako najjednoduchší algoritmus, ale jeho aplikácia je pomerne zložitá. Je to preto, lebo celý algoritmus je založený na veľmi jednoduchých matematických operáciách bez akejkoľvek intuície, či inteligencie dostať sa čo najskôr k požadovanému výsledku.

Ako bolo spomenuté v minulej kapitoly u genetického algoritmu, celý proces hľadania výsledku je vo vlastnej podstate náhodný. Dôležité je nájsť kompromis čo sa týka počtu jedincov. Opäť platí, že čím viac jedincov bude vytvorených v iniciálnej generácii, tak síce je vyššia šanca, že sú medzi nimi jedinci blízki hľadanému riešeniu a po pár iteráciách môže byť dosiahnuté riešenie. Naopak ale platí, že vyšší počet jedincov má za následok ďaleko vyššiu výpočtovú náročnosť a čas, za ktorý sa dôjde k riešeniu môže byť naopak kontraproduktívny.
Z týchto dôvodov musela byť skrátená vzorkovacia perióda opäť ako pri riadení MPC na hodnotu $Ts = 0{,}01$s. Tým pádom simulácia každého jedinca je kratšia a výsledkom sú aj celkové kratšie iterácie. Po pár pokusoch sa osvedčilo voliť veľkosť populácia v intervale 100 – 200 jedincov. V takomto rozmedzí trval výpočet jednej iterácie do 60s.
Ukončovacie podmienky boli vytvorené na základe podmienok z minulej kapitoly. Teda ukončenie algoritmu nastane v dvoch prípadoch:

1. *Počet iterácií dosiahne maximálny dovolený počet* – tento počet je nastavený na začiatku algoritmu. Pre testovanie bol zvolený počet iterácií (generácií) v rozsahu 20-30.
2. *Hodnota hodnotiacej funkcie klesne pod nastavenú hodnotu* – hodnota je opäť nastavená na začiatku algoritmu. Z tejto podmienky vychádza, že riadenie nebude nikdy perfektné, teda hodnotiaca funkcia konverguje do nuly, ale až v nekonečne. Pre testovanie bola táto hodnota nastavená na 0.001.

Z vyššie uvedeného je jasné, že samotné testovanie genetického algoritmu je náročný proces hlavne z časového hľadiska. Pre spomínané nastavenia bolo normálne, že výpočet trval 5-10 minút a nemusel ani nájsť vhodné riešenie. Treba brať do úvahy, že popisovaný genetický algoritmus v sebe neobsahuje žiadnu inteligenciu, čiže nejaký prvok, ktorý podporí skoršiu konvergenciu k výsledku. Takýmto prvkom môže byť napríklad sofistikovanejšie narábanie s jedincami, ako napr. iná stratégia výberu jedincov, či iný druh mutácie. V implementovanom algoritme boli nadradené prvky náhodných operácií.

5.8 Zhodnotenie algoritmov

Základným cieľom pre experiment bolo navrhnúť vhodný popis systému lineárneho žeriava tak, aby dosiahol čo najlepšie výsledky riadenia pohybu s ohľadom na podmienky anti-swing. Jednotlivé algoritmy získavajú riešenie problému rôznym spôsobom. Všetky skúmané algoritmy zhodnotíme v troch bodoch s prihliadnutím na aplikovaný problém, ktorý bol riešený v tejto práci.

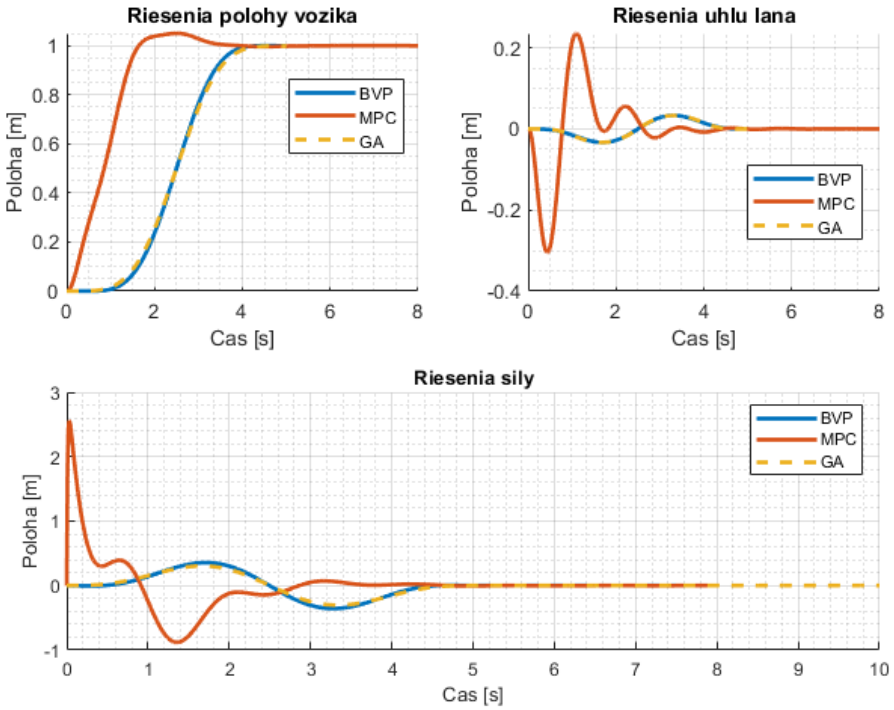
Ako prvé budú algoritmy zhodnotené z *teoretického hľadiska*, ktoré bolo popísané v 3 kapitole. Bol opísaný teoretický základ o všetkých použitých algoritmoch z tejto práce. Prvým algoritmom bol genetický algoritmus, ktorý sa na prvý pohľad javí ako najjednoduchší a naozaj tomu tak je. Je to preto, lebo nemá konkrétnu podobu, musí však obsahovať minimalizačnú úlohu a isté korekčné vzťahy, ktoré napomôžu konvergencii riešenia. Algoritmus BVP, ktorý rieši problém okrajových hodnôt je pomerne zložitá matematická úloha, preto bola využitá funkcia *bvp4c* v programe MATLAB. Avšak k tejto funkcii existuje iba veľmi slabá dokumentácia a preto na pochopenie použitého algoritmu tejto funkcie je potrebné hľadať v rôznych publikáciách a veľa testovania pre správne zadefinovanie problému. Za algoritmom prediktívneho riadenia je však ešte viac matematických vzťahov ako za algoritmom BVP, preto okrem mnohých maticových operácií bola využitá funkcia *quadprog* v programe MATLAB. Algoritmus MPC je preto uvedený ako najsofistikovanejší algoritmus z pomedzi uvedených.

V ďalšom bode budú algoritmy zhodnotené z *hľadiska počítačových simulácií*. Počítačové simulácie algoritmov sú uvedené v kapitole 4. Genetický algoritmus a algoritmus BVP boli vytvorené na rovnakom základe a to využitím popisu sily ako Fourierovej rady. Hoci z uvedených výsledkov nie je badateľný veľký rozdiel, rozdiel pozná hlavne obsluha, ktorá čaká na riešenie. V nasledujúcej tabuľke sú uvedené časy, ktoré zobrazujú ako dlho daný algoritmus počítal riešenie.

Algoritmus	**Čas [s]**
BVP	*0.477*
GA	*143.818*
MPC	*1.279*

Tab. 5. 5 Porovnanie časovej náročnosti algoritmov

Genetický algoritmus dosiahol takmer rovnakého výsledku (Obr. 5.12) približne po 300 násobne dlhšom čase (Tab. 5.5) ako algoritmus BVP, ktorý využíva MATLAB funkciu *bvp4c*. MPC algoritmus pri použití v offline simuláciách na PC narazil na problém menšieho prekmitu pred samotným dosiahnutím požadovaných koncových stavov. Okrem toho MPC algoritmus fungoval veľmi dobre. Od ostatných algoritmov sa líšil najmä v rýchlosti pohybu systému. Pri prvom pohľade na Obr. 5.12 s výsledkami je vidieť, že sila na vozík aj samotné stavy mali pri riadení MPC odlišný priebeh ako u algoritmov GA a BVP, ale ustálenie systému nastalo zhruba v rovnakom momente.



Obr. 5. 12 Porovnanie riešení počítačových simulácií jednotlivých algoritmov

V poslednom bode budú algoritmy hodnotené *z hľadiska aplikácie na reálnu sústavu*. Aplikácia algoritmov na reálnu sústavu si vyžadovala aj menšie úpravy v algoritmoch o čom pojednáva kapitola 5. Rovnaké závery pre GA a algoritmus BVP platia aj u tejto úlohy. GA síce nájde dobré riešenie, ale konvergenciu k tomuto riešeniu má zaručenú až v čase $t = \infty$. Oba algoritmy, BVP aj MPC, dosahovali veľmi dobré výsledky. Hlavný rozdiel bol v aplikácii akčného zásahu do systému. BVP aplikoval dopredu vypočítaný priebeh sily cez PI regulátor, pričom MPC aplikoval silu v online uzavretej slučke, čo znamená, že pre aktuálne stavy systému počítal optimálny akčný zásah v danom okamihu. Z toho vyplýva, že algoritmus MPC je veľmi náročný na výpočtovú techniku, ktorá riadi systém. Experimenty boli robené na priemerne výkonnom školskom počítači. Musela byť zvýšená dĺžka vzorkovacej periódy, aby boli potrebné výpočty stihnuté medzi jednotlivými krokmi simulácie.

6. Záver

Cieľom tejto práce bolo jednak navrhnúť vhodný popis systému lineárneho žeriava, teoreticky popísať vybrané tri algoritmy – *Genetický algoritmus, algoritmus Problému okrajových hodnôt* a *algoritmus Prediktívneho riadenia* – otestovať algoritmy na počítačových simuláciách a v neposlednej rade implementovať riadenie laboratórneho modelu lineárneho žeriava pomocou každého z algoritmov. Všetky ciele práce sa podarilo splniť, pričom algoritmy boli vytvorené v programe MATLAB a ich riadenie je reprodukovateľné z priložených súborov.

Zhrnutie algoritmov je popísané v kapitole 5.8, pričom bolo rozdelené podľa bodov zadania do troch častí. *Z teoretického hľadiska* bola popísaná štruktúra algoritmov, aké časti musia obsahovať a základné vzťahy, pričom po teoretickej stránke mal genetický algoritmus veľkú výhodu oproti ostatným vďaka svojej jednoduchosti.

V ďalšej časti boli algoritmy hodnotené z *hľadiska počítačových simulácií,* kde bol zvýraznený problém genetického algoritmu, ktorým je čas kým skonvergujú výpočty k vhodnému riešeniu. Napriek tomu dosahoval takmer identické výsledky ako algoritmus BVP, ktorý však má výhodu, že riešenie je nájdené takmer okamžite. Vytvorenie algoritmu MPC bolo najzložitejšie, práve pre svoju sofistikovanú štruktúru. Riadenie počítačového modelu pomocou MPC bolo odlišné od ostatných dvoch algoritmov, najmä lebo pohyb systému bol dosť rýchlejší ako u BVP či GA.

V poslednej časti práce bolo za úlohu vytvoriť hodnotenie algoritmov z *hľadiska aplikácie na reálnu sústavu*, ktorá najprv musela byť podrobená úprave na podobu lineárneho žeriava a následne pomocou nadstavby programu MATLAB (*Parameter estimation*) boli identifikované parametre systému, ktoré bez tohto úkonu sú neznáme, ako statické či dynamické trenie, moment zotrvačnosti a pod. Ako u počítačových simulácií, GA potreboval až príliš veľa času na nájdenie vhodného riešenia (5-10 minút podľa zadania pohybu). Algoritmus BVP bol rozšírený o riadenie v uzavretej slučke pomocou PI regulátora a funkcia tohto riadenia bola veľmi dobrá. Pohyb celého systému bol jemný, hladký a okrajové podmienky systém dodržal. MPC algoritmus bol upravený do online uzavretej riadiacej slučky, čo znamená, že na základe aktuálnych stavov systému počítal v každom kroku ideálny sled akčných zásahov, pričom v každom kroku ho zároveň aplikoval do systému. Takéto riadenie je na jednej strane veľkou výhodou, algoritmus vždy vypočítal optimálny zásah do systému, avšak algoritmus MPC je známy veľkým množstvom maticových operácií. Toto malo za následok zvýšenie vzorkovacej periódy. Riadenie však aj po tejto zmene bolo veľmi dobré. Rozdielom proti BVP a GA bola vyššia rýchlosť akou premiestnenie vozíka so závažím prebehlo.

Ako posledný cieľ bolo určiť, ktorý algoritmus je najvhodnejší pre riadenie daného systému. Po vykonaní experimentov bol najvhodnejší algoritmus, ktorý zároveň ukázal, že má najlepšie výsledky práve algoritmus MPC. Prispel k tomu aj fakt, že síce anti-swing riadenie predchádza vzniku kmitov, ale nerieši prípad, keď takéto kmity vznikajú, práve algoritmus MPC v implementácii v online uzavretej slučke je schopný riešiť aj prípad vzniku náhodných kmitov počas pohybu.

Literatúra

[1] CLOSE, M. *What is a Gantry Crane? A Closer Look at the Different Types and Design* [online]. In:. [cit. 15.6.2020]. Dostupné z: https://www.mazzellacompanies.com/Resources/Blog/gantry-cranes-different-types-design

[2] LEE H., CHO S. a J. CHO, *A New Anti-Swing Control of Overhead Cranes,* IFAC Proceedings Volumes, Volume 30, Issue 13,1997,Pages 115-120,ISSN 1474-6670. [cit. 19.3.2020]. Dostupné z: https://doi.org/10.1016/S1474-6670(17)44380-1.

[3] JADLOVSKÁ, Anna. *Úvod do nelineárnych systémov*. Fakulta elektrotechniky a informatiky Technickej univerzity v Košiciach, 2016. [cit. 15.3.2020]. Dostupné z: http://matlab.fei.tuke.sk/orhs/subory/prednasky/PrednaskyNS.pdf

[4] Equations of Motion for the Inverted Pendulum (2DOF) Using Lagrange's Equations. https://www.youtube.com/watch?v=7Tvo8jXlPuk. Video [cit. 24.3.2020].

[5] QIAN, D. a J. Yi, *Hierarchical sliding mode control for under-actuated cranes, Design,analysis and simulation*, 2015. [cit. 24.3.2020]. ISBN 978-3-662-48415-9.

[6] SINGH, A. P. a H. AGRAWAL. *A fractional model predictive control design for 2-D gantry crane system*. *Journal of Engineering Science and Technology* [online]. 2018(13_7), 2224-2235. [cit. 27.3.2020]. Dostupné z: http://jestec.taylors.edu.my/Vol%2013%20issue%207%20July%202018/13_7_23.pdf?fbclid=IwAR2AGvfZGWg1Ru4JFY-5eqdjItcpU1jEDDrpbgdpVJ39ujyMdwxPT3E041k

[7] ZHAN-DONG Y. a W. XIAN-FENG, *Anti-swing and position control for bridge cranes by BVP arithmetic*, 2011 International Conference on Consumer Electronics, Communications and Networks (CECNet), XianNing, 2011, s. 1986-1989, [cit. 15.4.2020], doi: 10.1109/CECNET.2011.5768631.

[8] MALLAWAARACHCHI, V. *Introduction to Genetic Algorithms* [online]. 8.7.2017 [cit. 4.2.2020]. Dostupné z: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[9] Genetic Algorithm Tutorial - How to Code a Genetic Algorithm. https://www.youtube.com/watch?v=XP8R0yzAbdo. Video. [cit. 1.4.2020].

[10] TAKÁCS, G. a M. GULAN. *Základy prediktívneho riadenia*. Slovenská technická univerzita v Bratislave: SPEKTRUM STU, 2018. ISBN 978-80-227-4826-1. [cit. 7.4.2020].

[11] KIERZENKA, J. a L. F. SHAMPINE. *A BVP solver based on residual control and the Matlab PSE.* [online]. 2001. [cit. 20.4.2020]. Dostupné z: http://www.orcca.on.ca/TechReports/TechReports/2001/TR-01-02.pdf

[12] *Mathworks: Solving Boundary Value Problems* [online]. [cit. 25.4.2020]. Dostupné z: https://www.mathworks.com/help/matlab/math/boundary-value-problems.html

[13] DICKMANNS, E. a K. WELL. *Aproximate solution of optimal control problems using third order Hermite polynomial functions*, in Optimization Techniques. Lecture notes in computer science, Springer, 1975, (27), 158-166.

[14] Model Preditctive Control. https://www.youtube.com/watch?v=YwodGM2eoy4. Video [cit. 13.5.2020].

[15] *QUADPROG. Quadratic programming.* [online]. [cit. 15.5.2020]. Dostupné z: https://www.mathworks.com/help/optim/ug/quadprog.html

[16] *BVP4C. Solve boundary value problem – fourth order method.* [online]. [cit. 19.4.2020]. Dostupné z: https://www.mathworks.com/help/matlab/ref/bvp4c.html

[17] SISTA, S. G. *MA 417: Ordinary Differential Equations. Chapter 5.* [online]. [cit. 25.4.2020]. Dostupné z: http://www.math.iitb.ac.in/~siva/ma41707/ode6.pdf

[18] MOSTOVÉ ŽERIAVY NITRA [online]. [cit. 2020-6-19]. Dostupné z: https://www.kladkostroje-nitra.sk/mostove-zeriavy

[19] BAKHANOVIČ A. G., KUSYAK V. A., GURIN A. N., Le V. *Ovládanie paliva elektronických palivových motorov na základe PID softwareovej regulácie*. Science and Technology. 2017; 16 (1): 28-37. [cit. 21.5.2020]. Dostupné z: https://doi.org/10.21122/2227-1031-2017-16-1-28-37

[20] FRANKLIN, Gene F., J. David POWELL a Michael L. WORKMAN. *Digital control of dynamic systems.* 3rd ed. Menlo Park, Calif.: Addison-Wesley, c1998. ISBN 0201331535.

[21] ASTROM, Karl J. a Richard M. MURRAY. *Feedback systems: an introduction for scientists and engineers*. 2008. Princeton: Princeton University Press, c2008. ISBN 0691135762.

[22] KATALIN M., BOKOR J. a G. SZEDERKÉNYI. *Analysis and Control of Nonlinear Process Systems*. Springer, 2004. ISBN 978-1-85233-600-4.

Elektronické prílohy

Súčasťou diplomovej práce sú aj odovzdané prílohy v elektronickej podobe. Zložka *Prílohy.zip* obsahuje:

- Textový súbor *ReadMe.txt*, v ktorom sú bližšie informácie o prílohách.
- Textový súbor *Videa_odkaz.txt*, v ktorom sú internetové odkazy na videá vytvorené počas experimentov na laboratórnom modely.
- Zložka *Pocitacove simulacie*, ktorá obsahuje ku každému algoritmu samostatnú zložku (BVP, GA, MPC) so všetkými potrebnými súbormi (*.m, .slx, .mat, .png,* ...) pre vytvorenie rovnakých simulácií ako sú uvedené v tejto práci.
- Zložka *Parameter estimation* obsahuje súbory a výsledky z estimácie parametrov laboratórneho modelu lineárneho žeriava.
- V zložke *Porovnanie* je uvedené porovnanie výsledkov z počítačových simulácií pre rovnaké parametre systému všetkých troch algoritmov rovnako ako na Obr. 5.12.
- Posledná zložka *Prakticke_Testy* obsahuje súbory pre rekonštrukciu riadenia na laboratórnom modely. Navyše v jednotlivých zložkách algoritmov (BVP, MPC) je uvedených aj zopár nameraných riadiacich simulácií v zložke *Solution*.