ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA APLIKOVANÝCH VĚD

KATEDRA KYBERNETIKY

DIPLOMOVÁ PRÁCE

Metody rychlého prediktivního řízení pro mechatronické systémy

Plzeň, 2023 Bc. David Wébr

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. David WÉBR**
Osobní číslo:	**A21N0128P**
Studijní program:	**N3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Téma práce:	**Metody rychlého prediktivního řízení pro mechatronické systémy**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Seznamte se s metodami rychlého prediktivního řízení pro lineární systémy s kvadratickým kritériem kvality. Zaměřte se především na výpočetně úsporné solvery pro úlohy kvadratického programování a explicitní prediktivní řízení.
2. Navrhněte metodiku použití metod z bodu 1 pro třídu mechatronických systémů v úlohách řízení pohybu s požadavkem na krátkou periodu vzorkování.
3. Srovnejte simulačně navržené metody z hlediska výpočetní náročnosti a dosažené kvality řízení ve vybraných typových úlohách řízení mechatronických soustav.
4. Implementujte navržené metody v prostředí reálného času a otestujte na zvolené HW platformě.

Rozsah diplomové práce: **40-50 stránek A4**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

J.A. Rossiter, Model-Based Predictive Control; A practical approach, Taylor and Francis 2017
F. Borelli, A. Bemporad, M. Morari, Predictive Control for Linear and Hybrid systems, Cambridge 2017
M. Ondřej, Quadratic programming algorithms for fast Model-based predictive control, ČVUT Praha, 2013
H.J. Ferreau, Embedded MPC in industrial applications, ABB Corporate Research, 2016

Vedoucí diplomové práce: **Ing. Martin Goubej, Ph.D.**
Katedra kybernetiky

Datum zadání diplomové práce: **1. října 2022**
Termín odevzdání diplomové práce: **22. května 2023**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 1. října 2022

Prohlášení

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

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 22. května 2023

David Wébr

Bc. David Wébr

Poděkování

Tímto bych chtěl poděkovat panu Ing. Martinu Goubejovi, Ph.D. za skvělé vedení mé práce, cenné profesionální rady a připomínky. Bez jeho výborných znalostí a zkušeností v dané oblasti by tahle práce nebyla taková, jaká je. Dále dovolte vyjádřit díky mé rodině a partnerce za to, že mi byly oporou během celého mého studia. Zejména pak v posledních měsících kdy jsem trávil více času nad rovnicemi než společně s nimi.

Abstrakt

Práce se zabývá studiem metod rychlého prediktivního řízení a jejich praktickým použitím pro řízení systémů. V počátku práce je popsán princip činnosti prediktivní regulace ve standardním tvaru. Klíčová část u této verze MPC je rychlost výpočtu optimalizační úlohy. Z toho důvodu je v další části práce představena základní myšlenka řešení těchto optimalizačních úloh a jsou představeny i vybrané solvery, které lze použít k jejich řešení. Následuje popis principu činnosti explicitního prediktivního regulátoru, který se snaží vyhnout problému řešení optimalizační úlohy při on-line běhu prediktivního řízení. V praktické části diplomové práce se začne s popisem skutečného systému, se kterým se dále bude pracovat. Dále bude navržen prediktivní regulátor ve standardním tvaru pro řízení popsaného systému. Následně bude navržen i explicitní prediktivní regulátor, který bude porovnáván se standardní verzí prediktivního řízení. Poté proběhne vyhodnocení časové náročnosti u všech navržených prediktivních regulátorů. Poslední část práce se bude zabývat složitostí problému při použití popsaných přístupů k prediktivnímu řízení.

Klíčová slova

prediktivní řízení, MPC, regulace, regulátor, model, systém, kvadratické programování, solver, explicitní prediktivní řízení, hledání regionů, časová náročnost, paměťové nároky

Abstract

The main aim of this thesis is to study the methods of fast predictive control. At the beginning of this thesis it is described the principle of the predictive control in the standard form. In this form of predictive control it is necessary to solve the optimization problem in each time step. In the next part if this thesis there are introduced some of the solvers, which one can use for solving optimization problems. Then, the principle of explicit model predictive control is introduced. In this form of predictive control it is not necessary to solve the optimization problem in each time step. At the beginning of the practical part of this thesis will be described the real system. In the next chapters of this thesis this system will be controlled via introduced algorithms of predictive control. Firstly, it will be used the predictive controller in standard form and then it will be used explicit predictive regulator. In the next sections it will be analyzed computational complexity and memory requirement of the described predictive controllers.

Key words

regulation, model, system, predictive control, MPC, quadratic programming, solver, explicit predictive control, point location problem, computational complexity, memory requirement

Obsah

1 Úvod

Prediktivní regulace (angl. *Model-based predictive control* nebo zkráceně MPC) je moderní strategie řízení v oboru automatizace. Vychází se při tom z modelu řízené soustavy, pomocí kterého se formuluje optimalizační úloha. Konkrétně se model systému, který je chtěné řídit pomocí MPC, využije k získání predikce vývoje stavu a výstupu na konečném horizontu. Ze všech možných trajektorií systému se pak vybere ta, která vede na minimální hodnotu vhodně zvolené účelové funkce. Na základě toho se formuluje příslušná optimalizační úloha. Její konkrétní podoba se liší podle kombinace tvaru modelu a volby účelové funkce. Vyřešením této úlohy se následně získá výsledný akční zásah regulátoru, který se aplikuje do řízeného systému.

U MPC ve standardním tvaru je zapotřebí řešit v každém kroku algoritmu minimalizační úlohu, jejíž tvar se mění na základě aktuálního stavu řízeného systému. Nicméně řešení optimalizačních úloh je obecně časově náročný problém. U systémů s malou periodou vzorkování je tedy nutné mít k dispozici solver, který vyřeší optimalizaci za co nejmenší časový úsek. Je totiž potřeba, aby v každém kroku algoritmu byla doba optimalizace menší než perioda vzorkování řízeného systému. Mimo toho regulátor musí v každém kroku algoritmu řešit ještě další výpočty, které jsou nezbytné ke generování akčního zásahu. Minimalizační úloha se tedy musí vyřešit s dostatečnou časovou rezervou před koncem jedné periody vzorkování. Vzniká tedy myšlenka, jakým způsobem MPC implementovat, aby se všechny potřebné operace stačily vyřešit dostatečně včas vzhledem k periodě vzorkování řízeného systému.

Pro lepší představu zde je uveden příklad. Procesy, které je chtěné řídit v chemickém průmyslu, jsou obecně poměrně pomalé. Díky tomu je jejich perioda vzorkování velká a to například v řádech jednotkách sekund. Při snaze řídit takový proces pomocí prediktivní regulace nebude tedy výpočetní náročnost regulátoru nejspíše ten hlavní problém. V současné době totiž existují dobře odladěné solvery, které by v takových situacích s velkou periodou vzorkování měly při vhodném nastavení MPC stačit optimalizaci s dostatečnou časovou rezervou vyřešit. Nicméně dnes je stále více potřeba umět regulovat i systémy, které mají periodu vzorkování v řádech jednotek nebo dokonce desetin milisekundy. Jako příklad zde budou uvedeny kolaborativní roboti, jejichž popularita stále více roste. V tomto případě jsou periody vzorkování hodně malé. Je to z toho důvodu, aby se robot dokázal pohybovat co nejrychleji a co nejpřesněji a dokázal tak odvést co nejvíce práce. Druhým důvodem je bezpečnost. V případě nouze je tak robot schopen se zastavit téměř okamžitě dle potřeby. Při snaze řídit takové stroje pomocí prediktivní regulace je tedy nutné vhodně implementovat MPC tak, aby se úloha optimalizace a další potřebné výpočty stačily vyřešit před koncem periody vzorkování. Studiem této problematiky u prediktivní regulace se bude zabývat tato diplomová práce.

Diplomová práce bude organizována následovně. V této úvodní kapitole bude v krátkosti představen základní princip MPC ve standardním tvaru s on-line řešením kvadratického programu. Dále zde ještě bude uvedena hlavní myšlenka explicitního prediktivního regulátoru, u kterého není zapotřebí řešit optimalizační úlohu on-line. Následovat bude teoretická část. V té bude představen detailněji princip prediktivního regulátoru ve standardní verzi s počítáním minimalizační úlohy v každém kroku algoritmu. Konkrétně bude rozebráno, jak se tvoří predikce budoucího chování systému, dále problém horizontu predikcí a řízení a v neposlední řadě, jak se tvoří účelová funkce, jejíž minimalizací se získá optimální akční zásah. Další část práce bude věnována kvadratickému programování. Jeho znalost je nezbytná pro detailní pochopení prediktivního řízení. Zde se nejdříve definuje co je to kvadratický

program a co vyjadřují KKT podmínky optimality. Současně budou představeny i některé algoritmy a solvery, které se k řešení úloh kvadratického programování používají. Poslední část teoretické části diplomové práce bude věnována explicitnímu prediktivnímu řízení. Zde bude uvedeno, jak se tento přístup k prediktivní regulaci liší od toho klasického. Následovat bude základní algoritmus explicitního prediktivního regulátoru. To obnáší představení KKT podmínek pro tento typ úlohy, dále jak probíhá hledání všech regionů ve stavovém prostoru a jako poslední jak se řeší tzv. *point location problem*. Poté se již přistoupí k praktické části diplomové práce. Začne se s popisem skutečného systému, se kterým se bude v praktické části pracovat. Konkrétně se jedná o mechatronický stend pohonu s pružnou zátěží. Budou zde ukázány jeho nejdůležitější vlastnosti a charakteristiky. Další kapitola bude věnována řízení popsaného systému pomocí prediktivního regulátoru ve standardním tvaru s on-line solverem. Nejdříve proběhne návrh tohoto regulátoru. Posléze bude jeho chování odzkoušeno při řízení modelu systému a následně i při řízení skutečného systému. Další část práce bude věnována explicitnímu prediktivnímu regulátoru. Snaha bude u něj docílit totožného chování jako u standardní verze prediktivního regulátoru. Ve výsledku pak bude dobře možné oba tyto přístupy k prediktivní regulaci porovnat. Nejdříve proběhne návrh explicitního regulátoru a posléze opět jeho otestování při řízení modelu systému a následně i skutečného systému. Práce je zaměřena na metody rychlého prediktivního řízení, z toho důvodu bude další kapitola věnována analýze časové náročnosti všech navržených prediktivních regulátorů. Díky tomu pak bude možné určit, který z regulátorů dokáže nalézt výsledný akční zásah za nejkratší dobu. Následující část práce bude věnována složitosti problému prediktivní regulace. Při aplikaci regulátoru na cílový hardware je nutné sledovat nejen samotnou rychlost výpočtů, ale i paměťové nároky regulátoru. Je to proto, že každý hardware má omezenou kapacitu paměti. Může se tedy klidně stát, že navržený prediktivní regulátor bude z hlediska časové náročnosti dosahovat velice příznivých výsledků, ale z důvodu jeho velkých paměťových nároků ho nebude možné v praxi použít.

1.1 Prediktivní řízení ve standardním tvaru

Jak již název napovídá, tak při prediktivní regulaci se vychází z modelu řízeného systému. Pro návrh MPC je tedy nezbytné tímto modelem disponovat a to v jeho diskrétní podobě. Nejčastěji se při tom používá stavový popis. Současně je nutné znát i odpovídající periodu vzorkování řízeného systému.

Jako první se v algoritmu MPC predikuje chování systému a to pro určený časový úsek od současnosti do budoucnosti. K tomu se využije právě model řízeného systému. Ze všech přípustných trajektorií systému se pak určí ta, která vede na minimální hodnotu zvolené účelové funkce. Ta může být obecně volena libovolná. V této práci se bude pracovat s účelovou funkcí v kvadratické formě. Na základě uvedených skutečností vyjde optimalizační úloha ve tvaru kvadratického programu - jedná se tedy o speciální případ popsaného problému, kdy se zvolila kvadratická účelová funkce a použil se uvedený tvar modelu. Vyřešením kvadratického programu se získá sekvence optimálních akčních zásahů regulátoru pro několik dalších kroků do budoucnosti. První hodnota z nalezené sekvence se posléze použije jako vstup do řízené soustavy. Řízený systém se tak dostane do nového stavu a celý algoritmus se opakuje.

Z popsaného algoritmu je zřejmé, že v každém kroku je nutné on-line vyřešit optimalizační úlohu ve formě kvadratického programování [1]. Existuje celá řada solverů, které tento problém řeší. Mezi ně patří například solver `quadprog` (dokumentaci lze nalézt na *https://www.mathworks.com/help/optim/ug/quadprog.html*), `qp-OASES` [2], `FiOrdOs` [3] nebo

`MOSEK` [4]. Solvery se obecně mohou lišit přesností nalezeného výsledku ale také rychlostí konvergence k řešení, s čímž je spojená i časová náročnost. Každému solveru tedy může trvat jinak dlouho, než řešení optimalizační úlohy najde.

1.2 Explicitní prediktivní regulátor

V předchozím odstavci byl popsán algoritmus prediktivního regulátoru, kdy je MPC implementován ve standardním tvaru s on-line solverem. Řešení optimalizační úlohy se tam musí hledat v každém kroku algoritmu (on-line) na základě aktuálního stavu systému. Explicitní přístup k prediktivnímu řízení se snaží tomuto problému vyhnout a tím zkrátit čas potřebný k výpočtu optimálního akčního zásahu. Základní princip explicitního prediktivního řízení spočívá v následující myšlence. Namísto řešení optimalizačního problému on-line pro aktuální jeden stav systému je zde cílem vyřešit optimalizační úlohu předem (off-line) pro všechny přípustné stavy systému. Díky tomu je výsledný vztah pro optimální akční zásah dán explicitně jako funkce aktuálního stavu systému [5].

1.3 Přehled použitých symbolů

- $\mathbb{R}$ - obor reálných čísel, $\mathbb{N}$ - obor přirozených čísel
- tučná velká písmena od $\mathbf{A}$ do $\mathbf{Z}$ - matice odpovídající velikosti
- tučná malá písmena od $\mathbf{a}$ do $\mathbf{z}$ - vektory odpovídající velikosti
- $\mathbf{A}^{\mathrm{T}}$ $(\mathbf{a})^{\mathrm{T}}$ - transpozice matice (vektoru)
- $\mathbf{1}$ ($\mathbf{0}$) - jednotková (nulová) matice odpovídající velikosti
- $\mathbf{I}$ - matice odpovídající velikosti s jedničkami na diagonále (identická matice)

2 Prediktivní řízení

Prediktivní řízení se v angličtině označuje termínem *model-based predictive control*, což by se dalo doslovně přeložit jako *prediktivní řízení na základě modelu*. Oním modelem je myšlen model řízené soustavy. Pro implementaci prediktivního regulátoru je jeho znalost nezbytná. Tento model řízeného systému může být ve formě přenosové funkce nebo ve formě stavového popisu [1]. Častěji se ale vychází ze stavové reprezentace systému, což bude použito i v této diplomové práci. Algoritmus MPC je již z principu diskrétní a i model řízeného systému je nutné mít v jeho diskrétní podobě. Vzhledem k tomu je nezbytná nejen znalost matic **A**, **B**, **C**, **D** z diskrétní stavové reprezentace modelu systému ale rovněž i jeho perioda vzorkování. Ve výsledku vyjde i samotný prediktivní regulátor v diskrétním tvaru, přičemž bude vzorkován stejnou periodou jako řízený systém.

Hlavním tématem této kapitoly bude popsat princip fungování prediktivního regulátoru ve standardním tvaru. Při klasické implementaci MPC dochází k řešení minimalizační úlohy v každém kroku algoritmu (tzv. on-line varianta MPC). Tvar této optimalizační úlohy závisí na stavu systému. Konkrétně se musí hledat řešení úlohy kvadratického programování. K tomu existuje celá řada solverů, z nichž jsou některé implementovány právě pro účely prediktivního řízení [6].

Tato kapitola bude obsahově rozdělena následovně. Nejdříve bude uveden základní algoritmus MPC. Dále bude ukázáno, jak se počítají predikční matice, co vyjadřuje horizont predikcí a řízení a dále co obnáší technika *move blocking*. Následující podkapitola zabývající se popisem MPC ve standardním tvaru bude pojednávat o možných omezeních, která lze klást prostřednictvím vhodné formulace optimalizační úlohy na regulační smyčku. Dále bude ukázáno, jak se tvoří účelová funkce a jak probíhá optimalizace, což je jedna z nejdůležitějších částí celé prediktivní regulace. Na závěr této kapitoly bude ukázáno, jak implementovat algoritmus MPC tak, aby bylo při regulaci dosaženo nulové odchylky v ustáleném stavu mezi referenční hodnotou a výstupem systému.

Ve všech následujících podkapitolách bude uvedena pouze základní myšlenka. Detailní popis prediktivního regulátoru s on-line řešením optimalizační úlohy lze nalézt ve zdrojích [6] a [7].

2.1 Algoritmus MPC

Prvním úkolem v algoritmu MPC je definovat budoucí chování řízeného systému a to pro několik následujících časových okamžiků daných uvažovanou periodou vzorkování. Použijí se k tomu matice z diskrétní stavové reprezentace modelu řízené soustavy. Konkrétně se z nich vypočítají tzv. *predikční matice*. Pomocí nich a aktuálního stavu systému již lze predikovat, jak se systém bude v budoucnosti chovat.

Důležitým faktorem u prediktivní regulace je to, jak daleko do budoucnosti se bude stav řízeného systému odhadovat. To se definuje pomocí hodnoty tzv. *predikčního horizontu*. Ten se obecně označuje jako n_p a určuje pro kolik kroků od současnosti do budoucnosti se bude predikce provádět. Jedná se o jeden z klíčových parametrů prediktivního regulátoru a má i velký vliv na jeho celkovou složitost. Dalším důležitým parametrem prediktivní regulace je tzv. *horizont řízení* s označením n_c. Jeho hodnota určuje, pro kolik kroků od současnosti do budoucnosti se má počítat optimální akční zásah.

V okamžiku, kdy jsou známy predikční matice a hodnoty horizontu predikcí a horizontu

řízení, tak lze přistoupit k počítání optimálního akčního zásahu. Tento výpočet je realizován pomocí tzv. *účelové funkce*, jejíž tvar je dán mimo jiné i predikčními maticemi. Hodnotu účelové funkce je při tom chtěné minimalizovat a to přes vektor hodnot akčních zásahů o délce n_c. Jinými slovy se hledá takový vektor akčních zásahů, pro který bude hodnota účelové funkce nejmenší. Konkrétně se jedná o úlohu kvadratického programování (tzn. účelová funkce je kvadratická). Při řešení tohoto problému lze klást omezení na samotné akční zásahy a také na stavy a výstupy systému, což je jedna z velkých výhod prediktivní regulace. Výsledkem optimalizace je vektor s optimálními akčními zásahy pro následujících n_c kroků. Do řízené soustavy se ale aplikuje pouze první z nich. Tím se systém dostane do nového stavu, čímž je ukončen jeden krok algoritmu.

Nový stav ovšem nemusí být shodný s tím predikovaným. To může být dáno například vlivem poruchy. Z toho důvodu se pro aktuální krok opět provede optimalizace s aktuálním stavem systému. Celý uvedený proces se neustále opakuje a to v každém časovém okamžiku, který je dán periodou vzorkování.

2.2 Predikční matice

Odhad budoucího chování řízeného systému je jedna z klíčových úloh celého prediktivního řízení. Přesnost modelu systému vůči skutečnosti musí být co největší. I malá odchylka mezi získaným modelem systému a reálnou řízenou soustavou může vést na špatný odhad chování systému. Na základě toho nebude při minimalizaci nalezen optimální akční zásah, což může zapříčinit méně kvalitní regulaci skutečné soustavy nebo v nejhorším případě i nestabilní regulační smyčku.

Diskrétní řízená soustava s n stavy, m vstupy a p výstupy bude reprezentována jejím modelem ve stavové reprezentaci ve tvaru

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \tag{1}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k, \tag{2}$$

kde $\mathbf{x} \in \mathbb{R}^{n\times 1}$ označuje stavy systému, $\mathbf{u} \in \mathbb{R}^{m\times 1}$ vstupy a $\mathbf{y} \in \mathbb{R}^{p\times 1}$ výstupy řízené soustavy, dále matice dynamiky $\mathbf{A} \in \mathbb{R}^{n\times n}$, matice vstupů $\mathbf{B} \in \mathbb{R}^{n\times m}$, výstupní matice $\mathbf{C} \in \mathbb{R}^{p\times n}$ a vstupně-výstupní matice $\mathbf{D} \in \mathbb{R}^{p\times m}$.

Využitím uvedeného vztahu lze nyní odvodit vývoj stavů v následujícím kroku $\mathbf{x}_{k+2}$ a výstupů v následujícím kroku $\mathbf{y}_{k+1}$. Po dosazení vyjde

$$\mathbf{x}_{k+2} = \mathbf{A}\mathbf{x}_{k+1} + \mathbf{B}\mathbf{u}_{k+1} = \mathbf{A}^\mathbf{2}\mathbf{x}_k + \mathbf{A}\mathbf{B}\mathbf{u}_k + \mathbf{B}\mathbf{u}_{k+1}, \tag{3}$$

$$\mathbf{y}_{k+1} = \mathbf{C}\mathbf{x}_{k+1} + \mathbf{D}\mathbf{u}_{k+1} = \mathbf{C}\mathbf{A}\mathbf{x}_k + \mathbf{C}\mathbf{B}\mathbf{u}_k + \mathbf{D}\mathbf{u}_{k+1}. \tag{4}$$

Analogickým způsobem lze postupovat i pro odhad vývoje stavů a výstupů v následujících krocích. Pro kolik kroků od současnosti do budoucnosti se bude predikce počítat určuje hodnota horizontu predikcí n_p. Využitím popsané myšlenky vývoje stavů a výstupů lze odhadnout chování systému až do kroku n_p od současnosti. Pomocí matic to lze zapsat následovně

$$\underbrace{\begin{bmatrix}\mathbf{x}_{k+1}\\ \mathbf{x}_{k+2}\\ \vdots\\ \mathbf{x}_{k+n_p}\end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix}\mathbf{A}\\ \mathbf{A}^2\\ \vdots\\ \mathbf{A}^{n_p}\end{bmatrix}}_{\mathbf{T}} \mathbf{x}_k + \underbrace{\begin{bmatrix}\mathbf{B} & \mathbf{0} & \cdots\\ \mathbf{AB} & \mathbf{B} & \cdots\\ \vdots & \vdots & \ddots\\ \mathbf{A}^{n_p-1}\mathbf{B} & \mathbf{A}^{n_p-2}\mathbf{B} & \cdots\end{bmatrix}}_{\mathbf{S}_u} \underbrace{\begin{bmatrix}\mathbf{u}_k\\ \mathbf{u}_{k+1}\\ \vdots\\ \mathbf{u}_{k+n_p-1}\end{bmatrix}}_{\mathbf{U}_{np}}. \tag{5}$$

Pro zjednodušení zápisu se využije naznačené substituce. Vývoj stavů řízené soustavy od současnosti až do kroku n_p lze zapsat jako

$$\mathbf{X} = \mathbf{T}\mathbf{x}_k + \mathbf{S}_u\mathbf{U}_{np}. \tag{6}$$

Analogicky lze postupovat i pro odhad vývoje výstupů řízené soustavy

$$\underbrace{\begin{bmatrix}\mathbf{y}_k\\ \mathbf{y}_{k+1}\\ \mathbf{y}_{k+2}\\ \vdots\\ \mathbf{y}_{k+n_p-1}\end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix}\mathbf{C}\\ \mathbf{CA}\\ \mathbf{CA}^2\\ \vdots\\ \mathbf{CA}^{n_p-1}\end{bmatrix}}_{\mathbf{T}_c} \mathbf{x}_k + \underbrace{\begin{bmatrix}\mathbf{D} & \mathbf{0} & \mathbf{0} & \cdots\\ \mathbf{CB} & \mathbf{D} & \mathbf{0} & \cdots\\ \mathbf{CAB} & \mathbf{CB} & \mathbf{D} & \cdots\\ \vdots & \vdots & \vdots & \ddots\\ \mathbf{CA}^{n_p-2}\mathbf{B} & \mathbf{CA}^{n_p-3}\mathbf{B} & \mathbf{CA}^{n_p-4}\mathbf{B} & \cdots\end{bmatrix}}_{\mathbf{S}_{cu}} \underbrace{\begin{bmatrix}\mathbf{u}_k\\ \mathbf{u}_{k+1}\\ \vdots\\ \mathbf{u}_{k+n_p-1}\end{bmatrix}}_{\mathbf{U}_{np}}. \tag{7}$$

I v tomto případě se pro úsporu zápisu zavede substituce. Predikovaný vývoj výstupů systémů lze vyjádřit tímto způsobem

$$\mathbf{Y} = \mathbf{T}_c\mathbf{x}_k + \mathbf{S}_{cu}\mathbf{U}_{np}. \tag{8}$$

Matice $\mathbf{T}$ a $\mathbf{S}_u$ jsou označovány jako *stavové predikční matice*. Matice odpovídající výstupu systému, $\mathbf{T}_c$ a $\mathbf{S}_{cu}$, jsou nazývány *výstupní predikční matice*. Pro získání odhadu vývoje stavů a výstupů je tedy potřeba disponovat stavovým modelem řízené soustavy, stavem řízeného systému $\mathbf{x}_k$ v aktuálním kroku a vstupem do systému v následujících n_p krocích $\mathbf{U}_{np}$.

2.3 Horizont predikcí a řízení

Jak již bylo řečeno v předchozích podkapitolách, tak horizont predikcí se značí n_p a platí $n_p \in \mathbb{N}$. Jedná se o jeden z klíčových vstupních parametrů úlohy prediktivního řízení. Určuje, kolik kroků od současnosti do budoucnosti se bude chování systému odhadovat. Jinými slovy hodnota predikčního horizontu vyjadřuje, jak daleko do budoucnosti má prediktivní regulátor odhadovat hodnoty stavů a výstupů řízené soustavy. Na základě těchto odhadů se budou posléze vypočítávat optimální akční zásahy. Pokud se predikční horizont n_p vynásobí se vzorkovací periodou T_s řízeného systému, tak vyjde, do jaké doby v budoucnosti bude od aktuálního okamžiku počítán odhad chování systému. Bude-li například řízený systém vzorkován periodou $T_s = 1\ s$ a hodnota predikčního horizontu bude $n_p = 5$, tak se bude predikovat chování řízené soustavy až do času 5 sekund do budoucnosti od aktuálního okamžiku.

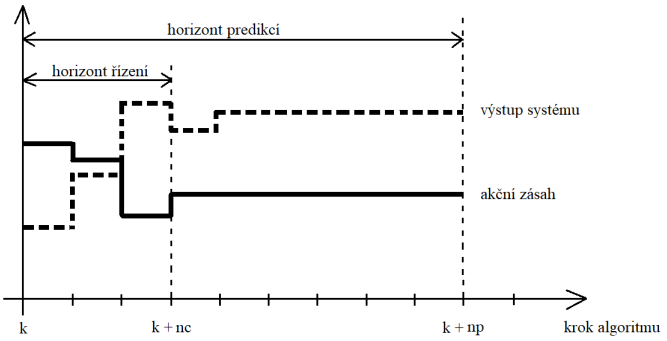
Klíčovým vstupním parametrem prediktivní regulace je také horizont řízení, který se obecně označuje n_c a rovněž platí $n_c \in \mathbb{N}$. Velikost horizontu řízení vyjadřuje, pro kolik kroků do budoucnosti od aktuálního okamžiku se mají hledat optimální akční zásahy. Hodnotu n_c je nutné volit menší nebo rovnu hodnotě horizontu predikcí n_p. Lze odvodit, že v opačné situaci, tj. $n_c > n_p$, by nebylo možné optimální akční zásah vypočítat, jelikož by nebylo možné získat predikované hodnoty stavů a výstupů řízené soustavy. Hodnota horizontu řízení se z pravidla volí o něco menší než je velikost horizontu predikcí. Je to proto, že na vývoj řízení v následujících okamžicích má největší vliv pouze několik prvních hodnot nalezených optimálních akčních zásahů. Vypočtené hodnoty řízení pro vzdálenější kroky v budoucnosti totiž budou s největší pravděpodobností v následujících krocích vypočteny rozdílně. Často se hodnota horizontu řízení n_c volí v rozmezí od 10 % do 20 % hodnoty predikčního horizontu n_p.

Pro odhad chování řízeného systému je ale nutné, dle rovnic (6) a (8), mít k dispozici hodnoty akčního zásahu pro celý predikční horizont. To se obecně řeší tak, že od kroku n_c až do kroku n_p setrvává hodnota akčního zásahu na stejné hodnotě. Akční zásahy v těchto krocích odpovídají tomu poslednímu vypočtenému, tj. tomu z kroku n_c od současnosti. Existuje ještě jeden přístup k řešení tohoto problému, a to že se hodnoty akčních zásahů od kroku n_c do kroku n_p nastaví na hodnotu 0. Této problematice se bude podrobněji věnovat následující podkapitola s názvem Technika *move blocking*.

Po minimalizaci účelové funkce (tj. vyřešení optimalizační úlohy), a tím i nalezení optimálních akčních zásahů pro následujících n_c kroků, se do řízeného systému aplikuje pouze první z nich. Hodnoty odhadovaných stavů, výstupů a i optimálních akčních zásahů od kroku $k+1$ již nejsou zapotřebí a z algoritmu regulace jsou vypuštěny. Řízený systém se po aplikaci prvního optimálního akčního zásahu dostane do nového stavu a je opět odhadováno chování soustavy až do kroku n_p od aktuálního okamžiku. Interval predikcí chování se tak současně se systémem dostal o jeden krok dopředu. Tento princip se nazývá *receding horizon* [1].

S narůstající velikostí predikčního horizontu n_p a horizontu řízení n_c lze obecně při vhodném nastavení ostatních parametrů dosáhnout kvalitnější regulace. Nicméně s každou hodnotou horizontu predikcí a řízení navíc roste výpočetní náročnost celé úlohy, jelikož se musí odhadovat chování systému dál v budoucnosti a zvyšuje se i velikost kvadratického programu. Hodnoty horizontů je tedy vhodné vždy volit s ohledem na charakter řízeného systému a na výkonnosti použitého hardwaru, na kterém by byl MPC regulátor realizován.

Problematiku horizontu predikce a řízení ilustruje tento obrázek.



Obrázek 1: Ilustrace predikčního horizontu n_p a horizontu řízení n_c [7]

2.4 Technika move blocking

Princip *move blocking* se obecně zabývá tím, že při minimalizaci účelové funkce se hledají optimální akční zásahy pouze do kroku n_c od aktuálního okamžiku. Od kroku n_c do kroku n_p mají hodnoty odhadovaných akčních zásahů do řízené soustavy konstantní velikost. Jejich hodnota odpovídá buď poslednímu akčnímu zásahu (tj. vypočtenému akčnímu zásahu z kroku n_c) nebo je zvolena nulová. V této práci se uvažoval druhý případ a tedy predikované akční zásahy od kroku n_c do kroku n_p budou uvažovány nulové. Důvody, proč se n_c volí v rozmezí 10 % až 20 % z hodnoty n_p, byly popsány v předchozí podkapitole. Hlavním argumentem bylo to, že se tím snižuje výpočetní náročnost výsledného MPC regulátoru.

Technika move blocking se do algoritmu MPC aplikuje následujícím způsobem. Cílem je, aby akční zásahy od kroku n_c do kroku n_p byly 0. Maticově to lze zapsat

$$\mathbf{U}_{np} = \begin{bmatrix} \mathbf{u}_k \\ \vdots \\ \mathbf{u}_{k+n_c-1} \\ \vdots \\ \mathbf{u}_{k+n_p-1} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{I} \\ & \mathbf{0} & \end{bmatrix}}_{\mathbf{L}} \underbrace{\begin{bmatrix} \mathbf{u}_k \\ \vdots \\ \mathbf{u}_{k+n_c-1} \end{bmatrix}}_{\mathbf{U}_{nc}}, \tag{9}$$

kde $\mathbf{L} \in \mathbb{R}^{n_p m \times n_c m}$. Hodnota m odpovídá počtu vstupů řízeného systému.

Vektor $\mathbf{U}_{nc}$ vyjde jako řešení optimalizační úlohy. Jinými slovy to znamená, že hodnota účelové funkce je pro tuto hodnotu $\mathbf{U}_{nc}$ nejmenší. Nicméně pro odhad budoucího chování řízeného systému jsou potřeba akční zásahy až do kroku n_p ($\mathbf{U}_{np}$) od aktuálního okamžiku. Ty lze jednoduše získat pomocí rovnice

$$\mathbf{U}_{np} = \mathbf{L}\mathbf{U}_{nc}. \tag{10}$$

Díky tomu lze stavovou predikční matici $\mathbf{S}_u$ a výstupní predikční matici $\mathbf{S}_{cu}$ upravit do nového tvaru, kdy za vstup bude uvažována pouze sekvence řízení $\mathbf{U}_{nc}$. Bude tedy platit $\mathbf{S}_{uL} = \mathbf{S}_u\mathbf{L}$ pro stavy a $\mathbf{S}_{cuL} = \mathbf{S}_{cu}\mathbf{L}$ pro výstupy. Upravené vztahy pro výpočet odhadu chování řízené soustavy tedy vyjdou jako

$$\mathbf{X} = \mathbf{T}\mathbf{x}_k + \mathbf{S}_{uL}\mathbf{U}_{nc}, \tag{11}$$

$$\mathbf{Y} = \mathbf{T}_c\mathbf{x}_k + \mathbf{S}_{cuL}\mathbf{U}_{nc}. \tag{12}$$

2.5 Omezení

Jedna z hlavních výhod MPC regulátoru je to, že už z jeho principu lze klást omezení na důležité veličiny v regulační smyčce. Poměrně jednoduše je tak možné vhodnou implementací algoritmu docílit toho, že stavy řízeného systému, regulovaná veličina a hlavně generovaný akční zásah budou nabývat pouze povolených hodnot určených návrhářem.

Při prediktivním řízení lze samotná omezení rozčlenit do dvou kategorií. První z nich

jsou označována jako tzv. **tvrdá omezení** a druhá se nazývají **měkká omezení**. Pro tvrdá omezení platí, že je není možné v žádné situaci porušit. Díky tomu se používají především k omezení velikosti akčních zásahů, které jsou v praxi vždy limitovány. Naopak měkká omezení lze za jistých podmínek překročit. Jejich porušení se ale projeví nárůstem hodnoty účelové funkce.

Detailnější informace o tom, jak zahrnout tvrdá nebo měkká omezení do algoritmu MPC, lze najít ve zdroji [7].

2.6 Účelová funkce a optimalizace

Účelová funkce (angl. *cost function*) hraje v prediktivním řízením zásadní roli. Dalo by se říci, že na tvaru účelové funkce je vlastně celá prediktivní regulace postavena. Prostřednictvím číselné skalární hodnoty určuje v každém kroku algoritmu kvalitu predikce budoucího chování řízeného systému.

Účelová funkce je tvořena vektorem akčních zásahů (tj. optimalizační proměnná), predikčními maticemi, aktuálním stavem systému (tj. vektorem stavu) a popřípadě ještě dalšími maticemi, které definují cíl prediktivního řízení (např. problém regulace nebo sledování referenčního signálu). Při prediktivní regulaci je cílem hodnotu účelové funkce minimalizovat a to přes vektor akčních zásahů, který má dimenzi n_c. Takto definovaný problém se označuje termínem *optimalizační úloha* a na té je celé prediktivní řízení založeno. Konkrétně tato optimalizační úloha vychází v podobě kvadratického programu (tzn. účelová funkce je kvadratická vzhledem k vektoru akčních zásahů). Dosažení minimální hodnoty účelové funkce by mělo vést k tomu, že se při řízení v následujících krocích bude výstup řízeného systému blížit k požadované hodnotě (např. hodnota regulované veličiny k referenční hodnotě). Zároveň by mělo nalezení minima účelové funkce v každém kroku algoritmu zaručit to, že rozdíl akčních zásahů ve dvou po sobě následujících krocích bude co nejmenší. To by mělo vést k "hladšímu"průběhu akčních zásahů generovaných regulátorem.

Po nalezení optimálních n_c akčních zásahů (resp. po vyřešení minimalizace účelové funkce) se do řízeného systému aplikuje pouze první z nich. Vlivem toho se systém dostane do nového stavu. Ten by měl odpovídat tomu predikovanému, ale například vlivem neměřitelné vstupní poruchy tomu tak být nemusí. Proto je opět proces optimalizace proveden znovu s novým vektorem stavu. To se opakuje v každém kroku algoritmu.

Jak již bylo uvedeno, tak úloha minimalizace účelové funkce je při prediktivním řízení nejvíce výpočetně náročný proces. Vzhledem k tomu se ukazuje jako vhodné volit velikost horizontu predikcí a horizontu řízení v rozumných mezích. Je to z toho důvodu, že hodnoty těchto parametrů mají zásadní vliv na velikost a tím pádem i výpočetní složitost celé úlohy. Čím větší horizonty predikcí a řízení budou, tím bude optimalizace více výpočetně náročnější.

Standardní tvar účelové funkce pro kvadratické programování je

$$J_{QP} = \frac{1}{2}\mathbf{U}_{nc}^{\mathrm{T}}\mathbf{H}\mathbf{U}_{nc} + \mathbf{U}_{nc}^{\mathrm{T}}\mathbf{G}, \tag{13}$$

kde $J_{QP} \in \mathbb{R}$ je funkční hodnota. V uvažovaném případě pro prediktivní regulátor mají proměnné ze vztahu (13) následující význam. $\mathbf{U}_{nc}$ reprezentuje vektor optimálních akčních zásahů, $\mathbf{H}$ označuje symetrickou a pozitivně definitní Hessovu matici a $\mathbf{G}$ je gradientní vektor. Matice $\mathbf{H}$ a $\mathbf{G}$ jsou dohromady nazývány matice optimalizace. Při prediktivním

řízení vychází pro nalezení optimálních akčních zásahů úloha minimalizace v následujícím tvaru

$$\min_{\mathbf{U}_{nc}} \frac{1}{2}\mathbf{U}_{nc}^{\mathrm{T}}\mathbf{H}\mathbf{U}_{nc} + \mathbf{U}_{nc}^{\mathrm{T}}\mathbf{G}. \tag{14}$$

Při hledání podoby matic optimalizace lze vyjít z obecného počátečního tvaru účelové funkce v podobě

$$J = \sum_{k=1}^{n_p} \left(\mathbf{x}_k^{\mathrm{T}}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_{k-1}^{\mathrm{T}}\mathbf{R}\mathbf{u}_{k-1}\right), \tag{15}$$

což lze rozepsat jako

$$J = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots \mathbf{x}_{n_p}\end{bmatrix} \underbrace{\begin{bmatrix}\mathbf{Q} & & & \\ & \mathbf{Q} & & \\ & & \ddots & \\ & & & \mathbf{Q}\end{bmatrix}}_{\overline{\mathbf{Q}}=\overline{\mathbf{Q}}^{\mathrm{T}}} \begin{bmatrix}\mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{n_p}\end{bmatrix} + \begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_{n_p-1}\end{bmatrix} \underbrace{\begin{bmatrix}\mathbf{R} & & & \\ & \mathbf{R} & & \\ & & \ddots & \\ & & & \mathbf{R}\end{bmatrix}}_{\overline{\mathbf{R}}=\overline{\mathbf{R}}^{\mathrm{T}}} \begin{bmatrix}\mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{n_p-1}\end{bmatrix}. \tag{16}$$

Účelovou funkci lze nyní po zavedení dříve uvedených substitucí zapsat takto

$$J = \mathbf{X}^{\mathrm{T}}\overline{\mathbf{Q}}\mathbf{X} + \mathbf{U}_{np}^{\mathrm{T}}\overline{\mathbf{R}}\mathbf{U}_{np}. \tag{17}$$

V tuto chvíli lze využít výše uvedených vztahů a pomocí dalších jednoduchých úprav lze dostat

$$J = \mathbf{U}_{np}^{\mathrm{T}}\left(\mathbf{S}_u^{\mathrm{T}}\overline{\mathbf{Q}}\mathbf{S}_u + \overline{\mathbf{R}}\right)\mathbf{U}_{np} + \mathbf{U}_{np}^{\mathrm{T}}2\mathbf{S}_u^{\mathrm{T}}\overline{\mathbf{Q}}\mathbf{T}\mathbf{x}_k, \tag{18}$$

Dále po porovnání tohoto vztahu s rovnicí (13) vyjdou matice optimalizace ve tvaru

$$\mathbf{H} = 2(\mathbf{S}_u^{\mathrm{T}}\overline{\mathbf{Q}}\mathbf{S}_u + \overline{\mathbf{R}}), \tag{19}$$

$$\mathbf{G} = 2\mathbf{S}_u^{\mathrm{T}}\overline{\mathbf{Q}}\mathbf{T}\mathbf{x}_k. \tag{20}$$

Matice optimalizace v uvedené podobě řeší úkol regulace, tj. převedení všech stavových proměnných do rovnovážného stavu. Nicméně častěji při regulaci mechatronických systémů je potřeba mít k dispozici algoritmus pro zaručení nulové odchylky v nulovém stavu. Jinými slovy pomocí prediktivního řízení je potřeba sledovat výstupem systému referenční signál a i odstranit vliv neměřitelné vstupní poruchy. Jak sestavit matice optimalizace pro tento účel, tím se bude zabývat následující podkapitola.

2.7 Algoritmus MPC pro nulovou odchylku v ustáleném stavu

Pro dosažení trvale nulové odchylky regulované veličiny od konstantního referenčního signálu je obecně při regulaci zapotřebí, aby akční zásahy v ustáleném stavu nabývaly konstantní hodnoty. To bude v případě prediktivní regulace řešeno úpravou účelové funkce.

Pro nulovou odchylku v ustáleném stavu je potřeba, aby hodnota účelové funkce setrvala na hodnotě nula. Ovšem v ustáleném stavu obecně nemusí být hodnoty akčních zásahů nulové (systémy bez astatismu). Vzhledem k této myšlence dojde v účelové funkci k následující úpravě. Namísto samotných hodnot akčních zásahů $\mathbf{u}_i$ se bude v každém kroku penalizovat hodnota rozdílu akčních zásahů ve dvou po sobě jdoucích krocích. Zavede se značení $\Delta\mathbf{u}_i = \mathbf{u}_i - \mathbf{u}_{i-1}$. Samotná optimalizace bude ale realizována přes vektor řízení $\mathbf{U}_{nc}$. Díky tomu lze stále poměrně jednoduše aplikovat tvrdá omezení na generovaný akční zásah. V případě optimalizace přes přírůstky $\Delta\mathbf{u}$ by bylo začlenění omezení přímo na hodnoty řízení $\mathbf{u}_i$ o mnoho komplikovanější.

Pro aplikaci algoritmu pro zaručení nulové odchylky v ustáleném stavu je nutné definovat nový, modifikovaný model řízené soustavy, na základě kterého budou počítány predikční matice a celá prediktivní regulace se tedy od něj bude odvíjet. Stavový model je nutné doplnit o stavy definující reference pro každý výstup systému. Dále je ještě nutné vektor stavu rozšířit o stavy, které budou reprezentovat aplikované hodnoty řízení do systému z posledního kroku algoritmu, tedy $\mathbf{u}_{-1}$. Samotný vektor stavu se rozšíří nejen o stavy pro referenční signály $\mathbf{y}_{r,k} \in \mathbb{R}^{p\times 1}$, ale i o aplikované hodnoty akčních zásahů z předchozího kroku $\mathbf{u}_{-1} \in \mathbb{R}^{m\times 1}$. V algoritmu to bude realizováno tak, že se aplikované hodnoty akčních zásahů z minulého kroku v aktuálním kroku přímo dosadí do vektoru stavu. Modifikovaný model řízené soustavy ve stavové reprezentaci, na základě kterého se bude odhadovat chování systému, tak ve finální podobě vypadá následovně

$$\underbrace{\begin{bmatrix}\mathbf{x}_{k+1}\\ \mathbf{y}_{r,k+1}\\ \mathbf{u}_{-1}\end{bmatrix}}_{\widetilde{\mathbf{x}}_{k+1}} = \underbrace{\begin{bmatrix}\mathbf{A} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{I}\end{bmatrix}}_{\widetilde{\mathbf{A}}} \underbrace{\begin{bmatrix}\mathbf{x}_{k}\\ \mathbf{y}_{r,k}\\ \mathbf{u}_{-1}\end{bmatrix}}_{\widetilde{\mathbf{x}}_{k}} + \underbrace{\begin{bmatrix}\mathbf{B}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}}_{\widetilde{\mathbf{B}}} \mathbf{u}_k. \tag{21}$$

Výstupní rovnici z původního stavového modelu je vzhledem k zavedeným úpravám nutné definovat jako

$$\mathbf{y}_k = \underbrace{\begin{bmatrix}\mathbf{C} & \mathbf{0} & \mathbf{0}\end{bmatrix}}_{\widetilde{\mathbf{C}}} \widetilde{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k. \tag{22}$$

Rozdíl mezi referencemi a aktuálním výstupem systému bude počítán pomocí vztahu

$$\mathbf{e}_k = \underbrace{\begin{bmatrix}\mathbf{C} & -\mathbf{I} & \mathbf{0}\end{bmatrix}}_{\widetilde{\mathbf{C}}_e} \widetilde{\mathbf{x}}_k + \mathbf{D}\mathbf{u}_k. \tag{23}$$

Z takto definovaného rozdílu se bude vycházet při určení tvaru účelové funkce, jak bude ukázáno později.

Pro dosažení nulové odchylky v ustáleném stavu je zapotřebí, aby byly predikční matice počítány pomocí matic modifikovaného stavového modelu, tj. $\widetilde{\mathbf{A}}$, $\widetilde{\mathbf{B}}$, $\widetilde{\mathbf{C}}_e$, $\mathbf{D}$. Pro výsledné predikční matice se zavede značení $\widetilde{\mathbf{T}}$, $\widetilde{\mathbf{S}}_{uL}$, $\widetilde{\mathbf{T}}_c$, $\widetilde{\mathbf{S}}_{cuL}$. Predikci vývoje rozdílů požadovaných výstupů systému od těch skutečných pro celý horizont predikcí určuje následující vztah

$$\mathbf{E} = \widetilde{\mathbf{T}}_c \widetilde{\mathbf{x}}_k + \widetilde{\mathbf{S}}_{cuL} \mathbf{U}_{nc}. \tag{24}$$

V tuto chvíli je vhodné vyjádřit si rozdíl hodnot řízení ve dvou po sobě jdoucích krocích $\Delta\mathbf{U}_{nc} = \mathbf{u}_i - \mathbf{u}_{i-1}$ pro celý horizont řízení. Maticově zapsáno

$$\underbrace{\begin{bmatrix} \Delta\mathbf{u}_1 \\ \Delta\mathbf{u}_2 \\ \Delta\mathbf{u}_3 \\ \vdots \end{bmatrix}}_{\Delta\mathbf{U}_{nc}} = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \cdots \\ -\mathbf{I} & \mathbf{I} & \mathbf{0} & \cdots \\ \mathbf{0} & -\mathbf{I} & \mathbf{I} & \\ \vdots & & \ddots & \ddots \end{bmatrix}}_{\mathbf{K}\in\mathbb{R}^{n_c\cdot m \times n_c \cdot m}} \underbrace{\begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \end{bmatrix}}_{\mathbf{U}_{nc}} + \underbrace{\begin{bmatrix} -\mathbf{I} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \end{bmatrix}}_{\mathbf{M}} \mathbf{u}_0. \tag{25}$$

Poslední hodnotu aplikovaného akčního zásahu do systému lze získat jednoduše pomocí vektoru stavu následujícím způsobem

$$\mathbf{u}_0 = \underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{0} & -\mathbf{I} \end{bmatrix}}_{\mathbf{L}_u} \widetilde{\mathbf{x}}_1. \tag{26}$$

2.7.1 Matice optimalizace G a $\mathbf{f^T}$

Tato podkapitola se bude zabývat již sestavením konkrétního tvaru účelové funkce. Ta, jak bylo ukázáno dříve, bude namísto jednotlivých hodnot akčních zásahů penalizovat rozdíly akčních zásahů ve dvou po sobě jdoucích krocích. Pro zajištění sledování referenční hodnoty se budou penalizovat odchylky mezi referencí a výstupem systému v aktuálním kroku. Matematicky to lze vyjádřit jako

$$J = \frac{1}{2} \left(\sum_{i=k}^{k+n_p-1} \mathbf{e}_i^{\mathrm{T}} \mathbf{Q} \mathbf{e}_i + \sum_{i=k}^{k+n_c-1} \Delta\mathbf{u}_i^{\mathrm{T}} \mathbf{R} \Delta\mathbf{u}_i \right), \tag{27}$$

kde $\mathbf{e}_i$ odpovídá rozdílu mezi vektorem referenčních výstupů a vektorem výstupů systému v aktuálním kroku. Matice $\mathbf{Q} \in \mathbb{R}^{p\times p}$, $\mathbf{Q} \succeq 0$ reprezentuje váhovou matici pro odchylku $\mathbf{e}_i$ a matice $\mathbf{R} \in \mathbb{R}^{m\times m}$, $\mathbf{R} \succeq 0$ je váhová matice pro odchylky akčních zásahů ve dvou po sobě jdoucích krocích.

Využitím rovnice (23) lze účelovou funkci (27) upravit do následující podoby

$$J = \frac{1}{2} \left(\left(\widetilde{\mathbf{T}}_c \widetilde{\mathbf{x}}_k + \widetilde{\mathbf{S}}_{cuL} \mathbf{U}_{nc} \right)^{\mathrm{T}} \mathbf{Q}' \left(\widetilde{\mathbf{T}}_c \widetilde{\mathbf{x}}_k + \widetilde{\mathbf{S}}_{cuL} \mathbf{U}_{nc} \right) + \Delta\mathbf{U}_{nc}^{\mathrm{T}} \mathbf{R}' \Delta\mathbf{U}_{nc} \right), \tag{28}$$

kde $\mathbf{Q}' \in \mathbb{R}^{n_p \cdot p \times n_p \cdot p}$ reprezentuje matici penalizující odchylky mezi referenční hodnotou a výstupem v aktuálním kroku pro celý horizont predikcí. Matice $\mathbf{R}' \in \mathbb{R}^{n_c \cdot m \times n_c \cdot m}$ odpovídá váhové matici pro odchylky hodnot řízení ve dvou po sobě jdoucích krocích pro celý horizont řízení.

V účelové funkci je dále možné rozepsat $\Delta \mathbf{U}_{nc}$ pomocí vektoru $\mathbf{U}_{nc}$ a matic $\mathbf{K}$, $\mathbf{M}$, $\mathbf{L}_u$. Finální tvar účelové funkce nyní vychází

$$J = \frac{1}{2}\left(\widetilde{\mathbf{T}}_c\widetilde{\mathbf{x}}_k + \widetilde{\mathbf{S}}_{cuL}\mathbf{U}_{nc}\right)^{\mathrm{T}}\mathbf{Q}'\left(\widetilde{\mathbf{T}}_c\widetilde{\mathbf{x}}_k + \widetilde{\mathbf{S}}_{cuL}\mathbf{U}_{nc}\right) + \frac{1}{2}\Big(\mathbf{K}\mathbf{U}_{nc} + \mathbf{M}\mathbf{L}_u\widetilde{\mathbf{x}}_k\Big)^{\mathrm{T}}\mathbf{R}'\Big(\mathbf{K}\mathbf{U}_{nc} + \mathbf{M}\mathbf{L}_u\widetilde{\mathbf{x}}_k\Big). \tag{29}$$

V tuto chvíli pouze zbývá upravit účelovou funkci do podoby vhodné pro minimalizaci. Pro úlohu optimalizace řešenou v každém kroku algoritmu, která vyšla ve tvaru kvadratické funkce

$$\min_{\mathbf{U}_{nc}} \frac{1}{2}\mathbf{U}_{nc}^{\mathrm{T}}\mathbf{H}\mathbf{U}_{nc} + \mathbf{U}_{nc}^{\mathrm{T}}\mathbf{G}, \tag{30}$$

s ohledem na možná omezení, vyšly výsledné matice optimalizace pro zaručení nulové odchylky v ustáleném stavu ve tvaru

$$\mathbf{H} = \widetilde{\mathbf{S}}_{cuL}^{\mathrm{T}}\mathbf{Q}'\widetilde{\mathbf{S}}_{cuL} + \mathbf{K}^{\mathrm{T}}\mathbf{R}'\mathbf{K}, \tag{31}$$

$$\mathbf{G} = \left(\widetilde{\mathbf{S}}_{cuL}^{\mathrm{T}}\mathbf{Q}'\widetilde{\mathbf{T}}_c + \mathbf{K}^{\mathrm{T}}\mathbf{R}'^{\mathrm{T}}\mathbf{M}\mathbf{L}_u\right)\widetilde{\mathbf{x}}_k = \mathbf{G}_x\widetilde{\mathbf{x}}_k. \tag{32}$$

kde $\mathbf{H} \in \mathbb{R}^{n_c \cdot m \times n_c \cdot m}$, $\mathbf{G} \in \mathbb{R}^{n_c \cdot m \times 1}$ a $\mathbf{G}_x \in \mathbb{R}^{n_c \cdot m \times n}$.

3 Kvadratické programování

Při prediktivní regulaci ve standardním tvaru se musí v každém kroku algoritmu řešit optimalizační úloha. Konkrétně dochází k minimalizaci účelové funkce přes vektor akčních zásahů. Účelová funkce je přitom kvadratická. Tento proces minimalizace kvadratické funkce je obecně označován jako kvadratické programování. Je tedy zřejmé, že pro detailní pochopení principu prediktivního regulátoru je znalost kvadratického programování nezbytná.

Existuje celá řada solverů, které řeší úlohu kvadratického programování. Tyto solvery se mohou lišit přesností nalezeného výsledku ale také například rychlostí konvergence k řešení, s čímž je spojená časová náročnost. Každému solveru může trvat jinak dlouho, než řešení optimalizační úlohy najde. Z toho vyplývá, že výpočetní náročnost celého MPC regulátoru ve standardním tvaru je na typu solveru kvadratické programování silně závislá.

Tato kapitola bude poskytovat stručný úvod do kvadratického programování. Nejdříve zde proběhne definice kvadratického programu. Další podkapitola bude věnována KKT podmínkám optimality, které hrají v kvadratickém programování zásadní roli. Dále bude nastíněn způsob řešení úloh kvadratického programování. Poté budou přestaveny některé solvery, které se používají k řešení kvadratických optimalizačních úloh. Na KKT podmínkách je založeno i multiparametrické kvadratické programování, které přistupuje k řešení optimalizační úlohy jiným způsobem. Multiparametrické kvadratické programování je současně základem explicitního prediktivního regulátoru, kterému bude věnován zbytek teoretické části diplomové práce.

Detailnější úvod do kvadratického programování lze najít například ve zdroji [6].

3.1 Kvadratický program

Před tím, než se přistoupí k definici kvadratického programu, tak by zde bylo vhodné určit, v jaké podobě se bude uvažovat kvadratická funkce. Kvadratickou funkci lze obecně zapsat ve tvaru

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{G}\mathbf{x} + \mathbf{f}^{\mathrm{T}}\mathbf{x} + c, \tag{33}$$

přičemž platí $q : \mathbb{R}^n \to \mathbb{R}$ a současně musí platit $\mathbf{G} \neq \mathbf{0}$. Dále $\mathbf{G} \in \mathbb{R}^{n\times n}$ je symetrická matice ($\mathbf{G} = \mathbf{G}^{\mathrm{T}}$), $\mathbf{f} \in \mathbb{R}^n$ reprezentuje sloupcový vektor a $c \in \mathbb{R}$ je konstanta.

Pro pozdější referenci se zde definují vztahy pro Jakobián, Hessián a gradient kvadratické funkce. V uvedeném pořadí platí

$$q'(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{G} + \mathbf{f}^{\mathrm{T}}, \quad q''(\mathbf{x}) = \mathbf{G}, \quad \nabla q(\mathbf{x}) = \mathbf{G}\mathbf{x} + \mathbf{f} = q'(\mathbf{x})^{\mathrm{T}}. \tag{34}$$

Minimalizace kvadratické funkce s lineárními omezeními se nazývá kvadratický program (často pouze QP). Vzhledem k tomu, že konstanta c nemá žádný vliv na výslednou hodnotu $\mathbf{x}$, tak se často často v úloze kvadratického programování neuvažuje. Matematicky lze kvadratický program definovat jako

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{G}\mathbf{x} + \mathbf{f}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_e\mathbf{x} = \mathbf{b}_e, \end{aligned} \tag{35}$$

kde omezení $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ a $\mathbf{A}_e \in \mathbb{R}^{l \times n}$, $\mathbf{b}_e \in \mathbb{R}^l$ definují množinu přípustných řešení (angl. *feasible set*) pro $\mathbf{x}$.

3.2 KKT podmínky optimality

Pro zaručení optimality řešení nelineární optimalizační úlohy musí platit jisté podmínky. Tyto podmínky se označují jako (*Karush-Kuhn-Tucker*) KKT podmínky optimality. Matematicky řečeno KKT podmínky optimality jsou podmínky **nutné** pro zaručení optimality řešení a v některých případech se dokonce jedná o podmínky **postačující** [6].

Pro některé solvery kvadratických úloh jsou KKT podmínky výchozím bodem pro nalezení řešení daného optimalizačního problému. Nicméně vychází z nich i princip explicitního prediktivního regulátoru. Z toho důvodu se jeví vhodné uvést zde jejich obecný tvar.

V nejobecnějším případě má optimalizační problém následující podobu

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}), \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{h}(\mathbf{x}) = \mathbf{0}, \end{aligned} \tag{36}$$

kde pro minimalizovanou funkci f platí $f : \mathbb{R}^n \to \mathbb{R}$ a pro funkce definující množinu přípustných řešení platí $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ a $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^p$. V bodě minima pak musí platit KKT podmínky optimality v následující podobě

$$\begin{aligned} \nabla f(\mathbf{x}) + \sum_{i=1}^{m} \mu_i \nabla g_i(\mathbf{x}) + \sum_{i=1}^{p} \lambda_i \nabla h_i(\mathbf{x}) &= \mathbf{0}, \\ \mathbf{g}(\mathbf{x}) &\leq \mathbf{0}, \\ \mathbf{h}(\mathbf{x}) &= \mathbf{0}, \\ \mu &\geq \mathbf{0}, \\ \forall i \in \{1, ..., m\} \quad \mu_i g_i(\mathbf{x}) &= 0. \end{aligned} \tag{37}$$

3.3 Algoritmy kvadratické programování

V nejobecnějším případě lze úlohy kvadratického programování rozdělit do dvou kategorií. Jedná se o kvadratické programování bez omezení a kvadratické programování s omezeními ve formě rovností nebo nerovností. V následujících dvou podkapitolách budou tyto dva typy úloh stručně popsány a současně budou ukázány i některé způsoby řešení.

3.3.1 Kvadratické programování bez omezení

Pro tento typ úloh existuje vždy jedno unikátní řešení, které minimalizuje funkční hodnotu zadané kvadratické funkce [5]. Množina přípustných řešení při tom uvažována jako $\mathbb{R}^n$. Formálně lze takovýto typ úloh zapsat následovně

$$\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{G}\mathbf{x} + \mathbf{f}^{\mathrm{T}}\mathbf{x}. \tag{38}$$

Přímočaré řešení takto zadaného problému vychází z KKT podmínek. Ukazuje se, že v tomto případě to vede pouze na jednu podmínku, které by měla platit pro minimum. Tato podmínka říká, že se gradient minimalizované funkce musí rovnat nule. Musí tedy platit

$$\nabla q = \mathbf{G}\mathbf{x} + \mathbf{f} = \mathbf{0}. \tag{39}$$

Jedná se o systém lineárních rovnic, tudíž se lze jednoduchou úpravou dostat k řešení ve tvaru

$$\mathbf{x}^* = -\mathbf{G}^{-1}\mathbf{f} = \mathbf{0}. \tag{40}$$

Nicméně řešení této rovnice nemusí být vždy vhodné a to kvůli nutnosti výpočtu inverze matice $\mathbf{G}$. Z toho důvodu zde budou ještě ve stručnosti představeny 2 další metody, která se používají k řešení úloh kvadratického programování bez omezení.

Metoda snižování gradientu se snaží najít řešení optimalizačního problému bez omezení pomocí iterativního vztahu

$$\mathbf{x}_{k+1} = \mathbf{x} - t_k \cdot \nabla q(\mathbf{x}_k). \tag{41}$$

K minimu se tato metoda posouvá ve směru záporného gradientu a to s délkou kroku t_k. Hlavním problémem u této metody je, jakým způsobem volit právě délku kroku t_k. Existují dva přístupy. V prvním z nich se uvažuje hodnota délky kroku po celou dobu hledání řešení konstantní. Platí tedy, že $t_k = t$ je pozitivní konstanta. Implementace této metody je sice vcelku jednoduchá, ovšem konvergence k řešení může být poměrně pomalá. U druhého přístupu se hodnota délky kroku t_k volí tak, aby v aktuálním kroku došlo k minimalizaci funkční hodnoty minimalizované funkce a to ve směru negativního gradientu. Jinými slovy tato metoda najde ve směru záporného gradientu takový bod, kde funkční hodnota kvadratické funkce nabývá své nejmenší hodnoty. Tento bod se v dalším kroku uvažuje jako ten výchozí. Délku kroku lze hledat pomocí následujícího vztahu [6]

$$t_k^* = \arg\min_{t_k \geq 0} q(\mathbf{x}_k + t_k \cdot \Delta\mathbf{x}_k), \tag{42}$$

kde $\Delta\mathbf{x}_k$ označuje směr záporného gradientu. Vzhledem k tomu, že v tomto případě je minimalizované proměnná t_k, tak výsledný iterativní vztah pro nalezení minima zadané

kvadratické funkce je

$$\mathbf{x}_{k+1} = \mathbf{x} + t_k^* \cdot \Delta\mathbf{x}_k. \tag{43}$$

Newtonova metoda se obecně zabývá řešením soustavy homogenních rovnic

$$\mathbf{g}(\mathbf{x}) = \mathbf{0}. \tag{44}$$

Začne se prvotním odhadem hodnoty řešení a poté se používá iterativní vztah podobně jako u metody snižujícího gradientu. Základní myšlenka Newtonovy metody ale spočívá v tom, že se $\mathbf{g}(\mathbf{x})$ nahradí svým Taylorovým rozvojem řádu 1. Matematicky zapsáno

$$\mathbf{g}(\mathbf{x}) \approx \mathbf{g}(\mathbf{x}_k) + \mathbf{g}'(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k). \tag{45}$$

Newtonova metoda pak řeší soustavu homogenních rovnic (44) s takto upravenou levou stranou. Pokud jsou splněny některé další podmínky, tak řešení soustavy lze nalézt v podobě

$$\mathbf{x} \approx \mathbf{x}_k - \mathbf{g}'(\mathbf{x}_k)^{-1}\mathbf{g}(\mathbf{x}_k). \tag{46}$$

Analogickým způsobem lze postupovat i při hledání řešení úlohy kvadratického programování. První podmínka optimality vyžaduje, aby byl gradient kvadratické funkce roven nule. Musí tedy platit $\nabla q(\mathbf{x}) = \mathbf{0}$. Ukazuje se, že se jedná rovněž o soustavu homogenních rovnic. Vzhledem k tomu může být k nalezení řešení uvažovaného optimalizačního problému použita Newtonova metoda. Její aplikací na daný problém lze dostat výsledný iterativní vztah pro nalezení minima kvadratické funkce jako

$$\mathbf{x}_{k+1} = \mathbf{x}_k - q''(\mathbf{x}_k)^{-1}\nabla q(\mathbf{x}_k). \tag{47}$$

3.3.2 Kvadratické programování s omezeními

Kvadratické programování s omezeními lze rozdělit do dvou skupin a to dle typu omezení. U první skupiny úloh je množina přípustných řešení definována pouze omezeními ve formě rovností. U druhé skupiny úloh figurují omezení pouze ve formě nerovností. Těmto dvěma kategoriím úloh kvadratického programování s omezeními budou věnovány následující dva odstavce.

Kvadratické programování s omezeními ve formě rovností řeší úlohu optimalizace v následující podobě

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{G}\mathbf{x} + \mathbf{f}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}_e\mathbf{x} = \mathbf{b}_e. \end{aligned} \tag{48}$$

Takto zadané optimalizační úlohy ovšem v prediktivním řízení nejsou moc časté. Ačkoli jejich řešení je nezbytné u kvadratických programů s omezením ve formě nerovností. Tento typ úloh lze řešit například pomocí KKT podmínek. Ty pro uvažovanou úlohu vycházejí v následující podobě

$$\begin{aligned} \mathbf{G}\mathbf{x}^* + \mathbf{f} + \mathbf{A}_e^{\mathrm{T}} \cdot \lambda = \mathbf{0}, \\ \mathbf{A}_e\mathbf{x}^* = \mathbf{b}_e. \end{aligned} \tag{49}$$

Takto zapsané KKT podmínky lze zapsat ve formě soustavy rovnic jako

$$\begin{bmatrix} \mathbf{G} & \mathbf{A}_e^{\mathrm{T}} \\ \mathbf{A}_e & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \lambda \end{bmatrix} = \begin{bmatrix} -\mathbf{f} \\ \mathbf{b}_e \end{bmatrix}. \tag{50}$$

Pokud $\mathbf{G}$ je pozitivně definitní, $\mathbf{A}_e$ má plnou hodnost a množina přípustných řešení je neprázdná, tak existuje unikátní řešení, které minimalizuje zadanou úlohu.

Kvadratické programování s omezeními ve formě nerovností řeší úlohu optimalizace v následující podobě

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{G}\mathbf{x} + \mathbf{f}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{51}$$

Množina přípustných řešení je zde definována pomocí sady nerovností $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. Každý řádek $\mathbf{a}_i^{\mathrm{T}}$ matice $\mathbf{A}$ společně s odpovídajícím prvkem b_i vektoru $\mathbf{b}$ definuje poloprostor $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i, \forall i \in \{1, ..., m\}$. Průnik všech těchto poloprostorů definuje množinu přípustných řešení. Tato třída úloh kvadratického programování je nejobecnější ze všech. Pomocí ní je totiž možné definovat úlohy kvadratického programování bez omezení a i s omezeními ve formě rovností.

Jak řešit takto zadané úlohy kvadratického programování, kde je přípustná množina definována pomocí sady nerovností, bude demonstrováno pomocí základní myšlenky metody aktivních množin (další metody řešící tento typ problému jsou detailně ukázány ve zdroji [6]). Metoda aktivních množin využívá toho faktu, že minimum se často nachází na hranici množiny přípustných řešení. Jinými slovy některá omezení jsou pro nalezené řešení kvadratického programu *aktivní*. Aby dané omezení bylo aktivní, tak pro daný řádek ze soustavy $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ musí platit rovnost. Množina všech těchto omezení, pro které platí rovnost, se nazývá *aktivní množinou*. Aktivní množina se v každém kroku algoritmu mění v závislosti na tom, která omezení jsou aktivní. V každém kroku algoritmu je tedy nutné rozhodnout, zda do aktivní množiny přidat, či z aktivní množiny odebrat některé z omezení. To je realizováno tak, že se v každém kroku omezení z aktivní množiny uvažují jako omezení ve formě rovností. Ostatní omezení v daný okamžik uvažovaná nejsou. Dále je řešen kvadratický program s omezeními ve formě rovností, které odpovídají právě aktivní množině. Tím se algoritmus dostane do nového kroku a celý proces se znovu opakuje. To znamená, že se definuje nová aktivní množina, vyřeší se kvadratický program a algoritmus se dostane opět do nového bodu. Algoritmus končí, když již nelze dosáhnout žádného dalšího snížení funkční hodnoty zadané kvadratické funkce.

Mezi další algoritmy řešící úlohy kvadratického programování s omezeními ve formě nerovností patří například metoda rychlého gradientu nebo metoda vnitřních bodů.

3.4 Solvery úloh kvadratického programování

V současné době existují poměrně dobře odladěné solvery, které se k řešení kvadratických programů používají. Některé jsou už přímo vytvářeny pro aplikaci do prediktivních regulátorů. Tato podkapitola bude sloužit k představení dvou solverů, které se při vytváření diplomové práce použily.

3.4.1 Solver quadprog z Matlabu

Praktická část této diplomové práce byla ze značné části realizována v softwaru Matlab. Ten nabízí pro řešení kvadratické optimalizační úlohy příkaz $\gg$ `quadprog`($\mathbf{H}$, $\mathbf{f}$, $\mathbf{A}$, $\mathbf{b}$, $\mathbf{A}_{eq}$, $\mathbf{b}_{eq}$, $\mathbf{LB}$, $\mathbf{UB}$). Úloha kvadratického programování je zde definována jako

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{H}\mathbf{x} + \mathbf{f}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & \mathbf{LB} \leq \mathbf{x} \leq \mathbf{UB}. \end{aligned} \tag{52}$$

Parametry příkazu `quadprog` tedy jsou matice optimalizace $\mathbf{H}$ a $\mathbf{f}$. Dále matice pro definici omezení na optimalizovanou proměnnou $\mathbf{A}$ a $\mathbf{b}$ ve tvaru $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. Matice $\mathbf{A}_{eq}$ a $\mathbf{b}_{eq}$ slouží pro specifikaci omezení na minimalizační proměnnou ve tvaru rovností. Posledními uvedenými vstupními parametry uvedeného příkazu pro minimalizaci kvadratické funkce jsou vektory $\mathbf{LB}$ a $\mathbf{UB}$. Pomocí těch lze přímo vymezit hodnoty, kterých optimalizační proměnná může nabývat.

Řešení minimalizační úlohy je zde hledáno prostřednictvím metody aktivních množin v základním tvaru. Tento solver ovšem nepatří obecně mezi ty nejrychlejší a čas, který potřebuje k nalezení řešení může být delší. Vzhledem k této skutečnosti by nebylo vhodné `quadprog` použít při implementaci prediktivního regulátoru, který by měl řídit systémy s malou periodou vzorkování. Z toho důvodu se přistoupilo k odzkoušení ještě jiného solveru kvadratických programů a to *qp-OASES*, který bude představen v následující podkapitole.

3.4.2 Solver qp-OASES

Pro solver *qp-OASES* by mělo být dosaženo lepších výsledků z hlediska výpočetní náročnosti než u solveru *quadprog* z Matlabu. Solver *qp-OASES* je volně stažitelný ze stránky *https://github.com/coin-or/qpOASES* a byl vytvořen právě pro účely prediktivního řízení. Výhodou je zde to, že ho lze poměrně jednoduše zahrnout i do prostředí Matlab. To bylo realizováno pomocí zdroje [2]. Jedná se o open-source implementaci určité varianty numerické metody vhodné pro výpočty s malou vzorkovací periodou. Konkrétně se používá upravená metoda aktivních množin. Využívá se zde toho, že v úlohách prediktivního řízení se kvadratický program v každém kroku liší pouze v hodnotách vektoru $\mathbf{f}$, které jsou závislé pouze na stavu systému. Výhodou tohoto solveru je rovněž možnost použití tzv. hot-start varianty, která má výrazně kratší dobu výpočtu. To je dosaženo pomocí použití mezivýsledků z minulého kroku.

Tento solver bude rovněž použit i v řídicím softwaru REXYGEN, pomocí kterého bude probíhat v praktické části diplomové práce řízení elektromechanického ramene. Použije se zde na řešení optimalizační úlohy u MPC regulátoru ve standardním tvaru.

3.5 Multiparametrické kvadratické programování

Cílem multiparametrického kvadratického programování je rovněž najít takovou hodnotu optimalizační proměnné, která by minimalizovala zadanou kvadratickou funkci s ohledem na omezení ve formě rovností a nerovností. Nicméně kromě minimalizační proměnné se zde uvažuje ještě další proměnná, na jejíž hodnotě je výsledek celé optimalizace závislý. Matematicky lze úlohu multiparametrického kvadratického programování zapsat jako

$$\begin{aligned} \min_{\mathbf{U}} \quad & \frac{1}{2}\mathbf{U}^{\mathrm{T}}\mathbf{H}\mathbf{U} + \mathbf{x}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}\mathbf{U} \\ \text{s.t.} \quad & \mathbf{G}\mathbf{U} \leq \mathbf{W} + \mathbf{S}\mathbf{x}. \end{aligned} \tag{53}$$

Hlavní myšlenka spočívá v tom, že namísto řešení úlohy kvadratického programování online pro jednu konstantní hodnotu proměnné $\mathbf{x}$ je zde cílem vyřešit optimalizační úlohu předem (off-line) pro všechny přípustné hodnoty $\mathbf{x}$ (tzn. $\mathbf{x} \in X_f$, X_f označuje množinu všech přípustných hodnot $\mathbf{x}$) [8]. Omezení na $\mathbf{U}$ je při tom zadáno ve formě rovnosti či nerovností a je součástí formulace řešené úlohy. Ve zdroji [9] je dokázáno, že pro úlohu multiparametrického kvadratického programování ve tvaru (53) je hodnota řešení $\mathbf{U}^*$ po částech lineární a spojitou funkcí $\mathbf{x} \in X_f$. Jinými slovy výslednou hodnotu optimalizační proměnné $\mathbf{U}^*$, která minimalizuje zadanou kvadratickou funkci, lze získat explicitně jako funkci proměnné $\mathbf{x}$. Pro řešení kvadratického programu tedy platí $\mathbf{U}^* = \mathbf{U}^*(\mathbf{x})$.

Z multiparametrického kvadratického programování vychází celý princip explicitního prediktivního regulátoru. Tato podkapitola měla sloužit pouze k zachycení hlavní myšlenky tohoto problému. Detailní řešení úlohy multiparametrického kvadratického programování bude ukázáno v následující kapitole, kde bude již aplikováno přímo na explicitní MPC regulátor.

4 Explicitní prediktivní řízení

Explicitní prediktivní řízení je speciální způsob prediktivní regulace, u které nedochází k řešení kvadratického programu v každém kroku algoritmu. Snaha je zde o to, aby prediktivní regulátor pro nalezení optimálního akčního zásahu musel provést on-line co nejméně operací. Ukazuje se, že výstupy explicitního prediktivního regulátoru a jeho ekvivalentní verze MPC ve standardním tvaru s počítáním optimalizace on-line jsou naprosto totožné [5]. I přes to je u explicitního prediktivního regulátoru dosaženo mnohonásobně menší výpočetní náročnosti. Z tohoto důvodu je vhodné používat explicitní verzi MPC pro řízení systémů s velice malou periodou vzorkování.

Celý princip explicitního prediktivního regulátoru vychází z úlohy *multiparametrického kvadratického programování*. Současně je důležité mít k dispozici ve formě nerovností všechna omezení, která je chtěné klást na regulační smyčku. Ukazuje se totiž, že v případě MPC bez jakýchkoliv omezení lze prediktivní řízení převést na klasickou stavovou zpětnou vazbu [7]. Na základě zadaných omezení je možné definovat v prostoru stavů řízeného systému jisté regiony. Každému z těchto regionů se přiřadí adekvátní zákon řízení, který je ve formě lineární funkce vzhledem k vektoru stavu řízeného systému. Jediné, co je pak nutné u explicitního prediktivního regulátoru řešit on-line, je určit, do jakého z nalezených regionů aktuální stav systému patří. Následně již stačí pouze vypočítat funkční hodnotu lineární funkce přidruženou k nalezenému regionu, do které se dosadí stav systému. Výsledkem je již optimální akční zásah, který se použije jako vstup do řízeného systému.

V této kapitole budou postupně rozebrány následující témata. Jako první zde bude definován problém explicitního prediktivního řízení. Konkrétně proběhne formální zápis úlohy multiparametrického kvadratické programování a určení, v jaké podobě je u explicitního regulátoru chtěné nalézt řešení. Následovat bude již samotný algoritmus explicitního MPC regulátoru. Současně zde budou definovány i problematické úlohy, které je nutné při návrhu tohoto regulátoru řešit. Řešení těchto problémů bude ukázáno v posledních dvou podkapitolách této sekce práce. Konkrétně jde o úlohu hledání všech regionů v prostoru stavů systému a určení konkrétního regionu pro daný stav systému.

4.1 Definice problému

U MPC regulátoru ve standardní podobě je nutné v každém kroku algoritmu řešit optimalizační úlohu ve tvaru

$$\begin{aligned} \min_{\mathbf{U}} \quad & \frac{1}{2}\mathbf{U}^{\mathrm{T}}\mathbf{H}\mathbf{U} + \mathbf{x}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}\mathbf{U} \\ \text{s.t.} \quad & \mathbf{G}\mathbf{U} \leq \mathbf{W} + \mathbf{S}\mathbf{x}, \end{aligned} \tag{54}$$

kde $\mathbf{U}$ definuje optimální akční zásahy pro následujících n_c kroků, $\mathbf{H} \in \mathbb{R}^{n_c \times n_c}$ je Hessova matice, $\mathbf{x} \in \mathbb{R}^n$ odpovídá aktuálnímu vektoru stavu systému, pro $\mathbf{F}$ platí $\mathbf{F} \in \mathbb{R}^{n_c \times n}$. Dále se pomocí matic $\mathbf{G} \in \mathbb{R}^{q \times n_c}$, $\mathbf{W} \in \mathbb{R}^q$ a $\mathbf{S} \in \mathbb{R}^{q \times n}$ definují omezení na akční veličinu, hodnotu stavů systému nebo regulované veličiny.

Optimální akční zásah se pro aktuální krok získá pomocí jednoduchého vztahu

$$\mathbf{u}^*(\mathbf{x}) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \dots \mathbf{0} \end{bmatrix} \mathbf{U}^*, \tag{55}$$

kde $\mathbf{U}^*$ odpovídá řešení minimalizačního problému (54).

Hlavní cíl u explicitního prediktivního regulátoru je následující. Na rozdíl od klasického MPC, kde se řeší on-line optimalizační úloha ve tvaru (54) pro jeden konkrétní stav systému, je zde snaha vyřešit kvadratický program předem (off-line) pro všechny přípustné hodnoty stavu systému $\mathbf{x}$. Ve výsledku pak vztah pro optimální akční zásah $\mathbf{u}^*$ vyjde explicitně jako lineární funkce stavu systému $\mathbf{x}$.

Nástroj, který se přímo nabízí k řešení takto definovaného problému, je *multiparametrické kvadratické programování*. Ve zdroji [9] je dokázáno, že pro úlohu multiparametrického kvadratického programování ve tvaru (54) je hodnota řešení $\mathbf{U}^*$ po částech lineární a spojitou funkcí $\mathbf{x} \in X_f$. Množina X_f při tom definuje množinu všech přípustných hodnot stavu systému $\mathbf{x}$. Výsledný obecný vztah pro optimální akční zásah, který se pro aktuální stav systému aplikuje do řízené soustavy, lze zapsat jako

$$\mathbf{u}^*(\mathbf{x}) = \begin{cases} \mathbf{M}_1\mathbf{x} + \mathbf{N}_1 & \text{if } \mathbf{A}_1\mathbf{x} \leq \mathbf{b}_1 \\ \vdots & \vdots \\ \mathbf{M}_M\mathbf{x} + \mathbf{N}_M & \text{if } \mathbf{A}_M\mathbf{x} \leq \mathbf{b}_M \end{cases}. \tag{56}$$

Zde matice $\mathbf{A}_m$ a vektory $\mathbf{b}_m$ definují prostřednictvím nerovnic regiony v prostoru stavů řízeného systému. Pomocí matic $\mathbf{M}_m$ a vektorů $\mathbf{N}_m$ se po dosazení aktuálního stavu systému získá optimální akční zásah pro daný region. Vše, co tedy musí explicitní prediktivní regulátor řešit on-line, je určit region, do kterého aktuální stav systému patří a následně pouze vyhodnotit odpovídající lineární funkci pomocí matice $\mathbf{M}_m$, vektoru $\mathbf{N}_m$ a stavu systému. Vypočtený akční zásah se posléze aplikuje do řízeného systému.

4.2 Algoritmus explicitního MPC

Vzhledem k tomu, že explicitní MPC vychází z multiparametrického kvadratického programování, tak algoritmus explicitního prediktivního regulátoru je totožný s algoritmem pro řešení úlohy multiparametrického kvadratického programování. Popis tohoto algoritmu bude uveden v této podkapitole. Jako zdroj informací zde bude použita především práce [5]. Ve výsledku se tedy dostanou vztahy, které se používají při implementaci explicitního prediktivního řízení.

Existuje několik algoritmů pro řešení úloh multiparametrického kvadratického programování, ovšem všechny vycházejí z KKT podmínek optimality. Ty mají pro uvažovanou úlohu následující tvar

$$\mathbf{HU} + \mathbf{Fx} + \mathbf{G}^{\mathrm{T}}\lambda = \mathbf{0}, \tag{57}$$

$$\lambda_i(\mathbf{G}^i\mathbf{U} - W^i - \mathbf{S}^i\mathbf{x}) = 0,\ \forall i = 1, ..., q, \tag{58}$$

$$\mathbf{GU} \leq \mathbf{W} + \mathbf{Sx}, \tag{59}$$

$$\lambda \geq \mathbf{0}, \tag{60}$$

kde $\lambda \in \mathbb{R}^q$ reprezentuje vektor Lagrangeových multiplikátorů. Pro striktně konvexní program (54) jsou uvedené KKT podmínky postačující pro zaručení optimality [5].

Algoritmus pro řešení multiparametrické kvadratické úlohy začíná náhodným určením hodnoty vektoru stavu systému $\mathbf{x}_0 \in \mathbb{R}^n$ (obvykle se volí rovnovážný stav řízeného systému). Následně je potřeba vyřešit kvadratický program (54) s určenou hodnotou $\mathbf{x}$. Tedy platí $\mathbf{x} = \mathbf{x}_0$. Získá se tak řešení minimalizační úlohy $\mathbf{U}^*$.

V dalším postupu je potřeba určit množinu aktivních a neaktivních omezení. Aktivní omezení se značí symbolem tildy a odpovídají řádkům matic $\mathbf{G}$, $\mathbf{S}$ a $\mathbf{W}$, pro které platí rovnost

$$\tilde{\mathbf{G}}\mathbf{U}^* = \tilde{\mathbf{W}} + \tilde{\mathbf{S}}\mathbf{x}. \tag{61}$$

Zbývající řádky z matic $\mathbf{G}$, $\mathbf{S}$ a $\mathbf{W}$ definují množinu neaktivních omezení. Ty se značí symbolem stříšky a platí pro ně

$$\hat{\mathbf{G}}\mathbf{U}^* \leq \hat{\mathbf{W}} + \hat{\mathbf{S}}\mathbf{x}. \tag{62}$$

Stejné dělení na aktivní i neaktivní platí i pro Lagrangeovy multiplikátory. Lze tedy zapsat

$$\tilde{\lambda}(\mathbf{x}) \geq \mathbf{0}, \tag{63}$$

$$\hat{\lambda}(\mathbf{x}) = \mathbf{0}. \tag{64}$$

Nerovnosti (62) a (63) společně tvoří množinu omezení definující region, kterému přísluší vybraný stav systému $\mathbf{x}_0$.

Pro jednoduchost se uvažuje, že řádky matice $\tilde{\mathbf{G}}$ jsou lineárně nezávislé. Ze vztahu (57) platí

$$\mathbf{U} = -\mathbf{H}^{-1}(\mathbf{F}\mathbf{x} + \tilde{\mathbf{G}}^{\mathrm{T}}\tilde{\lambda}). \tag{65}$$

Dále po dosazení této rovnice do vztahu (61) vychází

$$\tilde{\lambda} = -\tilde{\mathbf{M}}(\tilde{\mathbf{W}} + (\tilde{\mathbf{S}} + \tilde{\mathbf{G}}\mathbf{H}^{-1}\mathbf{F})\mathbf{x}), \tag{66}$$

kde $\tilde{\mathbf{M}} = \tilde{\mathbf{G}}^{\mathrm{T}}(\tilde{\mathbf{G}}\mathbf{H}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}})^{-1}$. Jako poslední se rovnice (66) dosadí do (65), z čehož vyjde finální vztah pro vektor optimálních akčních zásahů v závislosti na $\mathbf{x}$ ve tvaru

$$\mathbf{U}^*(\mathbf{X}) = \mathbf{H}^{-1}(\tilde{\mathbf{M}}\tilde{\mathbf{W}} + \tilde{\mathbf{M}}(\tilde{\mathbf{S}} + \tilde{\mathbf{G}}\mathbf{H}^{-1}\mathbf{F})\mathbf{x} - \mathbf{F}\mathbf{x}). \tag{67}$$

To lze ještě upravit do vhodnější podoby, aby byl lépe patrný lineární a konstantní člen lineární funkce. Poté vychází

$$\mathbf{U}^*(\mathbf{X}) = \left(\mathbf{H}^{-1}\tilde{\mathbf{M}}\tilde{\mathbf{S}} + \mathbf{H}^{-1}\tilde{\mathbf{M}}\tilde{\mathbf{G}}\mathbf{H}^{-1}\mathbf{F} - \mathbf{H}^{\mathrm{T}}\mathbf{F}\right)\mathbf{x} + \mathbf{H}^{\mathrm{T}}\tilde{\mathbf{M}}\tilde{\mathbf{W}}. \tag{68}$$

Tento vztah pro optimální akční zásah platí pro všechna $\mathbf{x}$ se stejnou množinou aktivních omezení. Respektive platí pro všechna $\mathbf{x}$, pro které jsou současně splněny nerovnosti (62) a (63). Jinými slovy tyto dvě nerovnosti definují v prostoru stavů řízeného systému region (resp. množinu hodnot $\mathbf{x}$), pro který platí zákon řízení (68).

Region, který obsahuje rovnovážný stav řízeného systému, se označuje jako kritický. Formálně lze zapsat, že pro něj platí

$$CR_0 = \{\mathbf{x} \in \mathbb{R}^n : \tilde{\lambda} \geq 0, \hat{\mathbf{G}}\mathbf{U} \leq \hat{\mathbf{W}} + \hat{\mathbf{S}}\mathbf{x}\}. \tag{69}$$

Obdobným způsobem lze nalézt region v prostoru stavů s odpovídajícím zákonem řízení pro jakoukoliv hodnotu stavu řízeného systému resp. pro jakoukoliv kombinaci aktivních omezení. Po prozkoumání stavového prostoru analogickým způsobem lze získat sadu regionů a jím odpovídajícím zákonům řízení ve formě (56). Otázkou ovšem zůstává, jakým způsobem systematicky všechny regiony s adekvátním zákonem řízení ve stavovém prostoru nalézt. Tomu bude věnována následující podkapitola. Po nalezení všech regionů je ale nutné řešit další klíčový problém. Ten se zabývá tím, jak při on-line běhu explicitního MPC nalézt co nejrychleji region, do kterého aktuální stav systému patří.

4.3 Hledání regionů

Jak již bylo řečeno dříve, tak jedna z problematických úloh explicitního prediktivního řízení je hledání všech regionů v prostoru stavů řízeného systému, pro které platí odlišné zákony řízení. Pro řešení tohoto problému již byly vynalezeny jisté metody. Některé z nich budou představeny v této podkapitole. Konkrétně zde bude představena metoda sekvenčního prohledávání stavového prostoru, metoda postupného otáčení nerovností, metoda proměnné délky kroku a metoda změny aktivní množiny na základě původu omezení.

Všechny tyto metody budou pro lepší představu demonstrovány na příkladu dvojitého integrátoru. Bude použit jeho diskrétní stavový popis ve tvaru

$$\begin{aligned} \mathbf{x}_{k+1} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k, \\ \mathbf{y}_k &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}_k. \end{aligned} \tag{70}$$

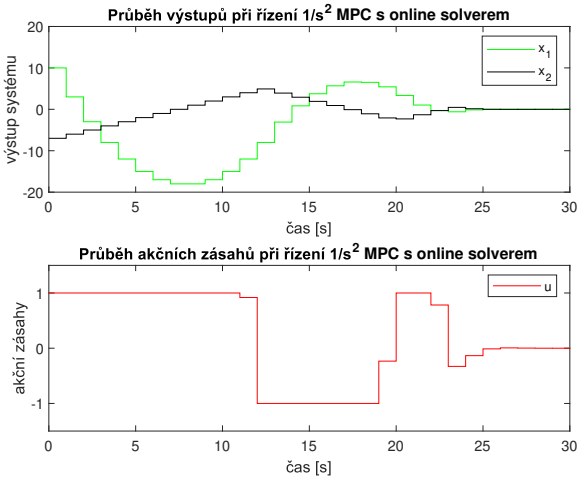
Jedná se o systém s jedním vstupem a dvěma výstupy s periodou vzorkování $T_s = 1\ s$. Tento systém má dva stavy a díky tomu bude dobře možné všechny regiony vykreslit v 2D prostoru. Pro horizont predikcí a řízení bude shodně platit $n_p = n_c = 2$. Matice optimalizace budou mít následující podobu

$$\mathbf{H} = \begin{bmatrix} 0.8365 & 0.3603 \\ 0.3603 & 0.2059 \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} 0.4624 & 0.1682 \\ 1.2852 & 0.5285 \end{bmatrix}. \tag{71}$$

Matice optimalizace v tomto tvaru pro prediktivní řízení řeší úlohu regulátoru, tj. převedení systému do rovnovážného stavu. Pro všechny simulace v této podkapitole bude uvažována následující počáteční podmínka pro stav systému $\mathbf{x}_0 = [10 - 7]^{\mathrm{T}}$. Bude kladeno omezení pouze na akční zásah a to pro celý horizont predikcí ve tvaru $-1 \leq u_k \leq 1$. Matice definující omezení pro úlohu multiparametrického kvadratického programování tedy budou mít tvar

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{S} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{72}$$

Při takto definovaném prediktivním řízení vyšel průběh stavů systému a výstupu regulační smyčky s MPC regulátorem ve standardní formě s on-line solverem následovně.



Obrázek 2: Průběh stavů a výstupu dvojitého integrátoru pří řízení MPC s on-line solverem

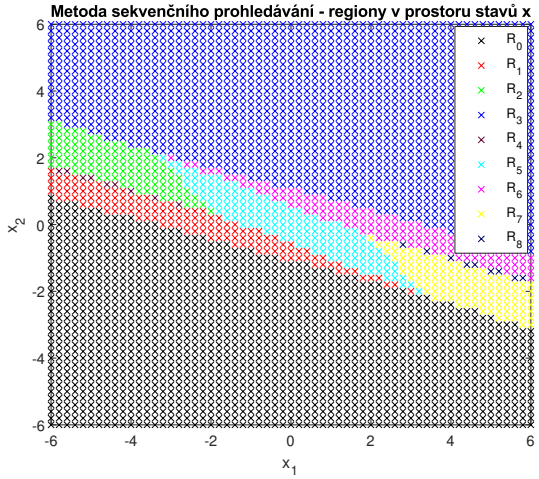
Jak je vidět, tak se podařilo pomocí MPC regulátoru převést systém do počátku stavového prostoru. Současně byla dodržena i omezení na hodnotu akční veličiny.

4.3.1 Sekvenční prohledávání stavového prostoru

Tento přístup k hledání regionů ve stavovém prostoru se jeví jako nejvíce přímočarý. Jeho princip spočívá v tom, že se v celém prostoru stavů řízeného systému definují body v dostatečně jemném rastru tak, aby se v každém regionu vyskytl alespoň jeden definovaný bod. V následujícím postupu se pro každý definovaný bod ve stavovém prostoru aplikuje algoritmus multiparametrického kvadratického programování pro nalezení regionu, do kterého aktuální stav patří. Pokud se jedná o dosud neznámý region, tak se přidá do množiny známých regionů a pokračuje se s následujícím stavem systému v pořadí. Algoritmus končí,

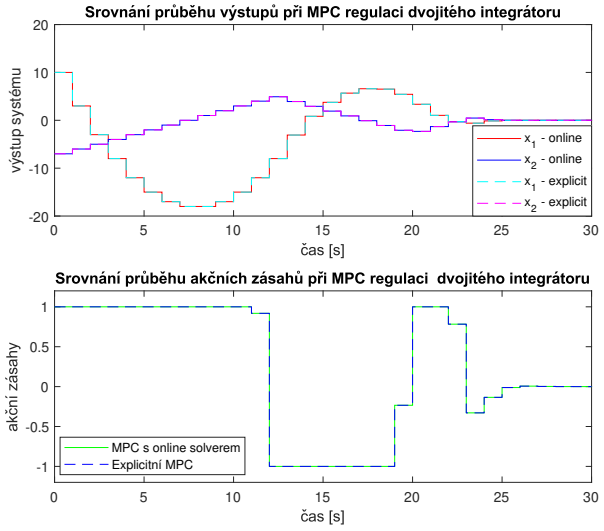
pakliže jsou tímto způsobem prozkoumány všechny definované body stavového prostoru.

Po aplikaci tohoto algoritmu se pro případ dvojitého integrátoru našly následující regiony.



Obrázek 3: Metoda sekvenčního prohledávání - nalezené regiony

Je zřejmé, že bylo nalezeno 9 regionů. Přitom pro regiony R_0, R_1 a R_4 byl nalezený stejný zákon řízení. Stejně tak u regionů R_3, R_6 a R_8 platí stejný zákon řízení. V tuto chvíli by bylo vhodné vyhodnotit, zda si odpovídají simulace chování uzavřené smyčky při řízení dvojitého integrátoru pomocí MPC ve standardním tvaru a pomocí explicitního MPC s takto nalezenými regiony. Výsledek zachycuje tento obrázek.



Obrázek 4: Metoda sekvenčního prohledávání - srovnání simulací s MPC s on-line solverem

Je vidět, že simulace si pro oba případy implementace odpovídají.

Ačkoliv se dostaly u této metody příznivé výsledky a je zaručené nalezení všech regionů v prostoru stavů při dostatečně jemném rastru, tak by jí v praxi nebylo možné použít. Tento způsob implementace MPC musí totiž vyhodnocovat kvadratickou úlohu pro obrovské množství stavů systému. Pro příklad na dvojitém integrátoru, který má pouze 2 stavy, bylo

ještě přípustné tuto metodu implementovat. Avšak pro jakýkoliv systém s více stavovými proměnnými by nebylo možné tuto metodu vzhledem k obrovskému množství operací, které by se musely provést, zrealizovat.

4.3.2 Metoda postupného otáčení nerovností

Tato metoda byla představena ve zdroji [9]. Prvním krokem je zde určit kritický region. Pro ten při tom platí

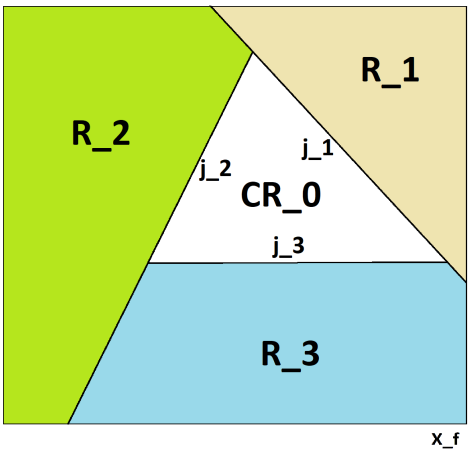
$$CR_0 = \{\mathbf{x} \in X_f : \mathcal{A}\mathbf{x} \leq \mathcal{B}\}. \tag{73}$$

Po jeho nalezení, tj. po nalezení všech nerovnic $\mathcal{A}\mathbf{x} \leq \mathcal{B}$, které ho definují, lze použít následující formuli pro nalezení zbývajících regionů

$$R_i = \{\mathbf{x} \in X_f : \mathcal{A}^i\mathbf{x} > \mathcal{B}^i, \mathcal{A}^j\mathbf{x} \leq \mathcal{B}^j, \forall j < i\}, i \in \{1, ..., m\}, \tag{74}$$

kde m značí počet nerovnic, které definují kritický region. Tento vztah říká, že pro nalezení i-tého regionu je potřeba otočit znaménko u i-té nerovnosti definující kritický region. Při tom je zapotřebí stále uvažovat všechny nerovnice j definující kritický region, tj. nerovnice $j < i$, a zanedbat nerovnice j, pro které platí $j > i$. Tímto způsobem se projde všech m nerovnic definující kritický region a tím pádem dojde k nalezení m regionů obklopující kritický region.

Pro lepší představu zde bude tato metoda ukázána na následujícím triviálním příkladu. Je uvažována následující situace v prostoru stavů.



Obrázek 5: Metoda otáčení nerovností - ilustrační příklad

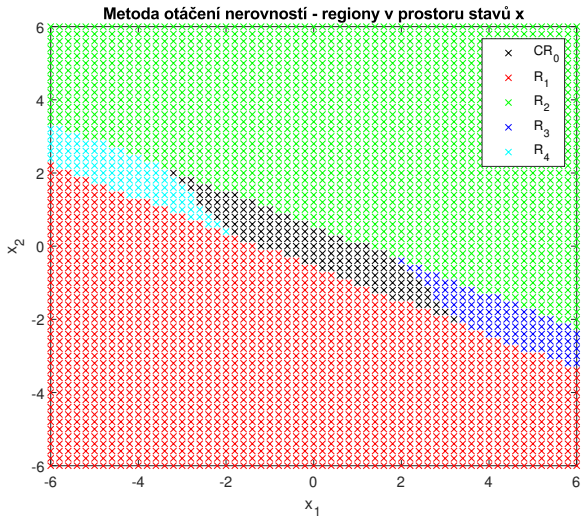
Po nalezení kritického regionu CR_0 jsou k dispozici nerovnosti j_1, j_2 a j_3, které ho definují. Pro nalezení regionu R_1 je zapotřebí otočit nerovnost u první nerovnice, která definuje kritický region, tj. j_1. U regionu R_2 se ponechá nerovnice j_1 v původním tvaru a u nerovnice j_2 se otočí znaménko nerovnosti. Analogickým způsobem se naleznou nerovnice definující region R_3.

Výhodou této metody je to, že z hlediska regionů vždy dojde k prozkoumání celého sta-

vového prostoru. Každý stav z přípustné množiny řešení tedy bude příslušit některému z regionů. Nicméně u této metody je i jedna velká nevýhoda. Mohou zde totiž vznikat tzv. *umělé střihy* některých regionů ve stavovém prostoru. Důsledkem je pak to, že některé regiony mají trochu jinak situované hranice, než je to ve skutečnosti. Důvod vzniku tohoto problému může být patrný ze vztahu (74), který popisuje způsob prohledávání stavového prostoru touto metodou. Z rovnice je zřejmé, že se musí pracovat s nerovnicemi. Právě ty přinášejí do problému v jistém smyslu tolerance, díky kterým umělé střihy vznikají. Tento problém bude pro lepší představu rozebrán na uvedeném ilustračním příkladu. Nechť nerovnice j_1 definuje region R_1 a je ve tvaru $\mathbf{ax} < \mathbf{b}$. Je zřejmé, že takto definovaná nerovnice tvrdě vymezuje poloprostor, ve kterém se $\mathbf{x}$ nacházet **nesmí**. Naopak tato nerovnice vymezuje také prostor, kde se $\mathbf{x}$ nacházet **může**. Problém je právě v tom nerovnítku "menší/větší než". Nikde totiž není definována přesná hranice určující analyzovaný region. Umělé střihy tedy vznikají na hranicích regionů R_i, kde prostřednictvím této metody nelze zcela přesně hranici regionu určit.

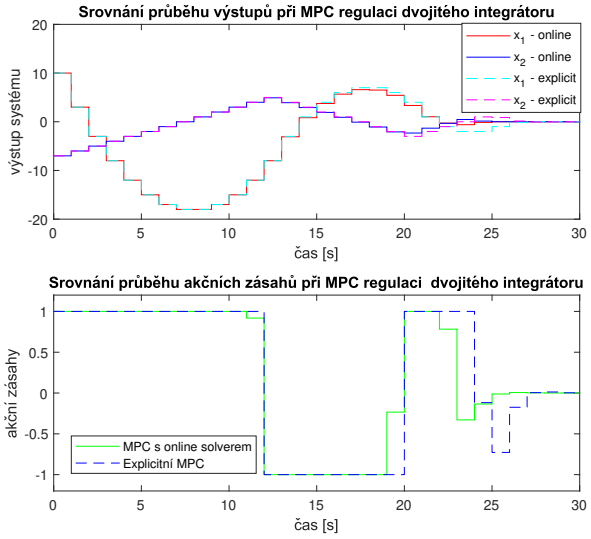
Další nevýhodou je to, že když by se přeházelo pořadí nerovnic definující kritický region, tak ve výsledku by vyšlo rozložení regionů trochu jiné než na přiloženém obrázku s ilustračním příkladem.

Po aplikaci tohoto algoritmu na systém dvojitého integrátoru vyjdou v prostoru stavů následující regiony.



Obrázek 6: Metoda otáčení nerovností - nalezené regiony

Při porovnání výsledku s předchozí metodou sekvenčního prohledávání je zřejmé, že zde skutečně došlo ke zmiňovaným umělým střihům. Zbývá vyhodnotit průběhy simulací chování uzavřené smyčky při řízení dvojitého integrátoru pomocí MPC ve standardním tvaru a pomocí explicitního MPC s takto nalezenými regiony. Výsledek zachycuje následující obrázek.

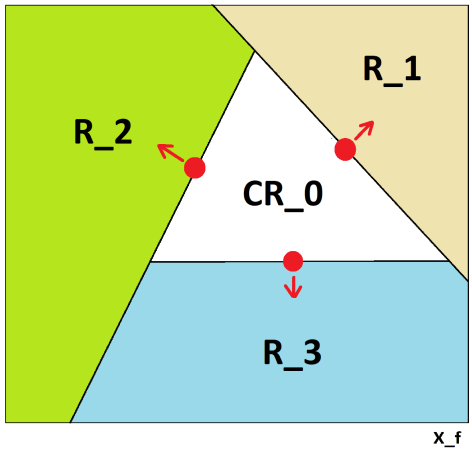


Obrázek 7: Metoda otáčení nerovností - srovnání simulací s MPC s on-line solverem

Je patrné, že došlo k očekávanému výsledku. Průběhy simulací si zde neodpovídají vlivem umělých střihů, ke kterým došlo při hledání regionů ve stavovém prostoru. V praxi by tedy nebylo vhodné tuto metodu použít.

4.3.3 Metoda proměnné délky kroku

Následující metoda byla prezentována ve zdroji [10]. Jako první je zde opět nutné nalézt kritický region. Tedy region kde se vyskytuje rovnovážný stav systému. Dále se opět pracuje s nerovnicemi, které kritický region definují. Konkrétně se používají rovnice definující nadroviny, což jsou vlastně hranice kritického regionu. Pro každou z těchto rovnic reprezentující hranici kritického regionu je potřeba najít její střed vzhledem k celkovému tvaru kritického regionu. Pro n-dimenzionální prostor je tedy zapotřebí hledat střed nadrovin definující kritický region. To lze provést například pomocí lineárního programování. Jednoduchý příklad této metody ve 2-D prostoru zachycuje následující obrázek.



Obrázek 8: Metoda proměnné délky kroku - ilustrační příklad

Středy jsou zde označeny červeným kolečkem. Klíčovou částí této metody je určit směr a délku kroku, se kterou je chtěné "vystoupit" z kritického regionu (naznačeno červenými šipkami). Problém je zde určit délku kroku, která musí být dostatečně velká vzhledem k numerickým omezením a současně dostatečně malá, aby nový bod patřil skutečně do sousedního regionu. Po nalezení takového bodu se prostřednictvím něj vyřeší multiparametrický kvadratický program, jehož výsledkem budou nerovnice definující nově nalezený region a také odpovídající zákon řízení.

Analogickým způsobem se takto postupuje u každého nově nalezeného regionu, dokud není prozkoumán celý stavový prostor.

Ačkoliv je tato metoda velice efektivní, tak i zde existuje jedna velká nevýhoda, kvůli které není vhodné tuto metodu v praxi používat. Tato metoda funguje dobře pouze tehdy, když pro každý jeden region jeho každá jedna hranice odpovídá právě jedné hranici sousedního regionu (angl. *facet-to-facet property*). To lze dobře ilustrovat například na uvedeném obrázku. V případě, že by zde region R_2 byl uvažován jako kritický, tak jedna jeho hrana sousedí s regionem CR_0 i R_3. Tím pádem by tato podmínka na *facet-to-facet* nebyla dodržena a nebylo by dosaženo správného výsledku.

4.3.4 Metoda změny aktivní množiny na základě původu omezení

Tento způsob hledání regionů v prostoru stavů systému je uveden ve zdroji [11]. Stejně jako v předchozích metodách je i zde nejdříve zapotřebí nalézt kritický region. Tedy region kde se nachází rovnovážný stav systému.

Po nalezení kritického regionu je k dispozici aktivní množina s nerovnicemi (resp. omezeními), které tento region definují, ve tvaru $\mathcal{A}\mathbf{x} \leq \mathcal{B}$. V dalším postupu se všechny tyto nerovnice $\mathcal{A}^i\mathbf{x} \leq \mathcal{B}^i$ z aktivní množiny (i značí odpovídající řádek $\mathcal{A}\mathbf{x} \leq \mathcal{B}$) po jedné procházejí a hledá se jejich původ. Jsou jen dva případy, kterého původu může omezení být.

V prvním případě dané omezení zaručuje dosažitelnost řešení (angl. *feasibility*). Tato omezení vycházejí z rovnice

$$\hat{\mathbf{G}}\mathbf{U}^* \leq \hat{\mathbf{W}} + \hat{\mathbf{S}}\mathbf{x}. \tag{75}$$

Detaily, jak tato rovnice vznikla jsou v kapitole o algoritmu explicitního prediktivního regulátoru. Pokud nerovnice $\mathcal{A}^i\mathbf{x} \leq \mathcal{B}^i$ slouží k dosažitelnosti řešení, tak musí být rovna některému řádku z maticové nerovnosti

$$(\hat{\mathbf{G}}\mathbf{M} - \hat{\mathbf{S}})\mathbf{x} \leq \hat{\mathbf{W}} - \hat{\mathbf{G}}\mathbf{N}. \tag{76}$$

Matice $\mathbf{M}$ a $\mathbf{N}$ při tom definují zákon řízení pro daný region.

Druhým případem jsou omezení, která zaručují optimalitu řešení (angl. *optimality*). Ty vycházejí ze vztahu

$$\tilde{\lambda}(\mathbf{x}) \geq \mathbf{0}. \tag{77}$$

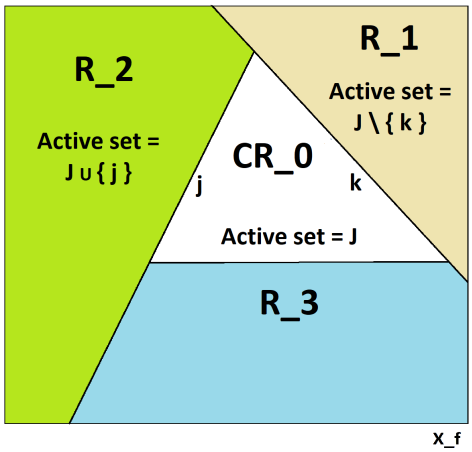
Aby nerovnost $\mathcal{A}^i\mathbf{x} \leq \mathcal{B}^i$ sloužila k zaručení optimality, tak musí být rovna některému řádku z maticové nerovnosti

$$(\tilde{\mathbf{G}}\mathbf{H}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}})^{-1}(\tilde{\mathbf{S}} + \tilde{\mathbf{G}}\mathbf{H}^{-1}\mathbf{F})\mathbf{x} \leq -(\tilde{\mathbf{G}}\mathbf{H}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}})^{-1}\tilde{\mathbf{W}}. \tag{78}$$

Po určení, jakého omezení nerovnost $\mathcal{A}^i\mathbf{x} \leq \mathcal{B}^i$ je, nastane jedna z následujících možností. Pokud omezení slouží k dosažitelnosti řešení, tak se daná nerovnost $\mathcal{A}^i\mathbf{x} \leq \mathcal{B}^i$ přidá do aktivní množiny. Naopak pokud je omezení pro zaručení optimality řešení, tak se z aktivní množiny daná nerovnost odstraní.

Ať už nastane jeden nebo druhý případ, tak vznikne nový region s takto nově definovanou aktivní množinou. Pro něj se následně vypočítá optimální zákon řízení a postup se opakuje. To znamená, že se opět po jedné prochází nerovnosti z jeho aktivní množiny a hledá se jejich původ. V případě, že se nalezne již známý region, tak algoritmus okamžitě pokračuje s následující nerovností v pořadí. Tento postup se neustále opakuje dokud není prozkoumán celý prostor stavů systému.

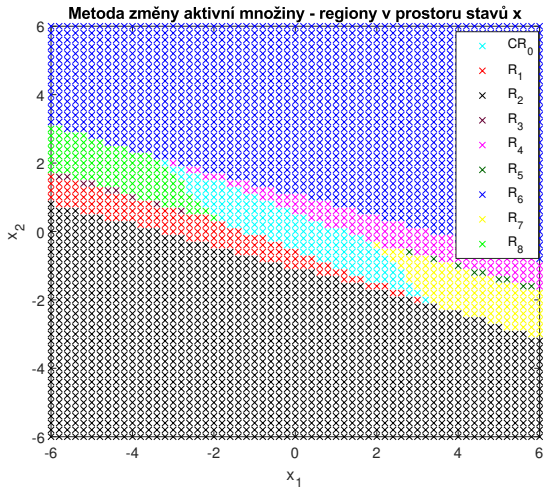
Uvedený postup dobře ilustruje následující obrázek.



Obrázek 9: Metoda změny aktivní množiny - ilustrační příklad

Jak je vidět, tak kritický region CR_0 definují 3 nerovnosti, které odpovídají aktivní množině $\mathcal{J}$. Nerovnost k přitom slouží k zaručení optimality řešení. Dojde tedy k jejímu odstranění z aktivní množiny, čímž vznikne nová aktivní množina $\mathcal{J}\backslash\{k\}$. Tato nová aktivní množina pak definuje region R_1. Další nerovnost j původní aktivní množiny regionu CR_0 slouží k dosažitelnosti řešení. Dojde tedy k jejímu přidání do aktivní množiny $\mathcal{J} \cup \{j\}$, díky čemuž vznikne nová aktivní množina definující nový region R_2. Stejným způsobem se pro regiony R_1 a R_2 budou hledat původy omezení z jejich aktivních množin. Budou se tak nalézat nové regiony, dokud se neprozkoumá celý prostor stavů řízeného systému.

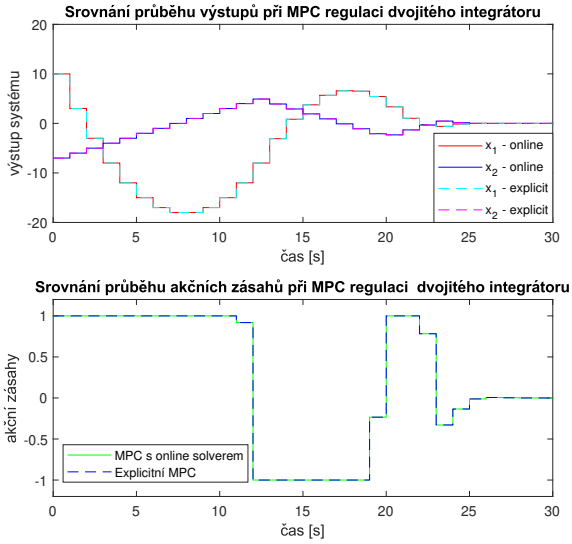
Po aplikaci tohoto algoritmu na systém dvojitého integrátoru vyjdou v prostoru stavů následující regiony.



Obrázek 10: Metoda změny aktivní množiny - nalezené regiony

Jak je vidět, tak bylo nalezeno 9 regionů, což je stejný počet jako pro první metodu sekvenčního prohledávání stavového prostoru.

Zbývá vyhodnotit průběhy simulací chování uzavřené smyčky při řízení dvojitého integrátoru pomocí MPC ve standardním tvaru a pomocí explicitního MPC s takto nalezenými regiony. Výsledek zachycuje tento obrázek.



Obrázek 11: Metoda změny aktivní množiny - srovnání simulací s MPC s on-line solverem

Je zřejmé, že při regulaci pomocí MPC ve standardní verzi a pomocí explicitního prediktivního regulátoru s takto nalezenými regiony bylo dosaženo stejného výsledku. To je dobrý předpoklad pro to, aby tuto metodu prohledávání prostoru stavů řízeného systému bylo možné použít i v praxi.

4.4 Určení regionu příslušícího aktuálnímu stavu systému

Po nalezení všech n_r regionů v prostoru stavů řízeného systému a určení všech k nim přidruženým zákonům řízení je ale potřeba řešit další klíčový problém. Ten nastává až při on-line běhu explicitního prediktivního regulátoru. V tu chvíli je totiž nezbytné pro aktuální stav systému $\mathbf{x}$ nalézt region R_i, do kterého aktuální stav patří a posléze vypočítat funkční hodnotu příslušné lineární funkce ve tvaru

$$\mathbf{u}(\mathbf{x}) = \mathbf{M}_i\mathbf{x} + \mathbf{N}_i, \tag{79}$$

jež udává výsledný akční zásah, který se aplikuje do řízeného systému.

To se ukazuje jako klíčová úloha pro snížení výpočetní náročnosti celého explicitního prediktivního regulátoru. V případě, že by bylo nutné řídit systém s hodně stavy a velkým počtem omezení, tak by to vedlo na velké množství regionů v prostoru stavů. Vypočtení optimálního akčního zásahu, který by se měl aplikovat do systému, by tak mohlo trvat dokonce delší dobu než u MPC regulátoru ve standardní verzi s on-line solverem.

Je tedy vhodné dobře rozmyslet, jakou strategii pro hledání regionů v prostoru stavů, kam aktuální stav systému patří, zvolit. V této podkapitole budou představeny 2 přístupy k tomuto problému. První z nich používá sekvenční prohledávání stavového prostoru a v druhém přístupu se využívá binární vyhledávací strom.

4.4.1 Sekvenční prohledávání

Tato metoda pro nalezení regionu příslušícího aktuálnímu stavu systému je přímočará a nejvíce jednoduchá na implementaci. Předpokládá se, že jsou k dispozici n_r regionů, které se našly prostřednictvím některé z metod představených v předchozí podkapitole. V dalším postupu je nutné tyto regiony, respektive nerovnice, které je definují, seřadit sekvenčně za sebou. Jako nejjednodušší se jeví regiony za sebou poskládat tak, jak byly postupně objevovány některou z uvedených metod.

Pro nalezení výsledného akčního zásahu se v každém kroku algoritmu explicitního prediktivního regulátoru při sekvenčním prohledávání stavového prostoru používá následující postup. Vstupním parametrem je aktuální stav systému (popřípadě jeho rozšířená verze viz kapitola 2.7). Dále je nutné sekvenčně procházet všechny známé regiony a určovat, do jakého z nich aktuální stav systému patří. Konkrétně se procházejí aktivní množiny všech známých regionů. Postupně se tedy vyhodnocuje sada nerovností ve tvaru

$$\mathcal{A}^i\mathbf{x} \leq \mathcal{B}^i \tag{80}$$

při dosazení aktuálního stavu systému $\mathbf{x}$ a to pro každý region R_i. Pokud po dosazení stavu systému do vztahu (80) všechny nerovnosti platí, tak je prohlášeno, že aktuální stav systému patří do regionu R_i. Procházení dalších regionů již není potřeba. Jako poslední stačí pro určený region R_i vyhodnotit příslušnou lineární funkci se stavem systému $\mathbf{x}$ definující výsledný optimální akční zásah ve tvaru

$$\mathbf{u}(\mathbf{x}) = \mathbf{M}_i\mathbf{x} + \mathbf{N}_i. \tag{81}$$

Vypočtená číselná hodnota $\mathbf{u}(\mathbf{x})$ se považuje za výstup regulátoru a je aplikována do řízené soustavy.

Tento postup se opakuje v každém kroku algoritmu explicitního MPC se sekvenčním prohledáváním stavového prostoru pro nalezení příslušného regionu. Je zřejmé, že v případě velkého množství regionů by se muselo vyhodnocovat obrovské množství nerovnic. V nejhorším případě by regulátor musel vyhodnotit všechny nerovnice, které definují všechny známé regiony. Vzhledem k tomuto faktu by nebylo vhodné tuto metodu používat v praxi.

Algoritmus na prohledávání prostoru stavů k určení regionu, kterému přísluší aktuální stav systému $\mathbf{x}$, lze strukturovaně zapsat takto.

Algorithm 1 Sekvenční prohledávaní stavového prostoru pro určení regionu

1: Proveď inicializaci $i \leftarrow 1$.
2: **while** $\mathbf{x} \notin R_i$ **and** $i \leq n_r$ **do**
3: Proveď inkrementaci $i \leftarrow i + 1$.
4: **end while**
5: **if** $i = n_r + 1$ **then**
6: Region nenalezen - nedosažitelný problém. KONEC.
7: **end if**
8: Vyhodnoť zákon řízení příslušící nalezenému regionu ve tvaru $\mathbf{u}(\mathbf{x}) = \mathbf{M}_i\mathbf{x} + \mathbf{N}_i$.

4.4.2 Metoda používající binární vyhledávací strom

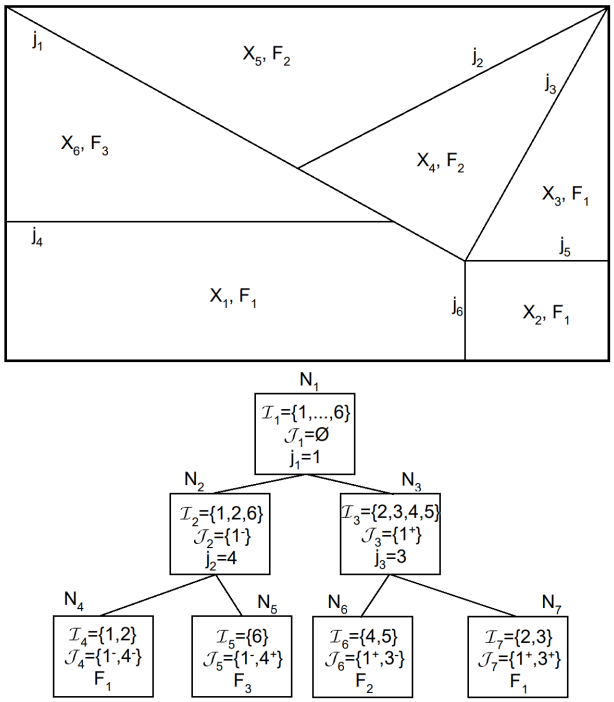
Tato metoda prohledávání stavového prostoru pro určení regionu odpovídajícímu aktuálnímu stavu systému byla převzata ze zdroje [12]. Jak již název napovídá, tak zde je pro prohledávání stavového prostoru nutné vytvořit off-line binární vyhledávací strom. Ten bude následně explicitní MPC on-line prohledávat v každém kroku algoritmu za účelem určení regionu odpovídajícímu stavu systému. Ukazuje se, že zde pro nalezení příslušného regionu je zapotřebí v nejhorším případě vyhodnotit tolik nerovnic, jaká je hloubka vytvořeného binárního stromu. Při správné implementaci této metody je výsledkem mnohonásobně kratší výpočetní náročnost regulátoru než u naivní metody sekvenčního prohledávání.

Tato metoda opět vychází ze znalosti všech regionů v prostoru stavů systému. Ty budou pro tuto kapitolu značeny $R_1, R_2, ..., R_{n_r}$, kde n_r odpovídá počtu známých regionů. Dále je nezbytná znalost odpovídajících lineárních zákonů řízení $F_1, F_2, ..., F_K$, kde K reprezentuje počet unikátních zákonů řízení. Obecně totiž může platit $K \leq n_r$ vzhledem k tomu, že některým regionům může příslušit stejný zákon řízení. V dalším postupu je nutné definovat sadu unikátních nadrovin definující všechny známé regiony jako $\mathbf{a}_j\mathbf{x} = b_j$ (díky rovnosti se jedná přímo o hranice regionů) pro $j = 1, 2, ..., L$, kde L značí počet těchto unikátních nadrovin. V algoritmu se totiž bude pracovat s následující hodnotou

$$d_j(\mathbf{x}) = \mathbf{a}_j\mathbf{x} - b_j, \tag{82}$$

kde vyjde $d_j(x) \in \mathbb{R}$.

Dále se zavede indexová reprezentace $\mathcal{J}$ pro množinu nadrovin $\mathbf{a}_j\mathbf{x} = b_j$. Ta bude obsahovat množinu indexů nadrovin j společně se znaménkem $+$ nebo $-$ určující kladnou nebo zápornou hodnotu d_j. Vysvětleno to bude na následujícím jednoduchém případu. Je-li nějaký region definován jako $\mathcal{J} = \{2^-, 4^-, 6^+\}$, tak pro něj platí, že po dosazení $\mathbf{x}$ z daného regionu do rovnic d_2 a d_4 vyjdou záporné hodnoty d a po dosazení $\mathbf{x}$ z daného regionu do rovnice s indexem $j = 6$ vyjde hodnota d_6 kladná. Nicméně pomocí této reprezentace lze vyjadřovat i celou množinu regionů. Například $\mathcal{J} = \{1^+\}$ může reprezentovat celou řadu regionů, pro které se dostane kladná hodnota d_1 po dosazení příslušného $\mathbf{x}$ do rovnice s indexem 1. Zavede se pro to další značení $\mathcal{I}(\mathcal{J})$ a to právě pro množinu přípustných regionů pro zadané $\mathcal{J}$. Je-li takto definovaná množina přípustných regionů, tak lze definovat i množinu odpovídajících zákonů řízení $\mathcal{F}(\mathcal{I})$. Ta definuje množinu všech zákonů řízení přípustných pro danou množinu regionů. Zavedené značení je dobře demonstrováno na následujícím obrázku.



Obrázek 12: Binární vyhledávací strom - příklad [12]

Cílem této metody je zkonstruovat takový binární vyhledávací strom, že se pomocí vstupního vektoru $\mathbf{x}$ (stav systému) v každém uzlu N_k vyhodnotí lineární funkce (82) a určí se zda je výsledek d_j kladný nebo záporný. V případě záporné hodnoty přejde algoritmus do levého následníka aktuálního uzlu. Naopak je-li hodnota d_j kladná, tak se přejde do pravého následníka uvažovaného uzlu. Tímto způsobem se bude procházet strom od kořene až do listu, ve kterém bude přípustný pouze jeden zákon řízení F_k. Snaha je při tom vytvořit strom s co nejmenší hloubkou a s co nejmenším počtem uzlů. To povede na to, že pro určení regionu bude potřeba vyhodnotit co nejmenší počet rovnic (82).

Každý uzel stromu N_k bude při tom definován pomocí dvojice $(\mathcal{I}_k, \mathcal{J}_k)$. $\mathcal{J}_k$ udává indexovou reprezentaci nadrovin, které jsou platné pro daný uzel, a pro $\mathcal{I}_k$ platí $\mathcal{I}_k = \mathcal{I}(\mathcal{J}_k)$. Symbolem $\mathcal{U}$ se dále bude označovat množina prozatím neprozkoumaných uzlů. Prozkoumaný uzel, který není listem, bude dále obsahovat index j_k odkazující na příslušnou rovnici (82), která se má v daném uzlu vyhodnotit. Listový uzel bude namísto indexu j_k obsahovat

odkaz na zákon řízení F_k, což bude vlastně výsledný zákon řízení, pomocí kterého se spočte výsledná hodnota akčního zásahu regulátoru.

Pro úsporu zápisu se nyní zavede značení $\pm$, které se použije v případě, že daný vztah platí pro obě znaménka + i −. Ve zdroji [12] je dokázáno, že platí

$$\mathcal{I}(\mathcal{J} \cup j^{\pm}) \subseteq (\mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^{\pm})). \tag{83}$$

Lze totiž ukázat, že když pro region R_i platí $R_i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+)$ a při tom současně $R_i \notin \mathcal{I}(\mathcal{J} \cup j^+)$, pak regionem R_i prochází nadrovina (resp. rovnice) j. Příkladem může být region, pro který platí $R_i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$. Stejného výsledku by se dosáhlo, když by bylo prohozené j^+ a j^-.

Při prozkoumávání uzlu stromu je hlavním cílem pro jeho oba následníky co nejvíce zmenšit počet přípustných zákonů řízení $|F_k|$. Matematicky to lze formulovat tak, že pro uzel $N_k = (\mathcal{I}_k, \mathcal{J}_k)$ je chtěné vybrat rovnici (resp. nadrovinu) j_k, pro kterou bude platit

$$j_k = \arg\min_j \max(|\mathcal{F}(\mathcal{I}_k^+)|, |\mathcal{F}(\mathcal{I}_k^-)|), \tag{84}$$

kde $\mathcal{I}_k^{\pm} = \mathcal{I}(\mathcal{J}_k \cup j^{\pm})$. To nicméně vyžaduje výpočet $\mathcal{I}_k^{\pm}$ pro každou nadrovinu j. Vztah (83) ukazuje vhodnou aproximaci tohoto výpočtu $\mathcal{I}_k^{\pm}$ jako $\mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^{\pm})$. Přesnou hodnotu $\mathcal{I}_k^{\pm}$ lze následně získat pro každý region $R_i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$ vyřešením dvou lineárních programů ve tvaru

$$\min_{\mathbf{x} \in R_i} \pm d_j(\mathbf{x}). \tag{85}$$

Aproximaci lze tedy využít k selekci několika vhodných nadrovin (resp. kandidátů). Pro získání přesného řešení je posléze nutné vypočítat minimalizace (85), ovšem počet těchto minimalizací, která je potřeba vyřešit, je již o mnoho menší.

V tuto chvíli je již možné představit **algoritmus pro konstrukci binárního vyhledávacího stromu**. Prvně je zapotřebí určit množiny $\mathcal{I}(j^+)$ a $\mathcal{I}(j^-)$ pro každou rovnici $j \in \{1, ..., N\}$. To může být realizováno například vyřešením $2Ln_r$ minimalizačních úloh (85). Jako kořen stromu se dále zvolí $N_1 = (\mathcal{I}_1, \mathcal{J}_1) = (\{1, ..., n_r\}, \emptyset)$. Uzel N_1 se posléze přidá do množiny prozatím neprozkoumaných uzlů $\mathcal{U}$. V tu chvíli je dokončena inicializační část algoritmu.

Další postup se neustále opakuje pro všechny uzly z množiny $\mathcal{U}$, dokud tato množina není prázdná. Prvním krokem je odebrat jakýkoliv uzel N_k z množiny $\mathcal{U}$. Tento uzel bude v aktuálním kroku prozkoumáván. Dále je potřeba pro daný uzel N_k vypočítat aproximace

$$\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^{\pm}) \tag{86}$$

pro všechny rovnice j. Poté se nadroviny j seřadí dle hodnoty

$$\max(|\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^+))|, |\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^-))|) \tag{87}$$

od nejmenší po největší.

V dalším kroku se uvažuje pouze prvních n_j nadrovin ze sařazeného listu z předchozího kroku, tj. nadroviny s minimální hodnotou (87). Je nutné si uvědomit, že těchto nadrovin s minimální hodnotou může být více. Pro těchto n_j nadrovin se následně vypočte přesné řešení $\mathcal{I}_k^{\pm} = \mathcal{I}(\mathcal{J}_k \cup j^{\pm})$. To je realizováno přes počítání lineárních programů ve formě (85) pro každý region, pro který platí $R_i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$. Následně již stačí pouze vybrat finální rovnici j_k, podle které se bude aktuální uzel N_k dělit. Ta se určí jako

$$j_k = \arg\min_j \max(|\mathcal{F}(\mathcal{I}_k^+)|, |\mathcal{F}(\mathcal{I}_k^-)|). \tag{88}$$

Po této operaci dojde k asociaci $N_k \leftarrow j_k$.

V dalším kroku se definují pro aktuální uzel dva následovníci $N^{\pm} \leftarrow (\mathcal{I}_k^{\pm}, \mathcal{J}_k \cup j^{\pm})$. Pro oba z nich je nutné rozhodnout, zda platí $|\mathcal{F}(\mathcal{I}^{\pm})| > 1$. V pozitivním případě se uzel $N^{\pm}$ přidá do množiny $\mathcal{U}$. V negativním případě se jedná o list a přiřadí se k němu příslušný zákon řízení. Matematicky to lze zapsat jako $N^{\pm} \leftarrow \mathcal{F}(\mathcal{I}^{\pm})$.

Tento algoritmus se neustále opakuje pro každý uzel stromu dokud množina $\mathcal{U}$ není prázdná. V opačném případě, kdy $\mathcal{U} = \emptyset$, algoritmus končí a binární vyhledávací strom lze považovat za dokončený.

Algoritmus pro konstrukci binárního vyhledávacího stromu lze tedy zapsat strukturovaně takto.

Algorithm 2 Konstrukce binárního vyhledávacího stromu

1: Urči množiny $\mathcal{I}(j^+)$ a $\mathcal{I}(j^-)$ pro každou rovnici $j \in \{1, ..., N\}$.
2: Kořen stromu inicializuj jako $N_1 \leftarrow (\{1, ..., n_r\}, \emptyset)$.
3: Množinu neprozkoumaných uzlů inicializuj jako $\mathcal{U} \leftarrow \{N_1\}$.
4: Vyber jakýkoliv neprozkoumaný uzel $N_k \in \mathcal{U}$ a proveď $\mathcal{U} \leftarrow \mathcal{U} \backslash N_k$.
5: Vypočti aproximace $\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^{\pm})$ pro všechny j a seřaď nadroviny dle velikosti $\max(|\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^+))|, |\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^-))|)$ od nejmenší po největší.
6: Vypočti přesné řešení $\mathcal{I}_k^{\pm} = \mathcal{I}(\mathcal{J}_k \cup j^{\pm})$ pro prvních n_j prvků se stejnou hodnotou ze seznamu z kroku 5. To lze provést vyřešením lineárních programů (85) pro každý region $R_i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$. Na základě výsledku vyber j_k jako $j_k = \arg\min_j \max(|\mathcal{F}(\mathcal{I}_k^+)|, |\mathcal{F}(\mathcal{I}_k^-)|)$.
7: Dokonči uzel přiřazením $N_k \leftarrow j_k$ a přiřaď mu 2 následníky jako $N^{\pm} \leftarrow (\mathcal{I}_k^{\pm}, \mathcal{J}_k \cup j^{\pm})$.
8: **if** $|\mathcal{F}(\mathcal{I}^{\pm})| > 1$ **then**
9: Přidej $N^{\pm}$ do $\mathcal{U}$.
10: **else**
11: $N^{\pm}$ je listový uzel. Proveď asociaci $N^{\pm} \leftarrow \mathcal{F}(\mathcal{I}^{\pm})$
12: **end if**
13: **if** $\mathcal{U} = \emptyset$ **then**
14: Jdi na řádek 4 algoritmu.
15: **else**
16: KONEC algoritmu.
17: **end if**

V okamžiku, kdy je binární vyhledávací strom vytvořený, tak lze implementovat algoritmus na jeho prohledávání do explicitního MPC regulátoru pro on-line určení regionu, do kterého aktuální stav systému patří. **Algoritmus pro on-line určení příslušného regionu pomocí binárního vyhledávacího stromu** vypadá následovně. Uvažuje se aktuální stav systému jako vstup explicitního MPC regulátoru. Začne se s uzlem N_1, pro který se vyhodnotí jemu příslušný vztah $d_j(\mathbf{x}) = \mathbf{a}_j\mathbf{x} - b_j$. V případě záporné hodnoty d se algoritmus přesune do levého následovníka aktuálního uzlu. Naopak je-li hodnota d kladná, tak se algoritmus přesune do pravého následovníka aktuálního uzlu. Pro nový uzel se opět vyhodnotí jemu příslušný vztah $d_j(\mathbf{x}) = \mathbf{a}_j\mathbf{x} - b_j$ a na základě výsledné hodnoty se určí, do jakého uzlu se algoritmus přesune.

Uvedený postup se neustále opakuje, dokud se v binárním vyhledávacím stromu nedojde do listového uzlu. Tomu přísluší právě jeden zákon řízení definující výsledný optimální akční zásah. Stačí tedy vyhodnotit odpovídající lineární funkci se stavem systému $\mathbf{x}$ definující výsledný optimální akční zásah ve tvaru

$$\mathbf{u}(\mathbf{x}) = \mathbf{M}_i\mathbf{x} + \mathbf{N}_i. \tag{89}$$

Vypočtená číselná hodnota $\mathbf{u}(\mathbf{x})$ se považuje za výstup regulátoru a je aplikována do řízené soustavy.

Tento způsob prohledávání binárního vyhledávacího stromu musí explicitní prediktivní regulátor provést v každém kroku algoritmu s nově příchozí hodnotou aktuálního stavu systému $\mathbf{x}$.

Algoritmus pro on-line hledání regionu pomocí binárního vyhledávacího stromu lze strukturovaně zapsat následujícím způsobem.

Algorithm 3 Prohledávání binárního vyhledávacího stromu

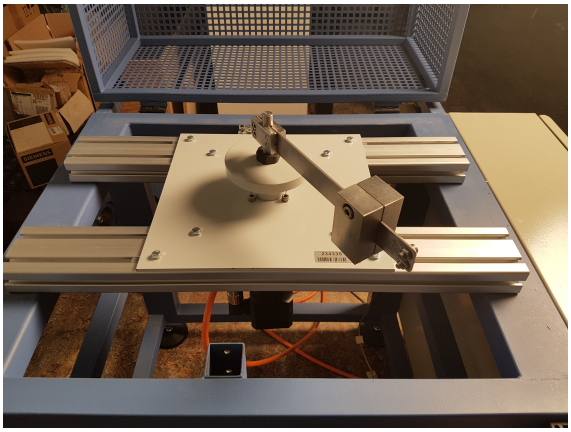
1: Nechť je aktuální uzel N_k kořenem stromu.
2: **while** N_k není listovým uzlem **do**
3: Vyhodnoť rovnici $d_j(\mathbf{x}) = \mathbf{a}_j\mathbf{x} - b_j$ příslušící uzlu N_k.
4: Nechť N_k je jedním z jeho následovníků určený na základě znaménka d_j.
5: **end while**
6: Vypočti výsledný akční zásah $\mathbf{u}$ na základě rovnice $\mathbf{u}(\mathbf{x}) = \mathbf{M}_i\mathbf{x} + \mathbf{N}_i$, která přísluší nalezenému listovému uzlu.

5 Popis řízeného mechatronického systému

V praktické části diplomové práce budou použity popsané algoritmy prediktivního řízení pro regulaci reálného mechatronického sytému. Tato úvodní kapitola praktické sekce práce bude sloužit k prvotnímu seznámení se s touto mechatronickou soustavou. Bude uveden i její obrázek společně s několika jejími důležitými parametry, které soustavu dobře charakterizují. Další podkapitola bude věnována nalezení modelu systému. Pro prediktivní řízení je totiž zapotřebí získat co nejpřesnější model řízené soustavy, aby regulace probíhala co nejlépe. Po nalezení modelu proběhne ještě jeho analýza v časové a frekvenční oblasti. Bude tedy vykreslena přechodová a frekvenční charakteristika, aby byly dobře patrné vlastnosti systému.

5.1 Popis systému

Konkrétně se jedná o mechatronický stend pohonu s pružnou zátěží. Detailní popis soustavy lze nalézt v článku [13]. Tento elektromechanický systém tvoří elektrický pohon (synchronní motor poháněný servozesilovačem), pružná spojka, setrvačník a odjímatelné pohyblivé rameno se zátěží. Toto zařízení disponuje jedním stupněm volnosti, který umožňuje rameni se zátěží pohybovat se kolem své vertikální osy. Systém je zachycen na tomto snímku.



Obrázek 13: Reálný mechatronický systém pro který bude navrhováno MPC

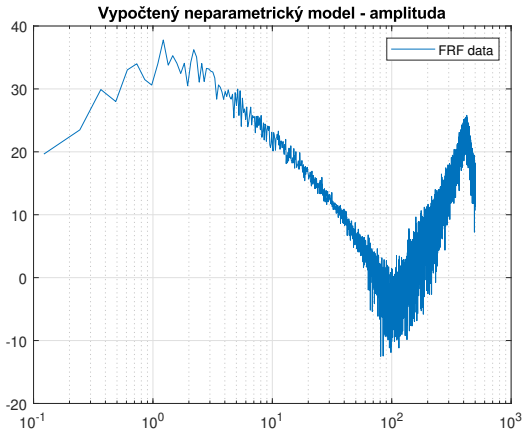
Následující tabulka poskytuje hodnoty vybraných parametrů pohyblivé části systému.

Délka ramene	0.235 m
Hustota materiálu ramene	8030 $kg \cdot m^{-3}$
Youngův modul materiálu	190,295,301,291.7 $N \cdot m^{-2}$
Plocha průřezu	8.9274 $\cdot 10^{-5} m^2$
Hmotnost užitečného zatížení	1.049 kg
Moment setrvačnosti rotační osy připojené k motoru	0.0024 $kg \cdot m^2$

Důležité je také upozornit, že pro pokusy realizované v této práci nebude uvažována na systému zátěž. Sledovat se tedy bude pouze poloha a rychlost motoru.

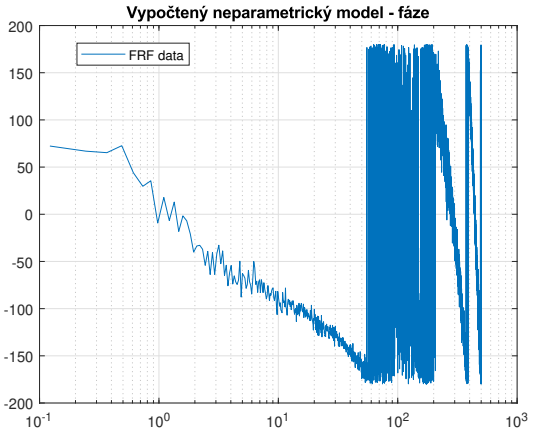
5.2 Model systému

Tato podkapitola bude věnována nalezení modelu řízeného systému. K tomu je nejdříve zapotřebí na základě dat z identifikačního experimentu nalézt odhad bodů frekvenční charakteristiky - neparametrický model. Výsledek je na následujících obrázcích. První z nich zachycuje průběh amplitudy.



Obrázek 14: Neparametrický model - body frekvenčního přenosu - amplituda

Na druhém obrázku je vidět průběh fáze.



Obrázek 15: Neparametrický model - body frekvenčního přenosu - fáze

Relevantní data jsou od přibližně od 2 Hz do konce (Nyquistova frekvence 500 Hz pro periodu 1 ms). Na nižších frekvencích se projevuje vliv nelineárního tření. Na vysokých frekvencích je dobře vidět hodnota rezonanční a antirezonanční frekvence, což by se dalo interpretovat fyzikálně jako vliv pružného módu spojení motor-zátěž.

Po získání bodů frekvenční charakteristiky z identifikačního experimentu je dalším krokem nalézt přenosovou funkci z požadovaného momentu motoru na jeho rychlost. K tomu se využilo System identification toolboxu v Matlabu. Jeho výstupem byl přenos v této podobě

$$P_{tf} = z^{-2} \cdot \frac{0.03559z^{-1} + 2.484z^{-2} - 0.4734z^{-3} - 2.779z^{-4} + 3.272z^{-5}}{1 + 0.3468z^{-1} - 0.7115z^{-2} - 0.5901z^{-3} - 0.02353z^{-4} + 0.007582z^{-5} + 0.01903z^{-6}}. \tag{90}$$

K prediktivnímu řízení je nicméně zapotřebí mít k dispozici diskrétní stavový model soustavy ve tvaru

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \tag{91}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k. \tag{92}$$

Přenosovou funkci na stavový model lze převést jednoduše za pomocí Matlabu příkazem `ss`. Stavový popis bude vytvořen tak, aby měl 1 vstup a 2 výstupy. Vstup modelu bude odpovídat požadovanému proudu/momentu motoru. První výstup bude reprezentovat polohu motoru a druhý výstup jeho rychlost. Výsledný stavový model řízeného systému, který se použije k návrhu prediktivního řízení, tedy vyšel v následující podobě

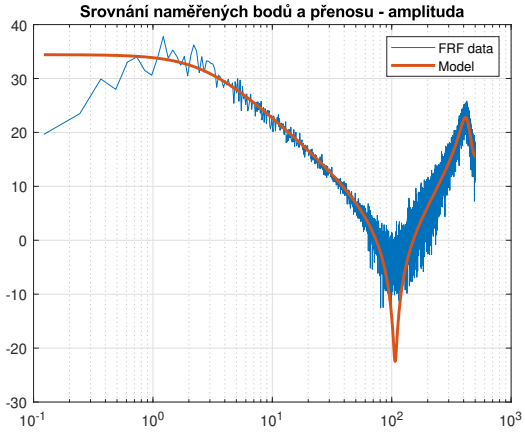
$$\mathbf{x}_{k+1} = \begin{bmatrix} 1.7677 & -1.0623 & 0.0363 & -0.0882 & -0.0249 & -0.0493 & 0.0296 & 0.0056 \\ 1.9495 & 0.5329 & 6.1676 & -0.0696 & -0.0197 & -0.0389 & 0.0234 & 0.0044 \\ -0.4516 & 0.1442 & 0.3194 & 0.2475 & 0.0057 & 0.0113 & -0.0068 & -0.0013 \\ 0.1974 & -0.2282 & -5.5738 & -1.0669 & 0.1377 & -0.0236 & 0.0142 & 0.0027 \\ 5.2586 & -3.4437 & 0.3452 & -3.6952 & -1.7674 & 0.3523 & 0.0050 & 0.0009 \\ -4.7112 & 3.4155 & -0.1374 & 3.6508 & 0.7149 & 0.6267 & 0.0529 & 0.0065 \\ -0.3683 & -0.4394 & 0.1215 & -0.2724 & 0.1807 & 1.3975 & 0.2442 & 0.0053 \\ 0.0636 & 0.0624 & -0.0225 & 0.0698 & -0.0160 & 0.0233 & -0.9920 & -0.0033 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0.0139 \\ 0.1034 \\ 0.4859 \\ 0.0568 \\ 0.0046 \\ 0.0059 \\ 0.0008 \\ 0.0001 \end{bmatrix} \mathbf{u}_k. \tag{93}$$

$$\mathbf{y}_k = \begin{bmatrix} -0.0003 & -0.0002 & 0.0001 & -0.0002 & -0.0001 & -0.0002 & -0.0000 & 0.0000 \\ -0.0194 & -0.0230 & 0.0060 & -0.0126 & 0.0058 & 0.0805 & -0.0733 & -0.0547 \end{bmatrix} \mathbf{x}_k, \tag{94}$$

kde perioda vzorkování je 1 ms.

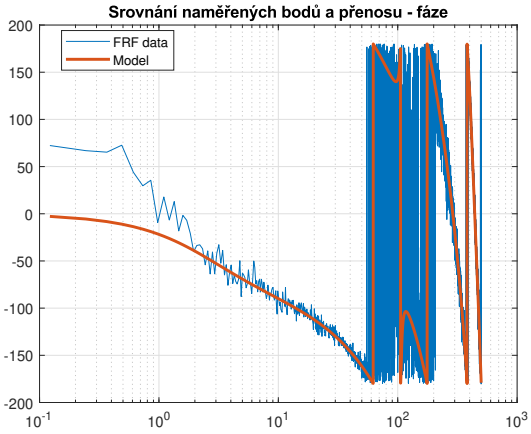
5.3 Analýza modelu

Je vidět, že řád systému vyšel relativně vysoký a to 8. Tato kapitola bude věnována analýze modelu systému ve frekvenční a časové oblasti. Jako první bude vykreslena frekvenční charakteristika systému. Současně budou vykresleny i body frekvenční charakteristiky neparametrického modelu, aby bylo dobře patrné, zda model odpovídá naměřeným datům. První obrázek zachycuje průběh amplitudy.



Obrázek 16: Srovnání frekvenčních charakteristik - naměřené body, model - amplituda

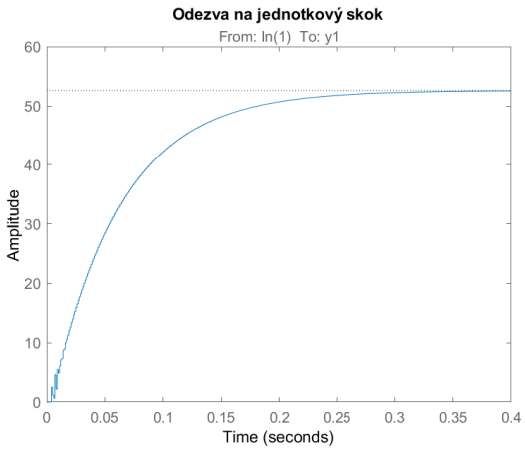
Zde je vidět vývoj fáze.



Obrázek 17: Srovnání frekvenčních charakteristik - naměřené body, model - fáze

U obou obrázků je vidět, že skutečně nalezený model dobře kopíruje data z identifikačního experimentu. Konkrétně u průběhu amplitudy je patrná stejná hodnota antirezonanční frekvence. Jak již bylo řečeno, tak na nízkých frekvencích si průběhy zcela neodpovídají a to z toho důvodu, že naměřená data jsou relevantní od cca 2 Hz do konce (Nyquistova frekvence 500 Hz pro periodu 1 ms). Na nižších frekvencích se projevuje vliv nelineárního tření.

Pro lepší seznámení se se systémem zde ještě bude uvedena jeho odezva na jednotkový skok. Tu zachycuje následující obrázek.



Obrázek 18: Odezva na jednotkový skok pro nalezený přenos z momentu na rychlost

Je patrné, že přechodový děj je téměř plynulý. Malé kmitání se vyskytlo pouze na začátku simulace. Z obrázku je vidět, že systém je poměrně rychlý, jelikož doba ustálení činní 0.4 vteřiny.

6 Řízení reálného systému MPC s on-line solverem

V této kapitole proběhne návrh a otestování MPC regulátoru ve standardním tvaru s on-line solverem navrženého pro reálný mechatronický systém popsaný výše. Návrh regulátoru bude realizován v softwaru Matlab, kde proběhnou i nezbytné simulace pro otestování funkčnosti a kvality regulátoru. Konkrétně se využije jeho nástroje Simulink, který slouží právě k simulačním účelům. Poté se již přistoupí k řízení skutečného systému a to pomocí řídicího programu REXYGEN Studio.

REXYGEN Studio je grafický nástroj určený pro tvorbu řídicích algoritmů v reálném čase s podporou rozsáhlé knihovny funkčních bloků systému REXYGEN. Jakýkoli algoritmus vyvinutý v REXYGEN Studiu lze okamžitě zkompilovat a stáhnout do libovolného cílového zařízení. Po úspěšné kompilaci a stažení algoritmu do cílového zařízení je možné přepnout REXYGEN Studio do režimu Watch, ve kterém lze pozorovat nebo upravovat všechny parametry a proměnné všech funkčních bloků. Kompletní dokumentaci tohoto SW lze najít na *www.rexygen.com*.

Kapitola bude koncipována následovně. Nejdříve proběhne samotný návrh MPC regulátoru ve standardní formě. Poté zde budou uvedena schémata zapojení ze Simulinku a REXYGENu, která se použila k otestování navrženého MPC regulátoru. V další podkapitole budou vykresleny průběhy důležitých veličin regulační smyčky, které byly získány během simulacích. Poslední část této kapitoly bude věnována experimentům s navrženým MPC při řízení reálného stroje.

6.1 Návrh prediktivního regulátoru ve standardní formě

V této podkapitole budou uvedeny všechny parametry MPC regulátoru, při kterých bylo během simulací dosaženo nejlepších výsledků. Horizont predikcí byl zvolen jako $n_p = 30$. Horizont řízení má hodnotu $n_c = 3$. Je tedy skutečně vidět, že hodnota horizontu řízení byla zvolena jako 10 % z hodnoty horizontu predikcí, jak bylo doporučováno. Akční zásahy budou saturovány do rozmezí ± 0.1. Ukázalo se, že to je rozumná mez vzhledem k možnostem uvažovaného systému. Další omezení je vhodné klást na maximální rychlost motoru. Zvolilo se, že toto omezení bude ± 5. Vzhledem k počtu výstupů a vstupů bude váhová matice $\mathbf{Q}$ dimenze 2 a váhová matice $\mathbf{R}$ penalizující řízení dimenze 1. Tvary těchto váhových matic byly zvoleny následovně

$$\mathbf{Q} = \begin{bmatrix} 2000 & 0 \\ 0 & 0.1 \end{bmatrix}, \mathbf{R} = 500. \tag{95}$$

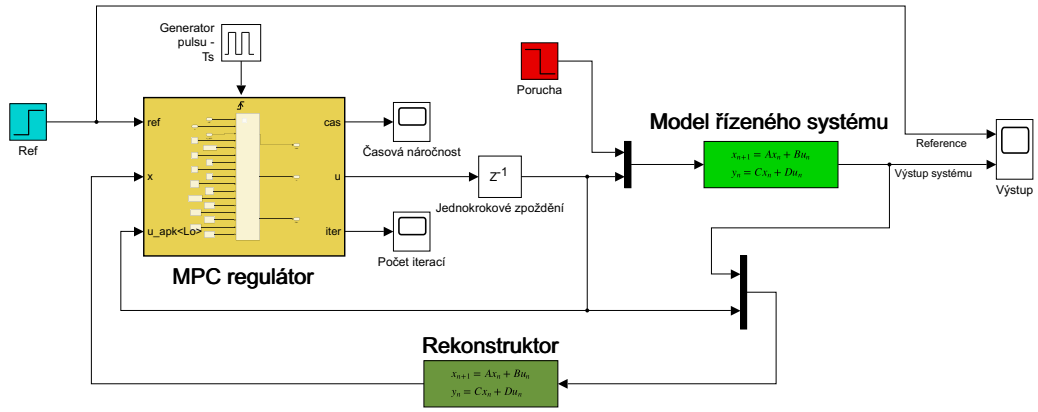
Je vidět, že důraz je kladen především na polohu motoru. Její průběh při regulaci by tedy měl být plynulejší.

Při návrhu byl použit algoritmus MPC pro dosažení nulové odchylky v ustáleném stavu popsaný v kapitole o prediktivním řízení ve standardní formě. Výsledné matice optimalizace pak vyšly

$$\mathbf{H} = \begin{bmatrix} 1011.2575 & -496.1707 & 65.3721 \\ -496.1707 & 1010.5947 & -439.7418 \\ 65.3721 & -439.7418 & 1219.1977 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} -0.4531 & 0.5326 & 1.4416 \\ 2.8304 & 3.1149 & 27.7447 \\ 8.6333 & 15.8725 & 129.0831 \\ 0.5313 & 1.5755 & 14.4835 \\ 0.0684 & 0.0571 & 1.0847 \\ 0.1554 & 0.1064 & 1.0876 \\ -0.0004 & 0.0096 & 0.0890 \\ -0.0004 & 0.0013 & 0.0118 \\ -481.8864 & -446.4055 & -3494.3272 \\ -1.8317 & -1.774 & -22.3203 \\ -500 & 0 & 0 \\ 80.4588 & 74.6823 & 844.828 \end{bmatrix}. \quad (96)$$

6.2 Schéma zapojení z Matlabu

K simulaci regulační smyčky s MPC regulátorem se využilo Simulinku, standardního grafického prostředí Matlabu pro simulaci dynamických systému. Použité schéma zapojení pro uvažovaný případ s MPC regulátorem ve standardní variantě s on-line solverem je na následujícím obrázku.



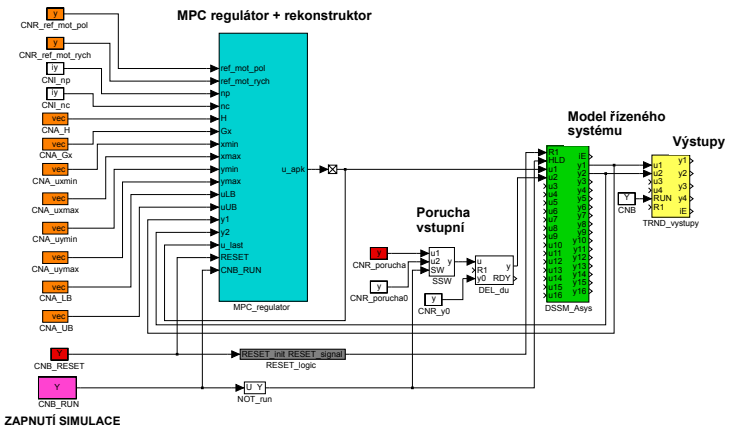
Obrázek 19: Schéma zapojení MPC s on-line solverem pro simulaci v Matlabu

Zcela vlevo modrý bloček reprezentuje vstupní referenci. Pro analýzu chování smyčky se bude používat skoková změna požadované polohy motoru. Dále se ve schématu nachází žlutý blok *Triggered subsystem*. Pomocí něho je implementován do zapojení MPC regulátor s on-line solverem. Do tohoto bloku lze vložit kód z Matlabu, který vykonává algoritmus prediktivní regulace. Součástí tohoto kódu je tedy i solver úloh kvadratického programování. MPC kompenzátor má zde 3 vstupy a to referenci, aktuální vektor stavu řízeného systému a poslední hodnotu aplikovaného řízení. Právě kvůli získání poslední hodnoty akčního zásahu se za MPC regulátorem nachází blok pro jednokrokové zpoždění. Výstupem regulátoru je čas, který regulátor potřeboval k vyřešení optimalizační úlohy, vygenerovaný akční zásah a počet iterací, který solver potřeboval k vyřešení optimalizační úlohy. Dále se ve schématu nachází model řízeného systému (světle zelený), do kterého je jako další vstup implementována vstupní porucha. Další důležitou částí zapojení je rekonstruktor (tmavě zelený). Ten, na základě vstupů a výstupů řízené soustavy, regulátoru předává hodnoty všech stavů modelu řízeného systému. Rekonstruktor je implementován tak, aby dokázal odhadnout i stav neměřitelné poruchy. Výstup řízené soustavy je v pravé části schématu společně s referencí

napojen do bloku *Scope* pro vykreslení.

6.3 Schéma zapojení z REXYGENu

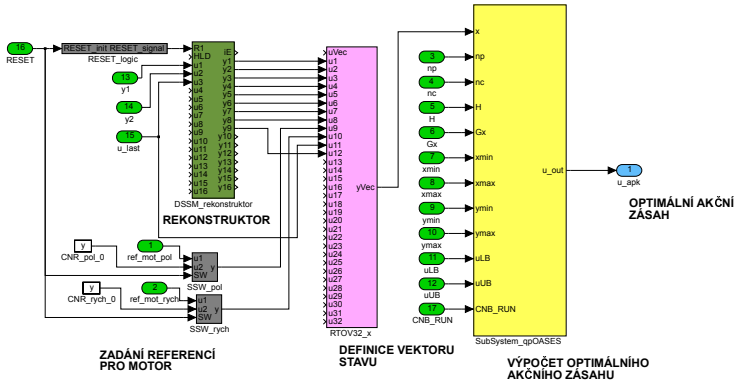
V softwaru REXYGEN se k simulacím uzavřené smyčky s MPC regulátorem používalo následující zapojení.



Obrázek 20: Schéma zapojení MPC s on-line solverem pro simulaci v REXYGENu

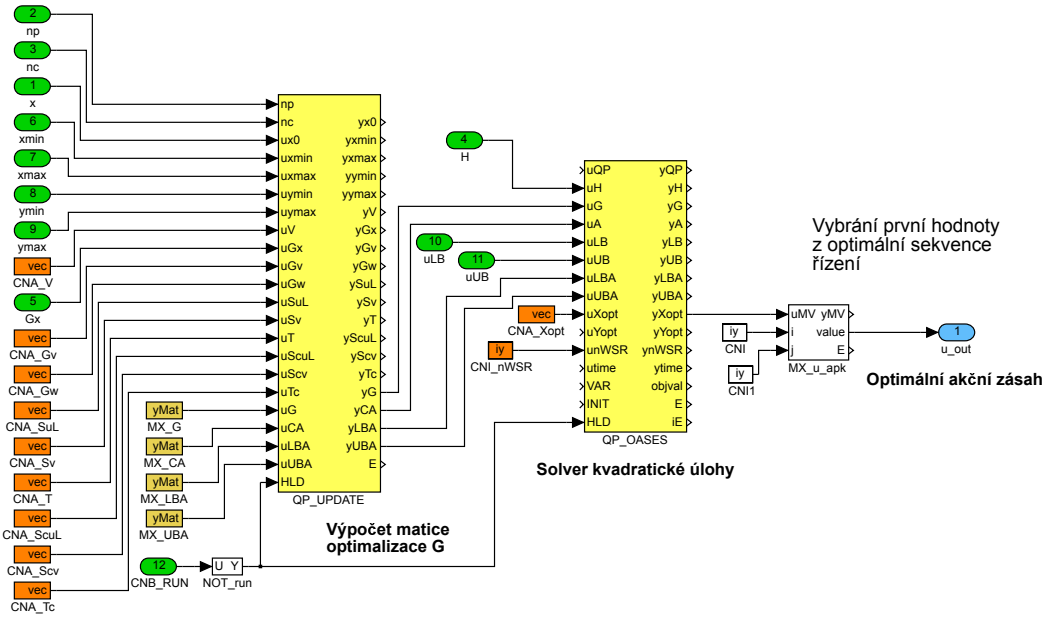
Celé schéma by se dalo rozdělit do třech hlavních segmentů, kterými jsou vstupní parametry regulace (oranžové bloky), MPC regulátor a rekonstruktor (tyrkysový blok) a model řízeného systému (zelený blok). Pomocí růžového bloku vlevo dole, *CNB_RUN*, se celá simulace spustí. Nad tímto blokem je ještě červený *CNB_RESET*. Ten byl vytvořen pro použití při regulaci reálné soustavy. Jeho spuštěním dojde k nastavení generovaného akčního zásahu a stavů řízeného systému na hodnotu 0. Tento proces je realizován ještě pomocí šedého bloku *RESET_logic*, který je na lince od *CNB_RESET*. Blok *CNB_RESET* pouze vhodně upravuje vstupní signál, aby se dostalo požadovaného efektu restartování regulační smyčky. Vstupní hodnoty pro regulaci (oranžově) jsou postupně od shora dolů následující. Jako první jsou 2 bloky pro zadání referenčních hodnot motoru - poloha, rychlost. Pomocí dalších dvou bloků se zadává hodnota horizontu predikcí a řízení. Další 2 bloky určují matice pro optimalizaci $\mathbf{H}$ a $\mathbf{G}_x$. Jejich tvary jsou uvedeny v kapitole o algoritmu prediktivního řízení. Následující 2 bloky udávají meze pro stavy řízené soustavy a další 2 bloky zase pro výstupy systému. Pomocí posledních tří bloků lze zadávat omezení na optimalizační proměnou, což je v uvažovaném případě vektor řízení $\mathbf{U}_{nc}$.

Dále je ve schématu nejdůležitější blok celého zapojení MPC regulátor + rekonstruktor (tyrkysově). Právě ten provádí samotnou prediktivní regulaci. První část jeho vstupů je popsána v předchozím odstavci. Dalšími vstupy do regulátoru jsou výstupy řízeného systému a poslední vygenerovaný akční zásah. Výstupem tohoto bloku je optimální akční zásah. Blok je realizován jako subsystém a jeho vnitřní zapojení znázorňuje následující obrázek.



Obrázek 21: Schéma zapojení MPC s on-line solverem pro simulaci v REXYGENu - MPC

Vstupní hodnoty subsystému zde mají světle zelenou barvu a jejich význam byl uveden v předchozím odstavci. Celé schéma tohoto subsystému by se dalo rozdělit do 4 částí. První z nich jsou šedé bločky vlevo dole sloužící k zadání referencí. Další částí subsystému je rekonstruktor (tmavě zelený). Ten na základě vstupů a výstupů řízeného systému odhaduje hodnoty všech jeho stavů. Současně prostřednictvím něho dochází i k odhadu stavu neměřitelné vstupní poruchy. Růžový blok reprezentuje multiplexor. Na jeho výstupu je vektor stavu (resp. jeho rozšířená verze viz kapitola 2.7), který se použije jako vstup do následujícího žlutého bloku. Ten realizuje již samotný výpočet kvadratického programu. Jeho vnitřní zapojení zachycuje následující schéma.



Obrázek 22: Schéma zapojení MPC s on-line solverem pro simulaci v Matlabu - solver QP

V levé části se nacházejí vstupní parametry optimalizace. Mimo vektoru stavu to jsou hodnoty horizontů, omezení na stavy a výstupy, matice pro optimalizaci a také matice predikce $\widetilde{\mathbf{T}}$, $\widetilde{\mathbf{S}}_{uL}$, $\widetilde{\mathbf{T}}_c$ a $\widetilde{\mathbf{S}}_{cuL}$. Kromě nich se tam ještě nachází matice s indexem v a w. Tyto matice slouží k definici poruch. Nicméně ty v uvažovaném případě nebyly použity, a proto není třeba se jimi dále zabývat. V zapojení následuje žlutý blok *QP_UPDATE*. Ten realizuje výpočet matice predikce $\mathbf{G}$ a to jako $\mathbf{G}_x \cdot \widetilde{\mathbf{x}}_k$, kde $\widetilde{\mathbf{x}}_k$ odpovídá aktuálnímu stavu řízené

soustavy společně s referencemi a posledním aplikovaným akčním zásahem z růžového bloku z předchozího obrázku. Dalším vstupem bloku *QP_UPDATE* jsou tmavě žluté bloky *Data storage*. Pomocí nich se inicializují matice, přes které se do optimalizace aplikuje omezení na výstupy. Hodnoty těchto matic blok *QP_UPDATE* rovněž počítá. V zapojení následuje blok *QP_OASES*, což je již solver, který řeší samotnou úlohu optimalizace. Jeho vstupem jsou tedy matice optimalizace **H** a **G** a dále matice definující omezení na optimalizační proměnnou a na výstupy řízené soustavy. Výstupem bloku *QP_OASES* je již vektor optimálních akčních zásahů o délce n_c. V dalším postupu je tedy nutné vybrat pouze první hodnotu z optimální sekvence zásahů a tu označit za výstup subsystému a tedy i celého regulátoru.

Nyní již zpět k celkovému zapojení regulační smyčky. Po MPC regulátoru se ve schématu nachází model řízeného systému (světle zelený). Jeho vstupem je jednak nalezený optimální akční zásah z regulátoru a také hodnota vstupní poruchy. Ta je realizována prostřednictvím bloku *CNR_porucha*. Bloky *SSW* a *DEL_du* slouží pouze k časovému zpoždění začátku jejího působení. Výstupy modelu jsou posléze přivedeny do bloku *TRND_vystupy*, kde lze pozorovat jejich průběh. Výstupy jsou rovněž přivedeny jako zpětná vazba do bloku regulátoru.

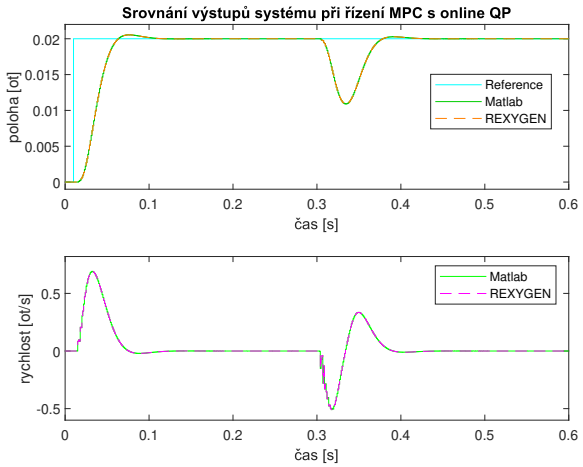
6.4 Simulace prediktivního řízení

V této podkapitole proběhnou simulace regulace v Matlabu a REXYGENu pomocí MPC regulátoru ve standardním tvaru s on-line řešením kvadratické úlohy. K simulaci se použijí tyto dva nástroje vzhledem k tomu, že v Matlabu probíhal samotný návrh MPC a že pomocí REXYGENu bude později řízen reálný systém. Při regulaci reálného stroje bude tedy možné v REXYGENu pouze zaměnit vstupy a výstupy modelu za vstupy a výstupy skutečné soustavy. V tuto chvíli bude snaha řídit model systému popsaný v předchozí kapitole. Samotný experiment bude probíhat tak, že na začátku simulace dojde ke skokové změně referenční hodnoty polohy z 0 na 0.02 a po skončení přechodového děje bude do systému aplikovaná neměřitelná vstupní porucha o velikosti -0.06.

Nejdříve zde budou vykresleny průběhy důležitých veličin regulační smyčky při popsané simulaci. Konkrétně se bude jednat o výstup systému a akční zásahy generované regulátorem. V další části této podkapitoly proběhne ještě srovnání časové náročnosti regulátoru při simulacích z Matlabu, kdy s k řešení kvadratické úlohy použijí postupně solvery *quadprog*, *qp-OASES* a *qp-OASES* v *hot-start* variantě.

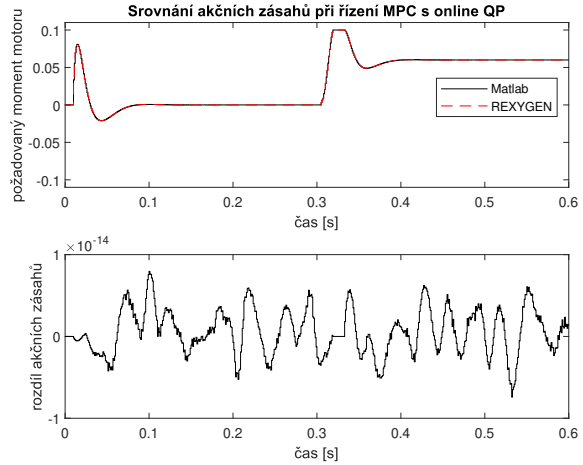
6.4.1 Časové průběhy důležitých veličin při simulaci

Jako první zde budou vykresleny průběhy poloh a rychlostí motoru při skokové změně referenční hodnoty polohy z 0 na 0.02. Po ustálení výstupů zasáhne do regulační smyčky vstupní porucha o velikosti -0.06. Výsledek znázorňuje následující obrázek.



Obrázek 23: Simulace řízení MPC regulátorem s on-line solverem - výstupy

Na první pohled je zřejmé, že pro simulaci z Matlabu i REXYGENu byl dosažený stejný výsledek. Co se týče samotného průběhu simulace, tak hned na začátku je dobře patrný přechodový děj, kdy se poloha motoru dostala na novou referenční hodnotu. V čase 0.3 sekundy byla do regulační smyčky zavedena vstupní porucha. Jak je vidět, tak její vliv se podařilo regulátoru poměrně rychle a plynule odstranit. Další obrázek zachycuje průběh akčních zásahů generovaných MPC regulátorem při simulaci.



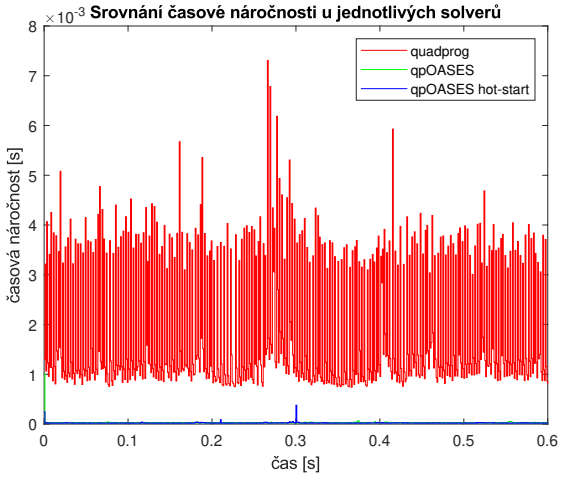
Obrázek 24: Simulace řízení MPC regulátorem s on-line solverem - řízení

I zde je patrná plynulost změn hodnot akčních zásahů během celé simulace. Po zásahu poruchy se akční zásahy dokonce dostaly na svou saturační mez. Je tedy dobře vidět, že i s touto situací si regulátor poradil dobře. Ve spodní části obrázku je ještě vykreslen rozdíl mezi generovanými akčními zásahy v Matlabu a REXYGENu. Hodnoty rozdílu jsou téměř zanedbatelné, což je dobrý předpoklad pro to, aby se mohlo v REXYGENu odzkoušet řízení reálného systému.

6.4.2 Časová náročnost solverů

Tato podkapitola bude sloužit ke srovnání rychlosti výpočtu kvadratické úlohy u jednotlivých solverů, které lze použít k řešení optimalizační úlohy u MPC ve standardním tvaru. Konkrétně se použije solver *quadprog*, *qp-OASES* a *qp-OASES* v *hot-start* variantě. Experimenty v této podkapitole budou prováděny na standardním stolním počítači s Windows 11 Home verze 22H2 a procesorem *AMD Ryzen 5 5600H* s taktovací frekvencí 3.3 GHz. Za poznámku stojí, že na cílovém HW při řízení reálného systému by nejspíše bylo dosaženo o něco kratších časů výpočtu u všech solverů. Tato podkapitola tedy bude sloužit pouze ke vzájemnému porovnání výpočetních časů uvedených solverů. Zjistí se tak alespoň škálově, jak jsou jednotlivé solvery vůči sobě rychlé.

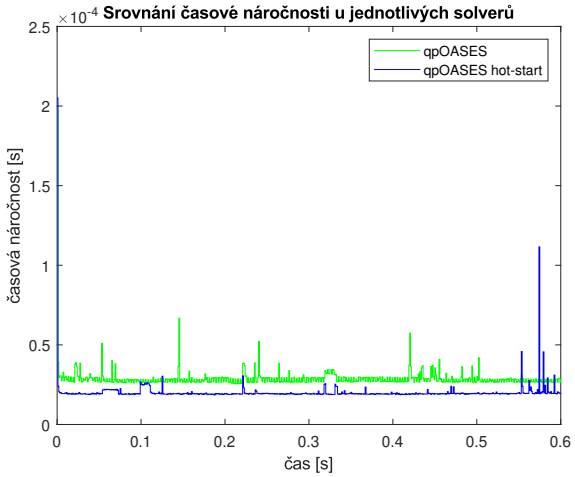
K testování výpočetní náročnosti solverů dojde při simulaci regulační smyčky s MPC regulátorem ve standardní verzi, kdy průběh experimentu bude totožný jako v předchozí podkapitole. Dojde tedy ke skokové změně referenční hodnoty polohy z 0 na 0.02 a poté do regulační smyčky zasáhne porucha o velikosti -0.06. Po provedení experimentu vyšel tento výsledek.



Obrázek 25: Srovnání časové náročnosti solverů při prediktivním řízení v Matlabu

Jak se dalo předpokládat, tak nejvíce času pro výpočet kvadratického programu potřeboval solver *quadprog*. Jeho průměrná doba výpočtu kvadratického programu je dokonce vyšší než perioda vzorkování řízené soustavy 1 *ms*. To by znamenalo, že tento solver by nešlo v praxi pro řízení mechatronického ramene pomocí uvedeného HW použít. Nicméně lze předpokládat, že na cílovém HW pro řízení reálného systému by byly naměřené časy u všech solverů menší. Je to díky jinému operačnímu systému a také kvůli odlišnému prostředí reálného času. Poměr rozdílu výpočetních náročností by však byl ale pravděpodobně stále zachován.

Aby byl vidět lépe rozdíl mezi časovými náročnostmi u solverů qpOASES, tak zde bude ještě uveden detail předchozího obrázku.



Obrázek 26: Srovnání časové náročnosti solverů při prediktivním řízení v Matlabu- detail

Jak je vidět, tak z hlediska časové náročnosti je na tom nejlépe skutečně solver *qpOASES* v *hot-start* variantě. Avšak vzhledem k velikosti úlohy, kterou bylo nezbytné pro uvažovaný příklad řešit, nedošlo k tak razantní časové úspoře oproti solveru *qpOASES* v jeho standardní formě. Pro tento případ totiž byl horizont predikcí nastaven na hodnotu $n_c = 3$, což vede na malou velikost kvadratického programu. Ukazuje se totiž, že čím vyšší je hodnota horizontu řízení, tím vyšší je rozdíl v časové náročnosti u solveru *qpOASES* v *hot-start* verzi oproti jeho standardní formě.

6.5 Prediktivní řízení reálného systému z REXYGENu

Předchozí kapitola byla věnována otestování navrženého MPC regulátoru při regulaci modelu systému. Ukázalo se, že regulátor si poměrně dobře dokáže poradit se skokovou změnou referenční hodnoty a i se vstupní poruchou, jejíž vliv po chvíli vyrušil. V této podkapitole proběhne otestování navrženého MPC ve standardní formě s on-line řešením kvadratického programu při regulaci skutečného systému, který je popsán v kapitole 5.

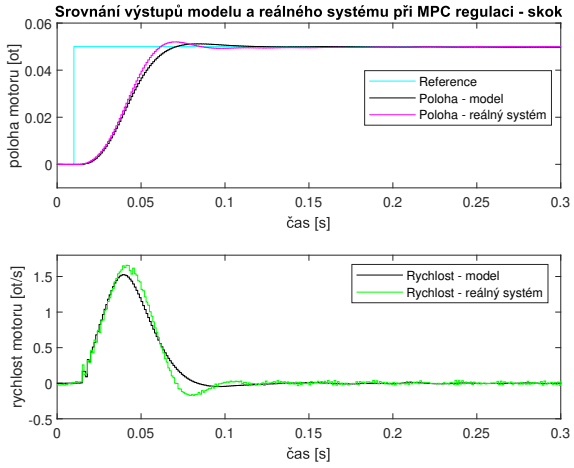
Algoritmus MPC bude implementován na průmyslovém počítači B+R Automation PC 910 (dokumentaci lze nalézt na https://www.br-automation.com/cs/produkty/prumyslove-pocitace/automation-pc-910/) ve variantě s quad-core i5 procesorem a taktovací frekvencí 2.7 GHz. Co se týče softwarové části, tak bude použit operační systém Debian Linux s real-time patchem. Aplikační prostředí reálného času bude REXYGEN. Vzhledem k tomu lze k regulaci skutečného systému využít pouze s několika malými úpravami schéma zapojení ze simulací z REXYGENu.

Vytvořený prediktivní regulátor bude otestován na situace, kdy dojde ke skokové změně referenční hodnoty a kdy se aplikuje do systému vstupní porucha. Do grafů v této kapitole budou současně vykresleny i průběhy veličin získaných během simulací, aby bylo dobře patrné, do jaké míry se chování liší.

6.5.1 Přechodový děj

Jako první proběhne srovnání výstupů systému získaných během simulací a během experimentů na reálné soustavě při skokové změně referenční polohy a to z hodnoty 0 na 0.05.

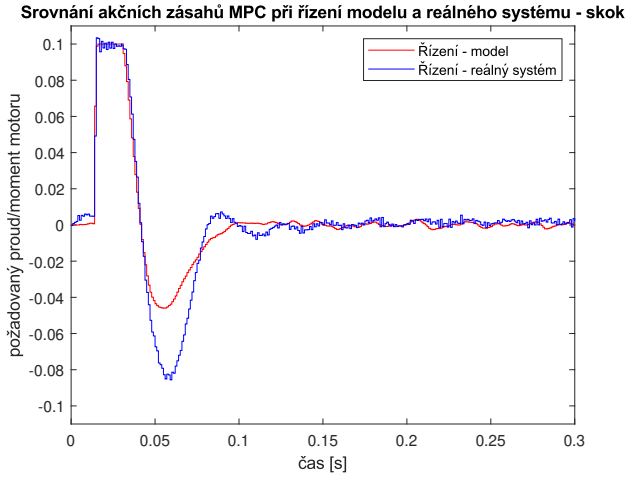
Výsledek zachycuje následující obrázek.



Obrázek 27: Srovnání výstupů modelu a skut. systému při standardní MPC regulaci - skok

Je zřejmé, že sledované průběhy si přibližně odpovídají. Doba regulace je přibližně stejná a to 0.12 vteřiny. Dále oba přechodové děje jsou plynulé a vyskytuje se v nich překmit o téměř shodné velikosti. Nicméně u reálného systému se v průběhu polohy vyskytuje při ustalování na nové referenční hodnotě ještě jeden podkmit, což u modelu není. Co se týče průběhů rychlostí během přechodového děje, tak nejdříve došlo k jejímu navýšení ve snaze dostat se na novou referenční hodnotu. Nicméně u reálného systému byla naměřena větší maximální hodnota. Poté se rychlost v obou případech opět snižovala k 0. Při řízení modelu systému byl ale přechod k nule o něco plynulejší. Dále lze pozorovat v průběhu rychlosti motoru u reálného systému přítomnost šumu s větší amplitudou.

Jako poslední proběhne srovnání akčních zásahů generovaných regulátorem při řízení modelu a skutečného systému.



Obrázek 28: Srovnání generovaných akčních zásahů u modelu a skut. systému - skok

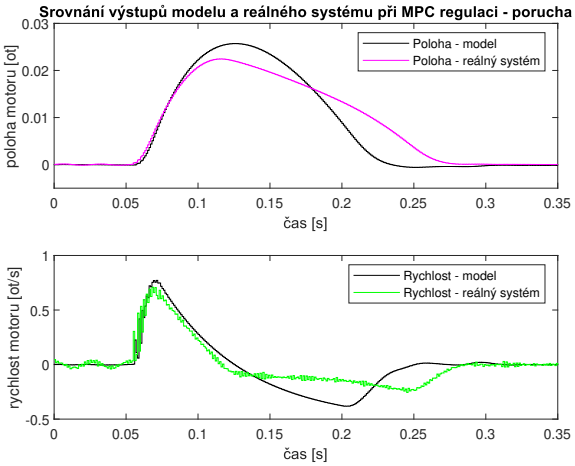
Je zřejmé, že v obou případech po změně referenční hodnoty začal regulátor generovat akční zásahy na mezi saturace. Následně se hodnoty akčních zásahů začaly snižovat a s jedním

podkmitem se ustálily okolo hodnoty 0, což ukazuje, že se systém dostal do nové referenční polohy. Nicméně lze pozorovat, že podkmit u reálného systému opět nabýval větších hodnot.

Shrnutím dosažených výsledků lze říci, že průběhy sledovaných veličin si vcelku odpovídají. V průbězích byly jasně patrné stejné trendy u všech veličin. Nicméně byly vidět i jisté rozdíly. U reálného systému nabývaly průběhy větších hodnot a celkově byly i více kmitavější. Lze tedy předpokládat, že skutečný systém oproti nalezenému modelu více kmitá. Další nesrovnalost bylo možné pozorovat ve velikosti amplitudy šumu, jehož přítomnost byla při řízení reálné soustavy více patrná. To by se dalo vysvětlit neideálními senzory, s čímž je ale nutné při řízení reálných systémů vždy počítat.

6.5.2 Odezva na poruchu

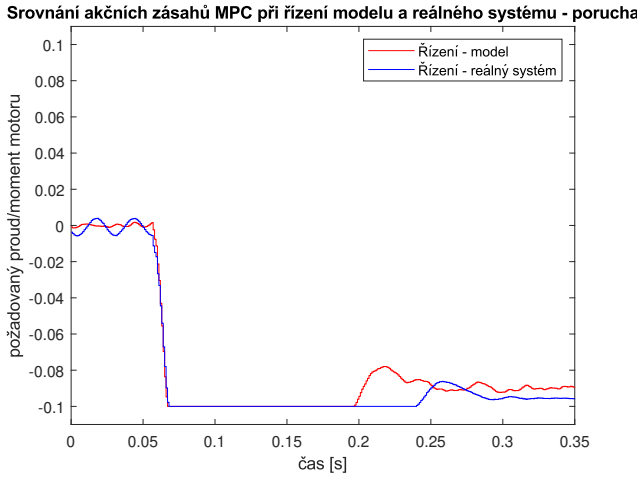
Dále proběhne srovnání výstupů získaných během simulací a během experimentů na reálné soustavě při aplikaci skokové vstupní poruchy do uzavřené smyčky. Velikost této poruchy bude činit 0.09. Nejdříve zde bude uveden obrázek, který zachycuje srovnání průběhů výstupů systému.



Obrázek 29: Srovnání výstupů modelu a skut. systému při standardním MPC - porucha

Je vidět, že zde již nebylo dosaženo takové shody jako u odezvy na skok na vstupu systémů. Nicméně v průbězích lze pozorovat stále stejný trend. Aplikace poruchy do regulační smyčky vyvolala plynulé odchýlení polohy motoru od požadované veličiny. Poté svými akčními zásahy regulátor opět dostal polohu motoru na referenční nulovou hodnotu. Co se týče rychlostí, tak bylo dosaženo obdobného výsledku jako u průběhu poloh. Nejdříve se motor vlivem poruchy začal otáčet na jednu stranu a po zapůsobení regulátoru se začal motor opět otáčet na stranu druhou, čímž se přesunul zpět na požadovanou pozici.

Jako poslední zde budou ukázány průběhy akčních veličin generovaných regulátorem při řízení modelu a skutečného systému.



Obrázek 30: Srovnání generovaných akčních zásahů u modelu a skut. systému - porucha

Rovněž je zde zachován stejný tvar průběhu akční veličiny při řízení modelu a reálného systému. Po aplikaci poruchy začal regulátor v obou případech generovat akční zásahy na mezi saturace a to poměrně na dlouhou dobu. To bylo způsobeno malým rozdílem mezi hodnotou saturace a velikostí poruchy. Výstup regulátoru se pak ustálil na hodnotě -0.09, což odpovídá velikosti aplikované poruchy. Díky tomu se výstupní poloha drží okolo nuly, i když porucha stále do systému působí.

Z vykreslených obrázků bylo vidět, že tvary průběhů sledovaných veličin si přibližně odpovídaly. Nicméně byl zřetelné i jisté rozdíly mezi modelem a skutečností. Po úvaze se došlo k tomu, že tento jev mohl být způsobený přítomností nelineárního tření u skutečného systému, které nebylo při modelování bráno v potaz. K rozdílu mezi simulacemi a reálnými experimenty jistě přispěl i šum, který se u reálné soustavy vyskytuje například kvůli neideálním senzorům.

7 Řízení reálného systému explicitním MPC

Hlavním cílem této kapitoly bude navrhnout a poté odzkoušet explicitní prediktivní regulátor při řízení skutečného systému, kterým je mechatronické rameno. Pro návrh explicitního regulátoru se použije algoritmus popsaný v kapitole 4. Konkrétně se pro nalezení všech regionů v prostoru stavů řízeného systému použije metoda změny aktivní množiny na základě původu omezení, u které bylo dosaženo nejlepších výsledků. Dále pro určení regionu, kterému přísluší aktuální stav systému, se použijí obě představené metody tedy sekvenční prohledávání stavového prostoru i algoritmus využívající binární vyhledávací strom.

Tato kapitola bude obsahově organizována následovně. Nejdříve zde proběhne návrh explicitního MPC regulátoru. Poté budou představena schémata zapojení z Matlabu a REXYGENu, která se použijí k simulacím regulační smyčky s explicitním MPC. Následovat budou výsledky, které se získaly během simulací, a jejich vyhodnocení. Poslední podkapitola se bude zabývat explicitním prediktivním řízením reálného mechatronického systému.

7.1 Návrh explicitního prediktivního regulátoru

Při návrhu explicitního prediktivního regulátoru se využilo stejných volitelných parametrů jako při návrhu MPC ve standardním tvaru s on-line solverem. Díky tomu by mělo být zaručeno, že se regulátory budou chovat naprosto totožně. U explicitní verze by ovšem mělo být dosaženo lepší časové náročnosti.

Konkrétně se zvolily tyto parametry. Hodnota horizontu predikcí je $n_p = 30$ a horizontu řízení $n_c = 3$. Akční veličina bude saturována do rozmezí ± 0.1. Pro explicitní prediktivní řízení to tedy vede na matice definující omezení v tomto tvaru

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, \mathbf{S} = \begin{bmatrix} \mathbf{0} \end{bmatrix}. \tag{97}$$

Váhové matice $\mathbf{Q}$ a $\mathbf{R}$ mají tvar

$$\mathbf{Q} = \begin{bmatrix} 2000 & 0 \\ 0 & 0.1 \end{bmatrix}, \mathbf{R} = 500. \tag{98}$$

Rovněž se využilo algoritmu pro zaručení nulové odchylky v ustáleném stavu. Matice optimalizace tedy vypadají takto

$$\mathbf{H} = \begin{bmatrix} 1011.2575 & -496.1707 & 65.3721 \\ -496.1707 & 1010.5947 & -439.7418 \\ 65.3721 & -439.7418 & 1219.1977 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} -0.4531 & 0.5326 & 1.4416 \\ 2.8304 & 3.1149 & 27.7447 \\ 8.6333 & 15.8725 & 129.0831 \\ 0.5313 & 1.5755 & 14.4835 \\ 0.0684 & 0.0571 & 1.0847 \\ 0.1554 & 0.1064 & 1.0876 \\ -0.0004 & 0.0096 & 0.0890 \\ -0.0004 & 0.0013 & 0.0118 \\ -481.8864 & -446.4055 & -3494.3272 \\ -1.8317 & -1.774 & -22.3203 \\ -500 & 0 & 0 \\ 80.4588 & 74.6823 & 844.828 \end{bmatrix}. \quad (99)$$

Po aplikaci metody změny aktivní množiny na základě původů omezení pro nalezení všech regionů v prostoru stavů řízeného systému se ukázalo, že pro uvažovaný případ se v prostoru stavů nachází 27 regionů s 11-ti unikátními zákony řízení.

Dalším krokem bylo implementovat algoritmus pro hledání konkrétního regionu pro jeden daný stav systému. Implementace metody sekvenčního vyhledávání byla jednoduchá. Přistoupilo se tedy k vytvoření metody generující binární vyhledávací stromu. Po jejím spuštění byl nalezen strom s počtem 223 uzlů. Hloubka tohoto stromu je přitom 9. Z toho vyplývá, že pro určení regionu příslušícímu aktuálnímu stavu systému stačí v nejhorším případě vyhodnotit pouze 9 lineárních nerovností.

7.2 Schéma zapojení z Matlabu

V obou případech, jak pro explicitní MPC se sekvenčním prohledávání stavového prostoru tak i pro explicitní MPC s binárním stromem pro určení regionu, se používalo schéma z Matlabu ve stejné podobě jako je na obrázku 19. Rozdíl je v tom, jak byl implementován kód v bloku *Triggered subsystem*. V případě sekvenčního prohledávání se postupně procházely regiony, dokud se nenašel ten, do kterého aktuální stav patří. U metody využívající binární vyhledávací strom byl v bloku *Triggered subsystem* implementován kód na procházení tohoto binárního stromu, jak to bylo popsáno v kapitole vysvětlující explicitní prediktivní řízení.

7.3 Schéma zapojení z REXYGENu

V této podkapitole budou představena 3 zapojení z REXYGENu, která se použila k simulaci regulace s explicitním MPC. Všechna tři používají k určení regionu příslušícího aktuálnímu stavu systému binární vyhledávací strom.

V prvním z nich se k implementaci algoritmu pro prohledávání binárního stromu využil blok REXLANG. To je standardní programovatelný blok REXYGENu, pomocí kterého lze potřebný kód dobře vytvořit. Používá se při tom programovací jazyk podobný C se svou vlastní knihovnou funkcí, které lze v kódu používat. Některé tyto funkce jsou nativně napsané v C++ a běží zkompilované. Ve výsledku by to mělo zaručit rychlejší běh těchto funkcí. Rovněž se odzkouší i blok PYTHON, který lze použít ke stejnému účelu. Pomocí toho je možné rovněž algoritmus na prohledávání stromu implementovat, ale zde pomocí programovacího jazyka Python. V tomto prostředí lze využít knihovny *numpy*. Ta nabízí matematické funkce již kompilované do jazyka C, což by mělo vést na jejich rychlé vyhodnocení.

Pro oba popsané bloky, REXLANG i PYTHON, v prostředí REXYGEN platí, že jsou

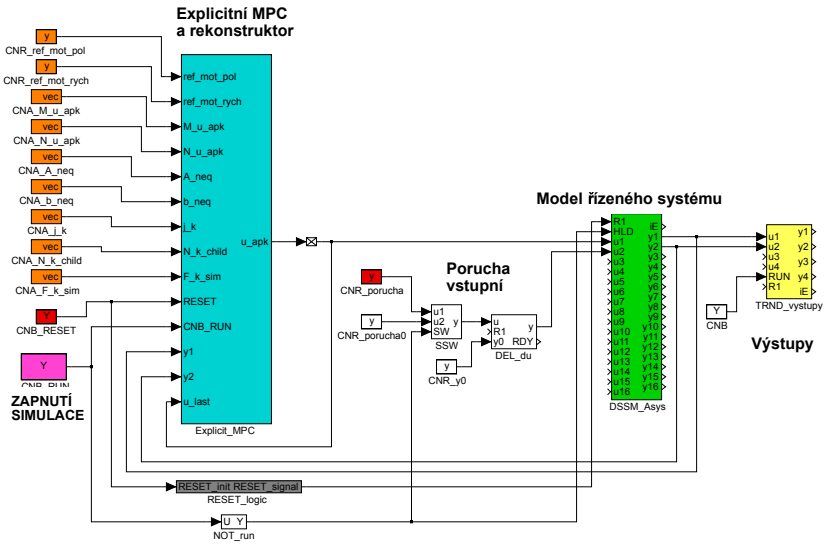
interpretované, nikoliv kompilované. Z toho důvodu je snaha používat knihovny funkcí, které tyto bloky nabízejí. Tyto funkce jsou totiž již kompilované a měly by tedy být vyhodnocovány řádově rychleji.

Z uvedených důvodů se přistoupilo k vytvoření samotného bloku do programu REXYGEN s názvem MPCE, který provádí prohledávání binárního stromu a vypočítává optimální akční zásah. Blok byl navržen právě pro účely explicitního prediktivního řízení. Třetí použité schéma zapojení bude tedy s tímto blokem. V tomto případě se již jedná o blok kompilovaný nikoliv interpretovaný a z hlediska výpočetních nároků by zde mělo být dosaženo lepších výsledků než při použití bloků REXLANG a PYTHON.

Tato podkapitola bude rozdělena do třech části, kde každá z nich se bude zabývat jedním uvedeným způsobem implementace.

7.3.1 Schéma explicitního MPC při použití bloku REXLANG

Při použití bloku REXLANG pro implementaci algoritmu na prohledávání binárního stromu vypadá zapojení v REXYGENu takto.

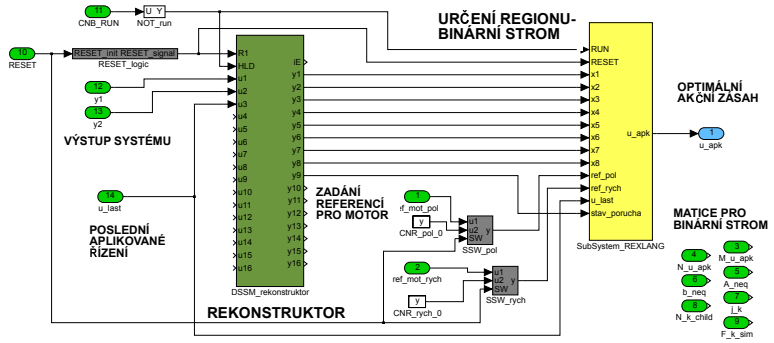


Obrázek 31: Zapojení explicitního MPC v REXYGENu s binárním stromem v REXLANG

Jak je vidět, tak toto schéma se od zapojení MPC regulátoru s on-line solverem v REXYGENu zásadně neliší. Odlišná je pouze implementace samotného regulátoru (tyrkysově) a jeho vstupní hodnoty (oranžově). Následující odstavce se tedy zaměří pouze na tuto část zapojení. Zbytek schématu již byl popsán při vysvětlení zapojení standardního MPC s on-line solverem.

Vstupní parametry (oranžově) explicitního prediktivního regulátoru jsou od shora tyto. Pomocí prvních dvou bloků se zadává referenční hodnota pro polohu a rychlost motoru. Zbylé oranžové bloky určují vstupní parametry přímo pro binární vyhledávací strom. Postupně se jedná o bloky určující matice $\mathbf{M}$ a $\mathbf{N}$ pro výpočet optimálního akčního zásahu. Matice $\mathbf{A}_{neq}$ a b_{neq} definují hranice všech regionů. Blok *CNA_j_k* odkazuje na dělící přímku j_k pro každý uzel stromu. Dále *CNA_N_k_child* určuje dva následovníky pro každý uzel, který není listem. Poslední oranžový blok s označením *CNA_F_k_sim* definuje optimální zákon řízení pro každý listový uzel stromu.

Vnitřní zapojení bloku *Explicit_MPC* obsahující explicitní prediktivní regulátor a rekonstruktor vypadá takto.

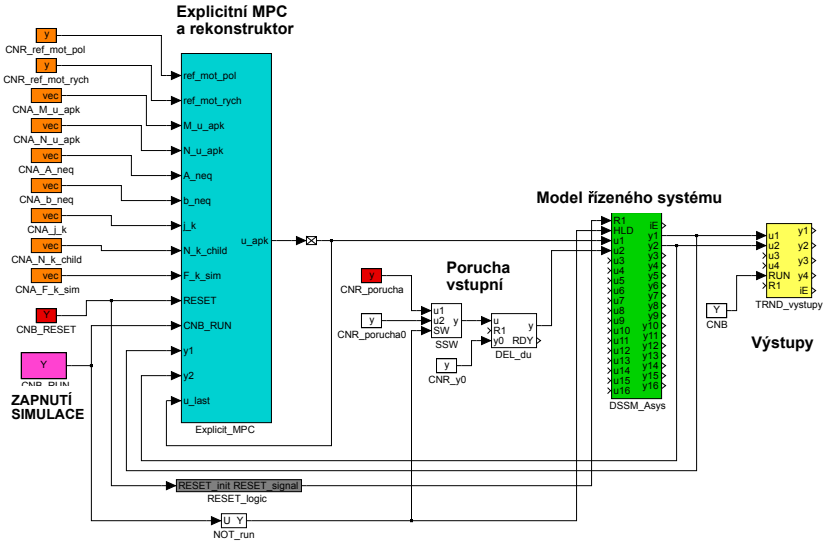


Obrázek 32: Zapojení expl. MPC v REXYGENu s binárním stromem v REXLANG - MPC

Světle zelené bloky představují vstupy subsystému, jejichž význam byl již vysvětlen dříve. Tmavě zelený blok je rekonstruktor. Ten na základě vstupů a výstupů řízeného systému odhaduje hodnoty všech jeho stavů. Současně prostřednictvím něj dochází k odhadu hodnoty neměřitelné poruchy. Odhadnutý vektor stavu systému společně s hodnotou poruchy, poslední hodnotou aplikovaného řízení a referencemi tvoří vstup pro žlutý blok, což je vlastně samotný explicitní prediktivní regulátor. Právě v něm je implementován algoritmus pro prohledávání binárního stromu. Za povšimnutí ještě stojí, že v pravém dolním rohu jsou nezapojené vstupní hodnoty pro binární strom. Je to z toho důvodu, že se v algoritmu binárního vyhledávacího stromu používají přes reference. Výstupem regulátoru je již výsledná hodnota akčního zásahu, která se použije jako vstup do řízené soustavy.

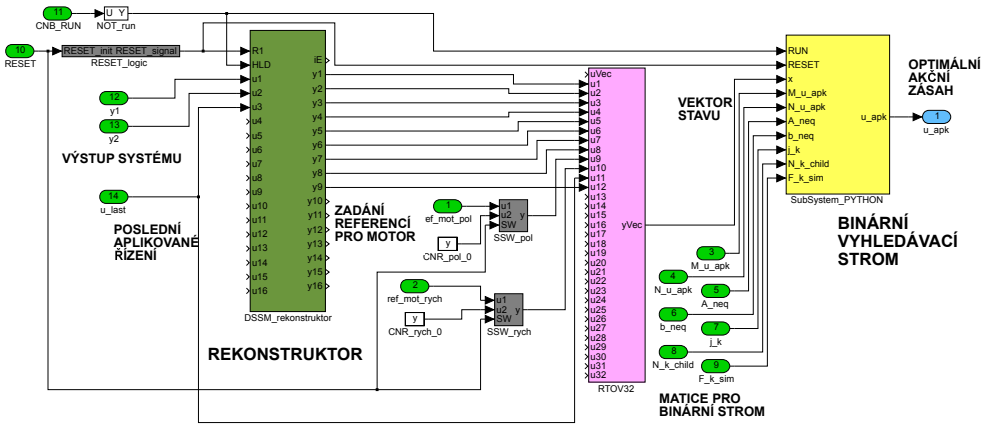
7.3.2 Schéma explicitního MPC při použití bloku PYTHON

Zapojení z REXYGENu, kde se k určení regionu a vypočtení optimální akčního zásahu použil blok PYTHON, vypadá následujícím způsobem.



Obrázek 33: Zapojení explicitního MPC v REXYGENu s binárním stromem v PYTHON

Je zřejmé, že v tomto případě je zapojení naprosto totožné se zapojením explicitního prediktivního regulátoru, kde dochází k prohledávání binárního stromu přes blok REXLANG. Liší se ovšem vnitřní zapojení bloku *Explicit_MPC*, ve kterém je implementovaný rekonstruktor a samotný explicitní regulátor. Explicitní regulátor je zde totiž realizován pomocí bloku PYTHON. Vnitřní zapojení tyrkysového bloku *Explicit_MPC* je tak následující.

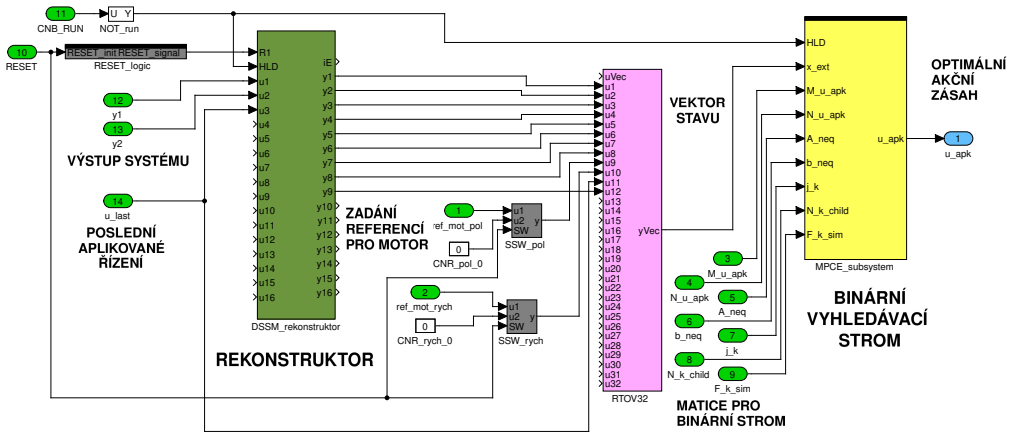


Obrázek 34: Zapojení expl. MPC v REXYGENu s binárním stromem v PYTHON - MPC

Světle zelené bloky jsou vstupy subsystému, jejichž význam byl vysvětlen již dříve. Dále se ve schématu nachází rekonstruktor, který na základě vstupů a výstupů řízeného systému poskytuje informaci o jeho stavech a rovněž poskytuje informaci o hodnotě neměřitelné poruchy. Pomocí růžového bloku se definuje rozšířený vektor stavu (tj. stav systému, referenční hodnoty, poslední aplikované řízení a odhadnutý stav neměřitelné poruchy), pro který je nutné nalézt příslušný region v prostoru stavů. To je realizováno přes žlutý blok, který reprezentuje samotný explicitní prediktivní regulátor. Ten prostřednictvím prohledávání binárního stromu nalezne odpovídající region pro stav systému a následně vypočte optimální akční zásah pomocí přidruženého zákonu řízení. Současně je vidět, že vstupem žlutého bloku jsou i matice určující parametry binárního vyhledávacího stromu. Výstupem explicitního regulátoru je již optimální akční zásah, který se použije jako vstup do řízeného systému.

7.3.3 Schéma explicitního MPC při použití bloku MPCE

Schéma zapojení z REXYGENu, kde se k určení regionu a vypočtení optimálního zásahu použil blok MPCE, vypadá totožně jako na obrázku 33. Liší se ovšem vnitřní zapojení tyrkysového bloku s názvem *Explicit_MPC*, který zahrnuje samotný explicitní regulátor a rekonstruktor. Vnitřní zapojení subsystému je na následujícím obrázku.



Obrázek 35: Zapojení expl. MPC v REXYGENu s binárním stromem v MPCE - MPC

Je zřejmé, že ani toto zapojení se neliší zásadně od toho na obrázku 34. Rozdíl je zde pouze v bloku na prohledávání binárního stromu a výpočet akčního zásahu, což zde realizuje blok MPCE. Jeho vstupem je vektor stavu systému rozšířený o reference, poslední aplikované řízení a hodnotu neměřitelné poruchy. Dalšími vstupy je 7 matic, které definují binární vyhledávací strom pro danou úlohu. Výstupem je již hodnota optimálního akčního zásahu celého explicitního prediktivního regulátoru.

7.4 Simulace explicitního prediktivního řízení

Tato podkapitola se zaměří na otestování navržených explicitních prediktivních regulátorů při řízení modelu mechatronického ramene. Testování bude probíhat nejdříve v Matlabu (resp. Simulinku), kde probíhal i samotný návrh regulátorů. Poté se přistoupí k simulacím v REXYGENu, kde bude později realizováno i řízení reálného systému pomocí vytvořených explicitních MPC. Algoritmus MPC pro simulace v této podkapitole bude implementován na již zmíněném stolním počítači s operačním systémem Windows 11 a procesorem o taktovací frekvenci 3.3 GHz.

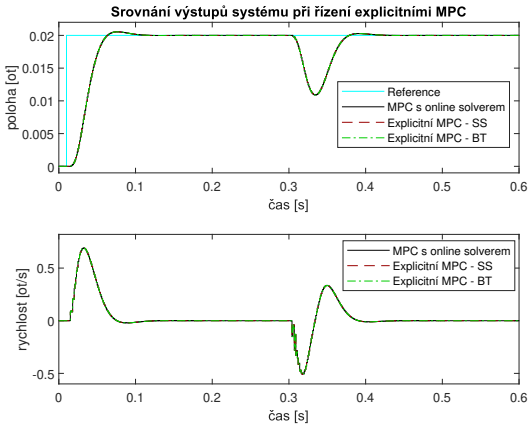
Nejdříve zde budou uvedeny výsledky simulací z Matlabu. V tomto prostředí proběhne otestování navrženého explicitního MPC používajícího sekvenční prohledávání stavového prostoru pro určení regionu příslušícímu aktuálnímu stavu a explicitního MPC využívajícího binární vyhledávací strom pro určení regionu. Další část této podkapitoly bude věnována simulacím z REXYGENu. Tam proběhne srovnání výsledků simulací, kde se k realizaci explicitního MPC použil blok REXLANG, PYTHON a MPCE.

Samotné simulace budou probíhat tak, že na začátku dojde ke skokové změně referenční hodnoty polohy z 0 na 0.02 a po skončení přechodového děje bude do systému aplikována neměřitelná vstupní porucha o velikosti -0.06.

7.4.1 Simulace z Matlabu

Pro ověření, že výstup explicitních MPC regulátorů při stejném experimentu je totožný s výstupem MPC ve standardním tvaru, zde budou v jednom grafu srovnány výsledky ze třech simulací. V první z nich se model systému bude řídit standardním MPC, ve druhé pomocí explicitního MPC se sekvenčním prohledáváním stavového prostoru (SS) a ve třetí bude model systému řídit explicitní MPC s binárním stromem pro určení regionu příslušícímu aktuálnímu stavu systému (BT).

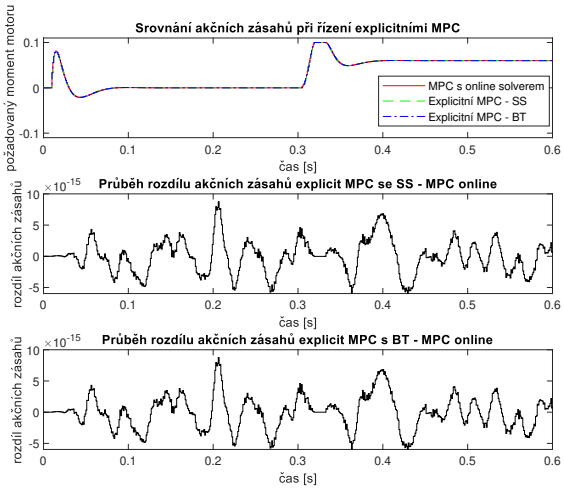
Po provedení simulací vyšly následující průběhy výstupů modelu řízeného systému.



Obrázek 36: Simulace řízení explicitními MPC regulátory v Matlabu - výstupy

Je zřejmé, že průběhy výstupů modelu během stejných simulací jsou při řízení explicitními MPC regulátory v obou případech zapojení naprosto stejné jako při regulaci se standardním MPC. Výstupy systémů tak ve všech třech případech plynule přešly na novou referenční hodnotu a dále si regulátory dobře dokázaly poradit i s vlivem neměřitelné poruchy.

Další obrázek zachycuje průběhy výstupů explicitních regulátorů během simulací se skokovou změnou referenční hodnoty a zásahem poruchy.



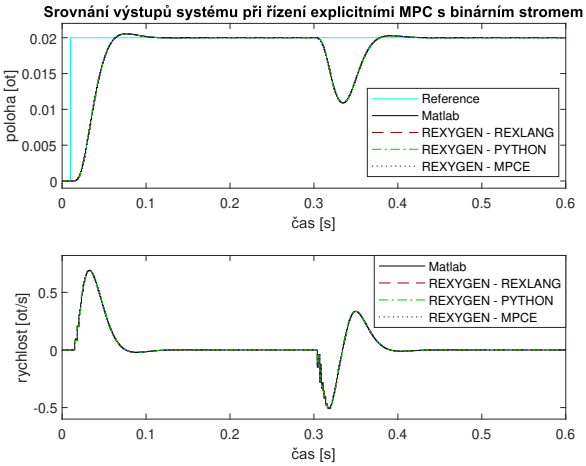
Obrázek 37: Simulace řízení explicitními MPC regulátory v Matlabu - řízení

Z průběhu rozdílu akčních zásahů je již zřejmé, že explicitní prediktivní regulátory se chovají naprosto stejně jako standardní MPC. Je vidět, že v obou případech stejně dobře zafungovalo i omezení na hodnoty akční veličiny. Vzhledem k těmto výsledkům lze přistoupit k simulacím v REXYGENu, kde později bude probíhat řízení reálného systému pomocí explicitních MPC.

7.4.2 Simulace z REXYGENu

V této části práce proběhne otestování navržených explicitních prediktivních regulátorů s binárním vyhledávacím stromem v REXYGENu. Uvažovány při tom budou 3 typy zapojení. V prvním z nich se k prohledávaní binárního stromu pro určení regionu příslušícímu stavu systému využil blok REXLANG, ve druhém blok PYTHON a ve třetím se ke stejnému účelu použil blok MPCE. Výsledky těchto simulací budou současně srovnány se simulacemi z Matlabu, kde se rovněž používal k určení regionu binární vyhledávací strom. V případě, že se budou simulace shodovat, tak to bude dobrý předpoklad pro odzkoušení explicitního MPC na řízení reálného systému.

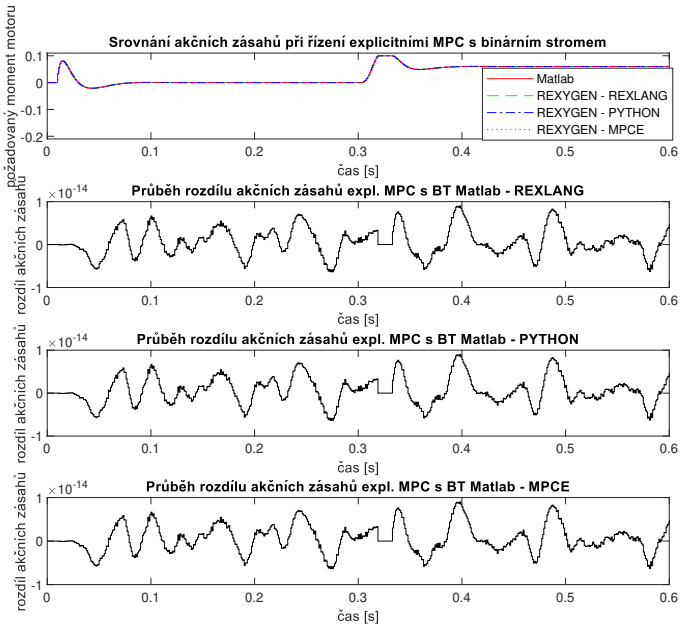
Po provedení simulací vyšly následující průběhy výstupu modelu řízeného systému.



Obrázek 38: Simulace řízení explicitními MPC regulátory v REXYGENu - výstupy

Z grafu je zřejmé, že všechny 4 explicitní regulátory používající binární vyhledávací strom dosáhly svými akčními zásahy stejného průběhu výstupů modelu systému. Při změně reference se tak výstup modelu dostal na novou požadovanou hodnotu a po zásahu poruchy se podařilo její vliv zanedlouho odstranit.

Na dalším obrázku jsou vidět akční zásahy generované explicitními MPC regulátory při stejném experimentu.



Obrázek 39: Simulace řízení explicitními MPC regulátory v REXYGENu - řízení

Z průběhů je i v tomto případě jasné, že regulátory generovaly akční zásahy se zanedbatelnými rozdíly. Vzhledem k těmto výsledkům lze přistoupit k experimentům s explicitními MPC regulátory při řízení skutečného systému.

7.5 Explicitní prediktivní řízení reálného systému

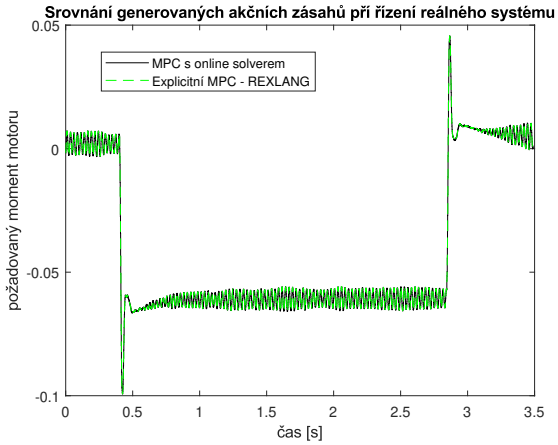
Experimenty z předchozí podkapitoly dokázaly, že během simulací se explicitní prediktivní regulátor chová totožně jako standardní verze MPC s on-line solverem. To je dobrý předpoklad k tomu, aby se mohl navržený explicitní MPC odzkoušet i k regulaci skutečného systému. Využije se přitom verze explicitního MPC, kde se k nalezení optimálního akčního zásahu používá blok REXLANG s algoritmem na prohledávaní binárního stromu.

Algoritmus explicitního prediktivního regulátoru bude pro řízení reálného systému implementován na průmyslovém počítači B+R Automation PC 910 ve variantě s quad-core i5 procesorem a taktovací frekvencí 2.7 GHz. Operační systém počítače je Debian Linux s real-time patchem a jako aplikační prostředí reálného času se využije REXYGEN.

Samotný experiment na reálném stroji bude probíhat tak, že v uzavřené smyčce s explicitním MPC dojde k aplikaci vstupní poruchy do systému o velikosti 0.06. Po ustálení výstupu systému porucha opět odezní. Výsledné akční zásahy explicitního MPC regulátoru budou následně srovnány s výstupy MPC regulátoru s on-line solverem, u kterého byl proveden stejný experiment. Porovnáním výstupů obou regulátorů se pak zjistí, zda explicitní MPC funguje stejně dobře jako MPC ve standardním tvaru.

7.5.1 Srovnání výstupu regulátorů při řízení reálného stroje

Výsledek popsaného experimentu zachycuje následující obrázek.

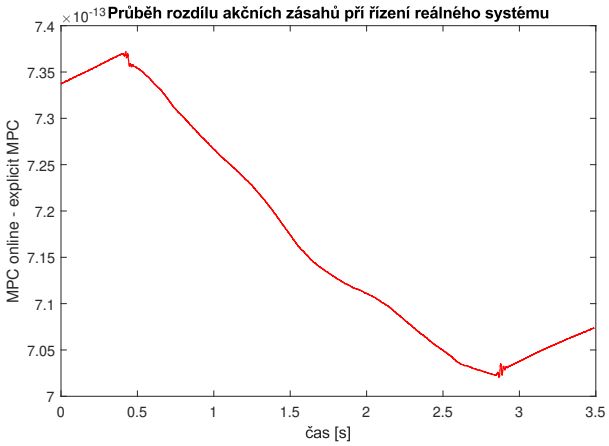


Obrázek 40: Srovnání akčních zásahů explicitního MPC při řízení reálného stroje

Na první pohled je dobře vidět, že výstupy obou regulátorů si odpovídají. Co se týče samotného průběhu akčních zásahů, tak je patrné, že v čase 0.45 sekund byla do systému aplikována porucha o velikosti 0.06, jejíž vliv se podařilo regulátorům odstranit. Následně v čase 2.8 vteřiny vliv poruchy odezněl a hodnoty akčních zásahů se opět vrátily k nule.

7.5.2 Srovnání rozdílu výstupu regulátorů při řízení reálného stroje

Pro úplnost by zde bylo ještě vhodné uvést rozdíl akčních zásahů obou regulátorů při provedeném experimentu. Výsledek je vidět na tomto obrázku.



Obrázek 41: Srovnání rozdílu akčních zásahů explicitního MPC při řízení reálného stroje

Z grafu je patrné, že velikost rozdílu se pohybovala řádově okolo 10^{-13}. Nicméně je zřejmé, že střední hodnota rozdílu není 0. Došlo se k tomu, že je to nejspíše způsobeno vlivem tolerancí, se kterými musí algoritmy pracovat. Obecně každý solver optimalizačních úloh má totiž nastavenou jistou toleranci od přesného řešení. Ve výsledku pak nemusí být nalezeno

naprosto přesné minimum účelové funkce a řešení stejného problému se pro více solverů může lišit. Jednotlivé přístupy k řešení optimalizačních úloh se mohou lišit i způsobem, kterým ve stavovém prostoru přistupují k řešení. Jinak řečeno z jakého směru ve stavovém prostoru přistupují k hledanému minimu. Díky tomu pak vlivem tolerancí může být opět nalezeno trochu jiné řešení stejného problému pro více přístupů k řešení optimalizačních úloh.

Tato situace nastala nejspíše i v uvažovaném případě, kdy jeden způsob hledání minima účelové funkce je přes solver `qp-OASES` a druhý přes multiparametrické kvadratické programování. Rozdíl mezi nalezenými akčními zásahy se pak akumuluje vlivem zpětné vazby, kde se do regulátoru předává i poslední hodnota nalezeného optimálního akčního zásahu. Vlivem toho lze v grafu pozorovat rozdíl akčních zásahů mezi oběma přístupy k MPC s nenulovou střední hodnotou. Nicméně tento výsledek nelze interpretovat tak, že je chování regulátorů rozdílné. Je to dáno vlastnostmi obou přístupů k prediktivní regulaci. Konkrétně způsobem řešení kvadratického programu, kde se pracuje s tolerancemi, se kterými se vždy musí u optimalizačních úloh počítat.

8 Časové náročnosti standardního a explicitního MPC

Tato práce je zaměřena na rychlé prediktivní řízení. Mimo jiné je tedy jejím cílem i analyzovat samotné časy, které prediktivní regulátory potřebovaly k vygenerování optimálního akčního zásahu. Srovnáním těchto časů při řízení modelu mechatronického ramene standardním prediktivním regulátorem a explicitními prediktivními regulátory se bude zabývat tato kapitola.

Časy, které regulátory počítaly hodnotu řízení, se budou měřit při stejném experimentu, který byl uvažován v dřívějších kapitolách. Nejdříve dojde ke skokové změně referenční hodnoty a poté bude do systému aplikována neměřitelná vstupní porucha. Současně systém bude vybuzen tak, aby se hodnoty akčních zásahů dostaly na svou saturační mez.

Měření časové náročnosti pro simulace v této kapitole bude probíhat na dvou platformách. Nejdříve se k prediktivnímu řízení modelu mechatronického ramene použije standardní stolní počítač, na kterém byly realizovány simulace již dříve. Získá se tak prvotní informace o tom, jak jsou analyzované přístupy k prediktivní regulaci vůči sobě rychlé z hlediska výpočetní náročnosti. Poté se přistoupí k simulacím, které budou realizovány na průmyslovém počítači, jenž se použil k řízení skutečného mechatronického ramene. Právě tyto simulace budou klíčové k analýze výpočetních časů, jelikož se jedná o experimenty z praktické aplikace navržených MPC. Lze předpokládat, že časy potřebné k nalezení optimálního akčního zásahu budou na tomto HW o mnoho nižší.

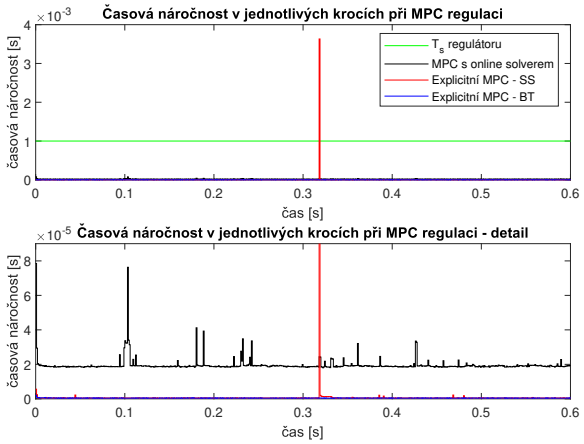
Konkrétně se bude používat stolní počítač s procesorem *AMD Ryzen 5 5600H* s taktovací frekvencí 3.3 GHz a operačním systém Windows 11 Home verze 22H2. Cílovým HW, který se použil k řízení reálného stroje, je průmyslový počítač B+R Automation PC 910 ve variantě s quad-core i5 procesorem a taktovací frekvencí 2.7 GHz. Operační systém je Debian Linux s real-time patchem.

Tato kapitola bude obsahově koncipována následovně. Nejdříve zde budou analyzovány časové náročnosti prediktivních regulátorů při simulacích z Matlabu a z REXYGENu, kde bude algoritmus MPC implementován na popsaném stolním počítači. Další podkapitola bude věnována srovnaní výpočetních časů standardního a explicitního MPC, jejichž algoritmus bude implementován přímo na cílovém HW, kterým je průmyslový počítač.

8.1 Srovnání časových náročností při simulacích z Matlabu

Tato podkapitola bude věnována vyhodnocení časové náročnosti jednotlivých přístupů k prediktivní regulaci z Matlabu na stolním počítači. Konkrétně budou použity tyto způsoby prediktivního řízení. Začne se s MPC regulátorem ve standardním tvaru, kde je nezbytné řešit v každém kroku algoritmu kvadratický program. Při tom se použije solver *qpOASES* v jeho *hot-start* variantě. Dále se bude uvažovat explicitní MPC regulátor, kde se k vygenerování optimálního akčního zásahu budou sekvenčně procházet všechny známé regiony z prostoru stavů. Jako poslední se bude analyzovat časová náročnost u explicitního MPC, který výsledný zákon řízení naleze pomocí binárního vyhledávacího stromu.

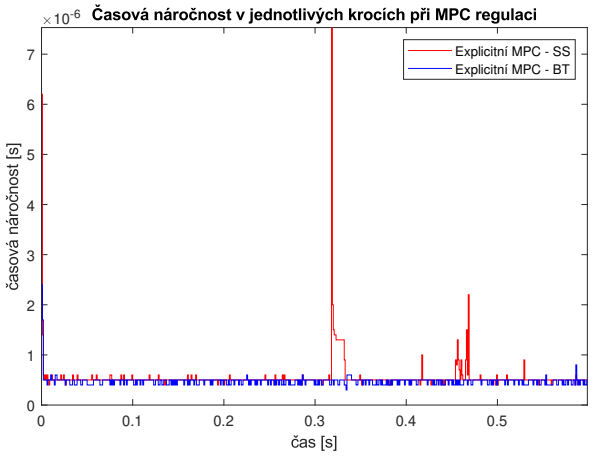
Po spuštění simulací vyšly pro každý zmiňovaný způsob prediktivního řízení následující časové náročnosti.



Obrázek 42: Srovnání časové náročnosti MPC regulátorů v Matlabu

Z horního grafu je zřejmé, že explicitní prediktivní regulátor se sekvenčním prohledáváním stavového prostoru (označený jako *Explicit MPC - SS*) by nebylo možné aplikovat pro řízení skutečného systému při použití uvedeného HW. V jednom okamžiku totiž tomuto regulátoru trval výpočet optimální hodnoty akčního zásahu dokonce delší dobu, než je perioda vzorkování řízeného systému. Ukázalo se, že v tu chvíli musel tento explicitní regulátor postupně projít všechny známé regiony v prostoru stavů, než našel ten, do kterého patří aktuální stav systému. V dolním grafu je ještě detail těchto časů ze stejné simulace. Je vidět, že u standardního MPC se solverem *qpOASES* (označený jako *MPC s on-line solverem*) se doba výpočtu optimálního akčního zásahu pohybovala okolo 0.02 ms. Později se ukázalo, že přibližně tento čas je minimální průměrný čas, kterého lze u této použité implementace solveru *qpOASES* při řešení dané úlohy prediktivní regulace v Matlabu dosáhnout.

Následující graf ještě zobrazuje detail průběhu výpočetních časů u explicitních MPC, aby byl patrnější rozdíl mezi nimi.



Obrázek 43: Srovnání časové náročnosti MPC regulátorů v Matlabu - detail

Jak je vidět, tak po většinu simulace bylo dosaženo u obou typů explicitních regulátorů (sekvenční prohledávání stavového prostoru = SS, explicitní MPC používající binární strom = BT) velice podobné doby výpočtu optimálního akčního zásahu. Při analýze simulace ale

bylo zjištěno, že v těchto okamžicích musel explicitní regulátor se sekvenčním prohledáváním projít pouze jeden region, aby vypočetl optimální akční zásah. Jak již ale bylo zmíněno, tak v jednom okamžiku (po zásahu neměřitelné poruchy), přibližně okolo 0.33 vteřiny simulačního času, musel tento regulátor projít všechny známé regiony. V tu chvíli mu trvala doba výpočtu hodnoty akčního zásahu dokonce delší čas, než je perioda vzorkování systému 1 ms.

Všechny dosažené výsledky shrnuje následující tabulka.

typ MPC	prům. čas výpočtu [μs]	max. čas výpočtu [μs]
standardní	19.5744	78.5
expl. - seq. search	6.5899	3631.6
expl. - binary tree	0.4499	2.1

Tabulka 1: Průměrné a max. časy výpočtu hodnoty řízení MPC při simulacích v Matlabu

Ze simulací z Matlabu je tedy zřejmé, že nejlepších výsledků z hlediska časové náročnosti bylo dosaženo u explicitního prediktivního regulátoru, který k nalezení optimálního akčního zásahu používá binární vyhledávací strom. U této formy explicitního MPC byly časy výpočtu hodnoty řízení dokonce 40x menší než u standardní verze MPC s on-line solverem.

8.2 Srovnání časových náročností při simulacích z REXYGENu

Tato podkapitola bude zaměřena na vyhodnocení časové náročnosti jednotlivých přístupů k prediktivní regulaci z REXYGENu na stolním počítači. Konkrétně se budou analyzovat tyto typy prediktivní regulace. Jako první to bude MPC ve standardní formě, kde se řeší optimalizace on-line v každém kroku. V REXYGENu lze k tomuto účelu použít blok s názvem *qpOASES*, který řeší úlohu kvadratického programování právě pomocí solveru *qpOASES*. Dále se bude pracovat se třemi explicitními MPC regulátory, které k nalezení optimálního akčního zásahu používají binární vyhledávací strom. V první případě je algoritmus na prohledávaní binárního stromu implementován v bloku REXLANG, ve druhém v bloku PYTHON a ve třetím případě se ke stejnému účelu použil blok MPCE.

Experiment zde probíhal tak, že u každého způsobu řízení byla provedena simulace se změnou referenční hodnoty a zásahem poruchy. Přitom se měřily maximální a průměrné časy, které daný regulátor potřeboval k vygenerování optimálního akčního zásahu. Po experimentu vyšly pro každý zmiňovaný způsob prediktivního regulace následující časové náročnosti.

typ MPC	prům. čas výpočtu [ms]	max. čas výpočtu [ms]
standardní	0.1732	0.6586
expl. - REXLANG	0.0656	0.1109
expl. - PYTHON	0.1321	0.5828
expl. - MPCE	0.007	0.0173

Tabulka 2: Průměrné a max. časy výpočtu hodnoty řízení MPC při simulacích v REXYGENu

Z tabulky je zřejmé, že z hlediska časové náročnosti je na tom skutečně nejhůře MPC ve standardním tvaru s on-line solverem. Průměrná doba výpočtu akčního zásahu je zde přibližně 0.17 ms, což je ze všech čtyř analyzovaných typů prediktivní regulace nejhorší. Další tři MPC byly explicitní používající ke generování akčního zásahu binární vyhledávací strom. Nejkratších časů potřebných k nalezení optimálního akčního zásahu bylo dosaženo u zapojení s blokem MPCE. Konkrétně tomuto bloku trvalo vygenerování akčního zásahu průměrně

0.007 ms, což je přibližně 9x kratší doba než u bloku REXLANG, přibližně 19x kratší doba než u bloku PYTHON a dokonce téměř 25x kratší doba než u standardní verze MPC s on-line solverem.

8.3 Srovnání výpočetních časů při simulacích na průmyslovém PC

V této podkapitole proběhne srovnání výpočetních časů jednotlivých přístupů k prediktivní regulaci, kdy algoritmus MPC bude implementován na popsaném průmyslovém počítači. Právě tento HW se použil v této práci k řízení reálného mechatronického ramene. Časová náročnost zde bude analyzována u standardního MPC, kde se musí v každém kroku algoritmu řešit kvadratický program, a u explicitního MPC, kde se k nalezení akčního zásahu využívá blok REXLANG s implementovaným algoritmem na prohledávaní binárního stromu.

Po spuštění simulací vyšly tyto výsledky.

typ MPC	prům. čas výpočtu [μs]	max. čas výpočtu [μs]
standardní	45.4159	204.072
explicitní - REXLANG	6.5391	8.999

Tabulka 3: Průměrné a max. časy výpočtu hodnoty řízení MPC při simulacích na prům. PC

Je zřejmé, že oproti simulacím na stolním počítači zde skutečně bylo dosaženo mnohem kratších výpočetních časů u obou způsobů prediktivní regulace. Vzhledem k periodě vzorkování systému, která je 1 ms, by šlo oba analyzované přístupy k prediktivní regulaci v praxi použít. Nicméně z tabulky je jasné, že kratších výpočetních časů bylo dosaženo u explicitního prediktivního regulátoru. U něj průměrný čas hledání optimálního akčního zásahu byl dokonce 7x menší než u klasické verze MPC.

8.4 Vyhodnocení výsledků

Pro všechny pokusy vyšel nejlépe z hlediska nejkratší doby výpočtu optimálního akčního zásahu explicitní MPC s binárním prohledávacím stromem. Při simulacích v Matlabu na stolním počítači byl tento přístup k prediktivní regulaci 40x rychlejší než standardní verze MPC. Další simulace proběhly v programu REXYGEN. Z výsledné tabulky bylo dobře vidět, že skutečně bylo nejrychlejší doby výpočtů dosaženo v zapojení s blokem MPCE, který je kompilovaný. Čas potřebný k vygenerování optimálního akčního zásahu zde byl oproti standardní verzi MPC rychlejší 25x. U bloku REXLANG se hodnota akčního zásahu počítala 2.5x rychleji než u MPC s on-line solverem a u bloku PYTHON pouze 1.3x rychleji. Z uvedených hodnot je vidět, že se velikost zrychlení mezi explicitní a standardní verzí MPC při použití bloku MPCE v REXYGENu blíží k hodnotě zrychlení mezi těmito formami MPC z Matlabu.

Je tedy zřejmé, že se vytvoření nového bloku MPCE, který je komplikovaný, skutečně z hlediska časové náročnosti vyplatilo. Jak již bylo avizováno dříve, tak v REXYGENu u bloků REXLANG a PYTHON nebylo dosaženo takového zrychlení oproti MPC s on-line solverem, protože tyto bloky jsou interpretované a nikoliv kompilované. Je to z důvodu, že tyto bloky jsou volně programovatelné a nabízejí tak uživateli široké spektrum možností, jak je využít. Nicméně cenou za to je právě jejich rychlost vyhodnocení.

Dále při simulacích na průmyslovém počítači vyšly doby výpočtu u explicitního MPC s blokem REXLANG přibližně 7x rychleji než u verze MPC s on-line solverem. Nicméně při použití kompilovaného bloku MPCE se dá předpokládat, že by bylo dosaženo analogického zrychlení jako u simulací na stolním počítači.

9 Složitost problému

Další důležitou otázkou u metod rychlého prediktivního řízení je, jaký vliv má na celkovou složitost úlohy regulace zvyšování hodnot horizontu predikcí a řízení. V teoretické části diplomové práce již bylo řečeno, že s rostoucími hodnotami horizontů dochází k nárůstu dimenze matic optimalizace a tím pádem narůstá i velikost kvadratického program, který je nutné u prediktivní regulace řešit. Na základě toho lze předpokládat, že dojde i ke zvětšení doby potřebné k nalezení optimálního akčního zásahu. Touto problematikou se bude zabývat tato kapitola.

Při analýze uvedeného problému se zde opět bude pracovat s modelem mechatronického ramene, jehož popis je součástí této práce. Uvažovat se při tom bude algoritmus MPC zaručující nulovou odchylku v ustáleném stavu. U tohoto algoritmu má na složitost úlohy vliv především horizont řízení. Z tohoto důvodu se v této kapitole bude zkoumat složitost úlohy prediktivní regulace právě při změnách této hodnoty.

Váhové matice budou nastaveny na

$$\mathbf{Q} = \begin{bmatrix} 2000 & 0 \\ 0 & 0.1 \end{bmatrix}, \mathbf{R} = 500. \tag{100}$$

Akční veličina bude pro celý horizont řízení saturována do rozmezí ± 0.1. Všechny simulace provedené v této kapitole proběhnou v Matlabu a budou mít délku 0.6 sekundy. Jejich průběh bude takový, že nejdříve dojde ke skokově změně referenční hodnoty a po ustálení přechodového děje zasáhne do systému vstupní porucha.

Při analýze složitosti úlohy se budou uvažovat 2 přístupy k prediktivní regulaci. První z nich je MPC ve standardním tvaru, kde je nezbytné řešit v každém kroku algoritmu kvadratický program. Při tom se použije solver *qpOASES* v jeho *hot-start* variantě. Dále se bude analyzovat složitost úlohy u explicitního MPC, který výsledný zákon řízení určuje pomocí binárního vyhledávacího stromu. Právě u tohoto regulátoru bylo doposud dosaženo nejlepších výsledků z hlediska časové náročnosti.

První podkapitola bude věnována paměťovým nárokům MPC a tomu, jak se mění při nárůstu hodnoty horizontu řízení. Následující podkapitola se bude zabývat tím, jaký vliv má zvyšování horizontu řízení na dobu potřebnou k výpočtu optimálního akčního zásahu pomocí MPC regulátorů.

9.1 Paměťové nároky

Mimo výpočetní rychlosti je u MPC regulátorů nezbytné sledovat i paměťové nároky celé prediktivní regulace. Každá řídicí jednotka realizující MPC má totiž jednak jinou rychlost výpočtů ale také disponuje jinak velkou pamětí, ve které je nezbytné uchovávat hodnoty daného prediktivního regulátoru. Tato podkapitola bude zaměřena na to, kolik konstantních hodnot je nutné uložit do paměti standardního a explicitního MPC s binárním stromem při zvyšování horizontu řízení u regulace modelu mechatronického ramene popsaného výše.

Pro analýzu paměťových nároků byl u standardního MPC a explicitního MPC s binárním stromem proveden následující experiment. Postupně se zvyšoval horizont řízení a přitom se sledovala dimenze všech vektorů a matic, z čehož se určil počet konstantních hodnot, které

se musí uložit do paměti regulátoru. Konstantními hodnotami jsou myšlena reálná čísla se standardní velikostí datového typu double - 64 bitů. Výsledek zachycuje tato tabulka.

počet hodnot v paměti \ hodnota n_c	1	2	3	4	5	6
standardní MPC	41	88	141	200	265	336
explicitní MPC	109	312	989	3850	13954	49616

Tabulka 4: Závislost paměťových nároků u daného typu MPC na hodnotě horizontu řízení

Je zřejmé, že u explicitního MPC je zapotřebí se zvyšujícím se horizontem řízení disponovat mnohem větší pamětí. Ukázalo se totiž, že s rostoucí hodnotou horizontu řízení roste exponenciálně počet regionů v prostoru stavů řízeného systému definující optimální zásah řízení. S tím je spojeno to, že tyto regiony musí být definovány pomocí více nerovností. To samé platí i pro počet optimálních zákonů řízení. Závislost počtu regionů ve stavovém prostoru na hodnotě horizontu řízení u explicitního MPC je v následující tabulce.

hodnota n_c	1	2	3	4	5	6
počet regionů u explicitního MPC	3	9	27	81	243	729

Tabulka 5: Závislost počtu regionů u explicitního MPC na hodnotě horizontu řízení

Shrnutím výsledků z této podkapitoly lze dojít k tomuto závěru. Ačkoliv existují jisté metody pro snížení výpočetní náročnosti explicitního prediktivního regulátoru (např. explicitní MPC se suboptimálním řešením viz [5]), tak explicitní MPC využívající multiparametrické kvadratické programování se jeví vhodné k použití pouze u relativně malých úloh. Konkrétně ve zdroji [5] je doporučováno řídit tímto přístupem systémy s jedním nebo dvěma vstupy a to do deseti stavových proměnných. Současně by se měly volit malé hodnoty horizontů.

V uvažovaném případě u reálné soustavy mechatronického ramene má úloha prediktivní regulace ve výsledku dokonce 12 stavových proměnných. Díky tomu se zvyšujícím se horizontem řízení rapidně narůstají i výpočetní nároky MPC. Vzhledem k nízkému horizontu řízení se tedy jeví jako vhodné pro snížení složitosti úlohy aplikovat na model systému některou z metod pro redukci jeho řádu. Nicméně to by nejspíše vedlo na snížení přesnosti modelu vůči skutečnosti, což by ale mohlo hodně zhoršit nebo dokonce znemožnit řízení reálného systému pomocí prediktivní regulace.

9.2 Výpočetní nároky s rostoucími hodnotami horizontů

Tato podkapitola bude sloužit k porovnání časové náročnosti u řízení modelu mechatronického ramene pomocí standardního MPC a explicitního MPC s binárním stromem při zvyšujících se hodnotě horizontu řízení. Vzhledem k výsledkům z předchozí kapitoly se zvolilo, že se tento problém bude analyzovat pro hodnoty horizontu řízení v rozmezí 1 až 4. Měření časové náročnosti zde bude probíhat na již zmíněném stolním počítači s operačním systémem Windows a procesorem *AMD Ryzen 5 5600H* s taktovací frekvencí 3.3 GHz.

Závislost průměrného času nezbytného k výpočtu akčního zásahu u obou typů MPC na hodnotě horizontu řízení je vidět v následující tabulce.

prům. čas výpočtu \ hodnota n_c	1	2	3	4
standardní MPC [μs]	32.9992	34.2177	36.0069	42.378
explicitní MPC [μs]	0.2755	0.4544	0.7541	1.1819

Tabulka 6: Závislost průměrné doby výpočtu akč. zásahu na horizontu řízení

Z tabulky je zřejmé, že u obou typů MPC se při zvyšujícím se horizontu řízení zvyšuje i průměrná doba výpočtu akčního zásahu, jak se očekávalo. Pro dané hodnoty n_c jsou průměrné časy výpočtu optimálního řízení v průměru 70x menší u explicitního regulátoru než u jeho standardní verze. Na základě výsledků z předchozí podkapitoly lze ale předpokládat, že existuje jistá hodnota n_c, od které bude dosaženo kratší průměrné doby výpočtu u standardního MPC regulátoru. Nicméně vzhledem k velikosti paměťových nároků při vyšších hodnotách horizontu řízení nelze tento experiment pro vyšší hodnoty horizontu řízení prakticky vyzkoušet.

Pro úplnost zde bude ještě uvedena tabulka se závislostí maximální doby výpočtu akčního zásahu u obou typů MPC na hodnotě horizontu řízení.

max. doba výpočtu \ hodnota n_c	1	2	3	4
standardní MPC [μs]	136	135.6	146.6	157.5
explicitní MPC [μs]	2.6	2.2	2.5	2.9

Tabulka 7: Závislost maximální doby výpočtu akč. zásahu na horizontu řízení

Dosažené výsledky v této podkapitole potvrzují závěry z předchozí sekce o paměťových nárocích MPC. Jak bylo vidět z tabulek, tak explicitní prediktivní regulátor s binárním vyhledávacím stromem poskytuje z hlediska výpočetní náročnosti mnohem lepších výsledků než standardní verze MPC u malých úloh. Při větších hodnotách horizontu řízení nebo při snaze řídit systém s vysokým řádem pomocí prediktivní regulace se jeví rozumnější volit standardní verzi MPC s on-line řešením kvadratického programu.

10 Závěr

Tématem a hlavním cílem diplomové práce bylo analyzovat princip metod rychlého prediktivního řízení a následně algoritmy těchto metod implementovat tak, aby se jimi daly řídit mechatronické systémy s krátkými periodami vzorkování.

V teoretické části práce proběhlo nejdříve seznámení s principem prediktivní regulace ve standardním tvaru. V této verzi MPC je nutné v každém kroku algoritmu řešit optimalizační úlohu. Konkrétně zde bylo ukázáno, jak se při regulaci odhaduje budoucí chování systému na základě modelu řízené soustavy. Dále co představují horizonty řízení a predikcí a jaký vliv mají na celou prediktivní regulaci. Následovala sekce o účelové funkci, prostřednictvím které se u prediktivní regulace definuje optimalizační úloha, jejíž vyřešením se získá výstup regulátoru. Poté byl ještě představen algoritmus MPC pro zaručení nulové odchylky v ustáleném stavu, který se v práci nadále využíval.

Nedílnou součástí prediktivního řízení je řešení optimalizační úlohy. Jak bylo v práci ukázáno, tak pro uvažovaný případ je optimalizační úloha ve formě kvadratického programu. Z toho důvodu byla další část práce věnována právě metodám řešení úloh kvadratického programování. Rychlost řešení těchto úloh je zásadní pro výpočetní náročnost celého regulátoru. Nejdříve zde proběhla definice samotného kvadratického programu a poté byly uvedeny KKT podmínky optimality pro tento typ úlohy. Dále byly představeny některé algoritmy a solvery, které lze k řešení úloh kvadratického programování využít. Poslední část této sekce byla věnována představení multiparametrického kvadratického programování, které je základem explicitního prediktivního řízení.

Explicitní prediktivní regulátor je speciální forma prediktivního řízení, kde není potřeba řešit v každém kroku algoritmu kvadratický program. Právě díky tomu by se u této verze MPC mělo dosahovat mnohem lepších výsledků z hlediska časové náročnosti než u MPC ve standardní formě s on-line řešením kvadratické úlohy. Nejdříve zde byl představen algoritmus explicitního prediktivní regulátoru, který je shodný s algoritmem pro řešení úloh multiparametrického kvadratického programování. Vycházelo se z KKT podmínek optimality. Klíčovým problémem u této úlohy bylo prohledávání stavového prostoru tak, aby byly nalezeny všechny regiony, prostřednictvím kterých se vypočetl optimální zákon řízení. Byly představeny 4 způsoby řešení tohoto problému. Ukázalo se, že nejlepších výsledků se dosahuje při použití metody změny aktivní množiny na základě původu omezení. Dalším zásadním problémem u explicitního MPC pro snížení výpočetní náročnosti úlohy bylo určení regionu příslušícího aktuálnímu stavu systému (tzv. *point location problem*). Pro řešení tohoto problému byly představeny 2 metody a to sekvenční prohledávání stavového prostoru a metoda využívající binární vyhledávací strom. Ukázalo se, že v nejhorším případě pro určení optimálního akčního zásahu bylo u metody s binárním stromem potřeba vyhodnotit tolik nerovností, jaká je hloubka stromu. To vedlo na mnohem lepší výsledky než u metody sekvenčního prohledávaní stavového prostoru.

Poté již následovala praktická část diplomové práce. Zde se začalo s popisem skutečného mechatronického systému, který bylo chtěné později řídit pomocí představených metod prediktivního řízení. Jednalo se o mechatronický stend pohonu. Proběhla zde i identifikace tohoto systému pro získání jeho modelu. Jak se ukázalo, tak přesnost tohoto modelu vůči skutečnosti, je u prediktivní regulace důležitým faktorem.

Další část práce byla věnována řízení popsaného reálného systému pomocí prediktivního regulátoru ve standardní formě s řešením kvadratického programu v každém kroku algoritmu. Začalo se s návrhem tohoto regulátoru a představením schémat, která se následně

použila k simulacím chování uzavřené smyčky s vytvořeným regulátorem. Současně zde proběhlo i odzkoušení 3 solverů z hlediska časové náročnosti, které se využily k on-line řešení kvadratického programu při běhu regulátoru. Nejlepšího výsledku bylo dosaženo se solverem `qp-OASES` ve variantě s hot-startem. Po úspěšných simulacích se přistoupilo k řízení skutečného mechatronického systému. I v tomto případě regulátor fungoval dobře a poradil si se změnou referenční hodnoty i zásahem poruchy.

Následoval návrh explicitního prediktivního regulátoru, kde byla snaha ho navrhnout tak, aby se choval naprosto totožně jako dříve vytvořený MPC regulátor s on-line solverem. Poté proběhlo představení schémat z Matlabu a REXYGENu, která se používala k otestování navrženého regulátoru. Přitom se detailně analyzovala verze explicitního MPC, kde se k výpočtu optimálního akčního zásahu používal binární vyhledávací strom. Algoritmus na prohledávaní tohoto binárního stromu byl implementován pomocí interpretovaných bloků REXLANG a PYTHON a kompilovaného bloku MPCE. Z výsledných simulací bylo dobře vidět, že všechny navržené varianty explicitního prediktivního regulátoru fungovaly totožně jako MPC regulátor ve standardním tvaru s on-line solverem. Přistoupilo se tedy k pokusům na reálném stroji, kde se ukázalo, že skutečně explicitní prediktivní regulátor pracuje stejně jako standardní verze MPC.

Další kapitola byla zaměřena na analýzu časové náročnosti u standardní a explicitní verze MPC. Při simulacích v Matlabu na stolním počítači bylo u explicitního prediktivního regulátoru dosaženo 40x rychlejšího výpočetního času než u standardní verze MPC. Poté se přistoupilo k vyhodnocení časové náročnosti z REXYGENu nainstalovaného na klasickém stolním počítači. Bylo zde dobře vidět, že v případě interpretovaných bloků REXLANG a PYTHON u explicitního regulátoru se dosáhlo pouze přibližně 2x rychlejšího výpočtu akčního zásahu než u MPC se solverem `qp-OASES`. U bloku MPCE, který je kompilovaný, došlo ke 25x rychlejšímu výpočtu pro stejný experiment. Další pokusy proběhly na průmyslovém počítači, kde doby výpočtu byly analogické jako při simulacích na stolním počítači, ale současně byly řádově nižší.

Poslední kapitola byla věnována složitosti problému u explicitní a standardní formy prediktivní regulace. Bylo dobře vidět, že s narůstajícími hodnotami horizontu řízení roste i doba, kterou regulátory potřebovaly k vygenerování optimálního akčního zásahu. Jako zásadní se ukázal nárůst paměťových nároků u explicitního MPC se zvyšující se hodnotou horizontu řízení. Provedené experimenty ukázaly, že u uvažovaného případu s modelem mechatronického ramene s každou hodnotou horizontu řízení roste exponenciálně i počet regionů ve stavovém prostoru. S tím je samozřejmě spojený i exponenciální nárůst počtu nerovnic, které tyto regiony definují, nebo počtu optimálních zákonů řízení. Z těchto výsledků se usoudilo, že explicitní prediktivní regulátor se jeví jako velice výhodný pro snížení výpočetní náročnosti u řízení systémů do přibližně 10-ti stavových proměnných. Pro systémy s vyšším řádem se doporučilo zvolit vzhledem k paměťovým nárokům standardní verzi MPC s některým z rychlých solverů kvadratických programů, jako je například `qp-OASES` v jeho hot-start variantě.

Seznam literatury a informačních zdrojů

[1] ROSSITER, J. A.. Model-based predictive control: a practical approach. Boca Raton: CRC Press, c2003. ISBN 0849312914

[2] FERREAU, Hans Joachim. QpOASES Users's Manual [online]. Lovaň, Belgie, 2012. Dostupné z: https://usermanual.wiki/Pdf/manual.2007140309.pdf. Use-r's manual. KU Leuven, Optimization in Engineering Center (OPTEC) and Department of Electrical Engineering

[3] ULLMAN, Fabian. FiOrdOs: A Matlab Toolbox for C-Code Generation for First Order Methods. Zurich, 2011. Masters's thesis. ETH Zurich

[4] MOSEK ApS. The MOSEK optimization toolbox for MATLAB manual. [online]. Danemark, 2015. Dostupné z: https://docs.mosek.com/6.0/toolbox.pdf. MOSEK ApS.

[5] BEMPORAD, Alberto. Explicit Model Predictive Control. V: BAILLIEUL, J., SAMAD, T.. Encyclopedia of Systems and Control. Springer, London, 2019. Dostupné z: http://cse.lab.imtlucca.it/ bemporad/publications/papers/encyclopedia_explicit_MPC .pdf

[6] MIKULÁŠ, Ondřej. Quadratic Programming Algorithms for Fast Model-Based Predictive Control [online]. Prague, 2009 [cit. 2021-04-21]. Dostupné z: https://wiki.control.fel.cvut.cz/mediawiki/images/9/94/Bp_2013_mikulas_ondrej.pdf. Bachelor thesis. Czech Technical University in Prague,Faculty of Electrical Engineering

[7] WÉBR, David. Prediktivní regulátor pro řízení pohybu elektromechanických soustav [online]. Plzeň, 2021 [cit. 2023-03-26]. Dostupné z: https://dspace5.zcu.cz/bitstream /11025/44768/1/Webr_BP.pdf. Bakalářská práce. Západočeská univerzita v Plzni, Katedra kybernetiky

[8] BORRELLI, Francesco, BEMPORAD, Alberto, MORARI Manfred. Predictive control for linear and hybrid systems / c Francesco Borrelli, University of California, Berkeley, Alberto Bemporad, IMT School for Advanced Studies, Lucca, Manfred Morari, ETH Zurich. New York, NY, USA: Cambridge University Press, 2017. ISBN 9781107652873

[9] BEMPORAD, Aalberto, MORARI, Manfred, DUA, Vivek, PISTIKOPOULOS, Efstratios. The explicit linear quadratic regulator for constrained systems. *Automatica*, 2002b, vol. 38, no. 1, p. 3–20.

[10] BAOTIC, Miko. An Efficient Algorithm for Multiparametric Quadratic Programming. Zurich 2002. Technical Report, ETH Zurich.

[11] TØNDEL, Petter, JOHANSEN, Tor Arne, BEMPORAD, Alberto. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, FL, USA, 2001, pp. 1199-1204 vol.2

[12] TØNDEL, Petter, JOHANSEN, Tor Arne, BEMPORAD, Alberto. Evaluation of piecewise affine control via binary search tree. *Automatica*, 2003, vol. 39, no. 5, p. 945–950.

[13] GOUBEJ, Martin, KÖNIGSMARKOVÁ, Jana, KAMPINGA, Ronald, NIEUWENKAMP, Jakko, PAQUAY, Stéphane. Employing Finite Element Analysis and Robust Control Concepts in Mechatronic System Design-Flexible Manipulator Case Study [online]. Appl. Sci. 2021, 11, 3689 [cit. 2023-04-11]. Dostupné z: https://doi.org/10.3390/app11083689. Case study.

Seznam obrázků