ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

DIPLOMOVÁ PRÁCE

2019

NIKITA
MAZURENKO

České vysoké učení technické v Praze

Fakulta strojní

Ústav přístrojové a řídící techniky

Obor: Přístrojová a řídicí technika

PLC Tecomat Foxtrot pro prediktivní řízení dynamických systémů

DIPLOMOVÁ PRÁCE

Vypracoval: Bc. Nikita Mazurenko

Vedoucí práce: prof. Ing. Milan Hofreiter, CSc.

Rok: 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mazurenko** Jméno: **Nikita** Osobní číslo: **424084**

Fakulta/ústav: **Fakulta strojní**

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

Studijní program: **Strojní inženýrství**

Studijní obor: **Přístrojová a řídicí technika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

PLC Tecomat Foxtrot pro prediktivní řízení dynamických systémů

Název diplomové práce anglicky:

Predictive control of dynamics systems by using PLC Tecomat Foxtrot

Pokyny pro vypracování:

1. Prostudujte metody globální optimalizace inspirované chováním živočichů v hejnech.
2. Prostudujte základy prediktivního řízení.
3. Vyberte vhodný hejnový algoritmus a s jeho pomocí vyřešte ve vývojovém prostředí Mosaic prediktivní řízení.
4. Ověřte navržené prediktivní řízení simulačně v prostředí Mosaic.
5. Ověřte navržené prediktivní řízení na vybrané laboratorní úloze využitím Tecomat Foxtrot.

Seznam doporučené literatury:

YANG, Xin-She. Nature-inspired metaheuristic algorithms. 2nd ed. Frome,: Luniver Press, 2010, vi, 148 s. ISBN 978-1-905986-28-6.
Programování PLC Tecomat podle normy IEC 61131-3 v prostředí Mosaic. Teco a. s., Kolin, 2007 (dostupné na www.tecomat.com).
J. Mareš, P. Hrnčiřík: Základy prediktivního řízení, VŠCHT, Praha, 2012.

Jméno a pracoviště vedoucí(ho) diplomové práce:

prof. Ing. Milan Hofreiter, CSc., U12110.3

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

Datum zadání diplomové práce: **31.10.2019** Termín odevzdání diplomové práce: **17.01.2020**

Platnost zadání diplomové práce:

prof. Ing. Milan Hofreiter, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

31. 10. 2019
Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně s tím, že její výsledky mohou být dále použity podle uvážení vedoucího diplomové práce jako jejího spoluautora. Souhlasím také s případnou publikací výsledků diplomové práce nebo její podstatné části, pokud budu uveden jako její spoluautor.

V Praze dne . . . . . Podpis . . . . . .

Poděkování

Rád bych poděkoval především panu prof. Ing. Milanu Hofreiterovi CSc. za odborné rady a cenné připomínky, kterými přispěl k vypracování této bakalářské práce.

Dále bych rád poděkoval svojí rodině a přátelům, kteří mi poskytli nezbytnou podporu během celého studia.

Nikita Mazurenko

Název práce:

PLC Tecomat Foxtrot pro prediktivní řízení dynamických systémů

Autor: Bc. Nikita Mazurenko

Obor: Přístrojová a řídicí technika

Druh práce: Diplomová práce

Vedoucí práce: prof. Ing. Milan Hofreiter, CSc.
Ústav přístrojové a řídící techniky
České vysoké učení technické v Praze

Abstrakt: Diplomová práce je věnována využití optimalizačních algoritmů inspirovaných živou přírodou pro prediktivní řízení. Navržené a programově realizované prediktivní řízení je v práci simulačně ověřeno na lineární soustavě využitím programového prostředí Mosaic a následně na reálné laboratorní úloze.

Teoretická část práce se věnuje metodě prediktivního řízení. Další kapitoly popisují algoritmy globální optimalizace. Větší pozornost je věnována algoritmu diferenciální evoluce a netopýřímu algoritmu.

Praktickou část práce lze rozdělit do dvou částí. První část se zabývá návrhem algoritmu prediktivního řízení s využitím diferenciální evoluce a netopýřího algoritmu. Druhá část popisuje testovaní navrženého algoritmu na simulované soustavě a na reální laboratorní úloze. Dosažené výsledky jsou doloženy časovými průběhy regulovaných veličin.

Klíčová slova: globální optimalizace; prediktivní řízení; diferenciální evoluce; netopýří algoritmus

Title:

Predictive control of dynamics systems by using PLC Tecomat Foxtrot

Author: Bc. Nikita Mazurenko

Specialization: Instrumentation and Control Engineering

Type of thesis: Master's thesis

Thesis supervisor: prof. Ing. Milan Hofreiter, CSc.
Faculty of Mechanical Engineering
Czech Technical University in Prague

Abstract: The master's thesis is devoted to use nature inspired optimization algorithms for model predictive control. The proposed and programmatically executed predictive control is simulated in a linear system and subsequently on a real laboratory system using the Mosaic programming environment.

The theoretical part deals with the model predictive control method. The next chapters focus on nature inspired optimization algorithms. More attention is paid to the differential evolution algorithm and the bat algorithm.

The practical part of the thesis can be divided into two parts. The first part deals with the design of the predictive control algorithm using differential evolution and bat algorithm. The second part describes verification of the designed algorithm with a simulated system and with a real laboratory system. The achieved results are documented by graphs of the control process.

Key words: global optimization; model predictive control; differential evolution; bat algorithm

Obsah

Kapitola 1

Úvod

V dnešní době nelze si představit žádný proces ve kterém není použitá automatizace. S automatizací se setkáváme nejen ve velkých fabrikách a složitých výrobních procesech, ale i v každodenním životě. Jsou automatizovány výrobní linky například letadel a automobilů. Složité systémy regulují proces výroby elektřiny, potravin a léků. Různá zařízení udržují jak požadovanou výšku letu letadla, tak i bezpečnou rychlost auta. Každý den se rozšiřuje oblast automatizovaných procesů, roste kvalita a rychlost regulátorů a objevuji se nové typy zařízení.

Proto pokud je aktuální automatizace procesů, bude také aktuální i problematika systémů řízení. Je zřejmě, že neexistuje universální řídicí systém, který najde uplatnění v každém řízeném procesu. Ale stále se vyvíjí metody řízení, které dokážou pokrýt co nejširší oblast automatizovaných procesů. Do této skupiny patří pokročila metoda řízení procesů – prediktivní řízení.

Hlavním cílem řídicího systému je řízení procesu a zároveň splnění veškerých podmínek a omezení vstupních a výstupních proměnných. V případě, že je k dispozici dostatečně přesný dynamický model procesu, pak tento model spolu se současnými měření reálného procesu mohou být použité k predikci chovaní výstupu systému. Na základě aktuálních měření a predikovaných výstupních trajektorií lze provést změny ve vstupních proměnných systému. Jinými slovy, pří nastavovaní vstupních proměnných je možné odhadnout odezvu systému na tyto provedené změny díky modelu procesu. Na této strategii je založená metoda prediktivního řízení. Pro hledaní optimálních hodnot vstupních proměnných u jednodušších aplikací metoda prediktivního řízení nevyžaduje komplikované výpočty. Naopak při řízení dynamických procesů, požadujících přesnou a rychlou regulaci, je třeba použit vhodný optimalizační algoritmus.

Otázka globální optimalizace má velmi dlouhou historii a je popsaná v mnoha pracích. Vzhledem k tomu, že problematika globální optimalizace se vyskytuje v mnoha různých oborech, pro hledání optima se používá celá množina optimalizačních algoritmů. Rozlišuje se mnoho typů optimalizačních algoritmů klasifikovaných na základě různých kritérií. Rozsáhlý přehled dělení algoritmů lze nalézt v publikaci Umělá inteligence v problémech globální optimalizace [1].

Kapitola 2

Prediktivní řízení

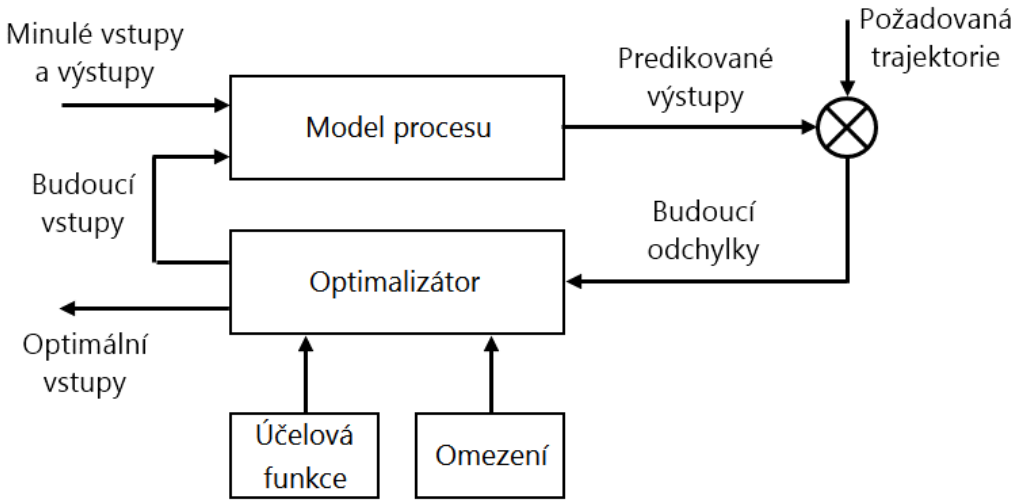
Prediktivní řízení (Model Predictive Control – MPC) má dlouhou historii v oblasti řídicí techniky. Je to jedna z metod řízení, o které se stále zajímají výzkumní pracovníci jak v průmyslové, tak i v akademické komunitě [2]. Důvodem proto je nejenom široký rozsah oblastí uplatnění, ale i to, že prediktivní řízení poskytuje velmi rozumný kompromis mezi optimálnosti a rychlosti výpočtů [3].

MPC nedefinuje specifickou strategii řízení, ale představuje rozsah řídicích metod, které explicitně využívají stanovený model procesu k získaní řídicího signálu minimalizací účelové funkce. Konkrétní metody jsou odlišné v detailech a jsou přizpůsobeny k určitým typům řešených úloh nebo řízených procesů. Proto je důležitější pochopit a dodržovat filozofii prediktivního řízení. Filozofii MPC lze jednoduše popsat následujícím způsobem – předpovídat budoucí chovaní procesu na základě modelu systému, měření nebo odhadů současného stavu systému a hypotetických budoucích trajektorií vstupu nebo zásady zpětnovazebního řízení. Přestože algoritmy využívající tuto filozofii jsou odlišné, lze vyjádřit několik základních principů, které se objevují ve větším nebo menším rozsahu v oblasti prediktivního řízení [4]. Tyto principy jsou následující:

- Explicitní užití modelu systému pro předpověď výstupních veličin procesu v budoucích časových okamžicích.
- Stanovení sekvence řídicích signálů na základě minimalizace zvolené účelové funkci.
- Posunutí horizontu predikce v každém časovém okamžiku do budoucnosti.
- Opakovaní výpočtů nad aktualizovanými daty a realizace prvního řídicího signálu ze sekvence v každém časovém okamžiku.

2.1 Základní komponenty MPC

Následující sekce diplomové práce je věnována popisu základních komponent metody prediktivního řízení. Znalosti těchto komponent a důvodů jejich implementace v řídicím systému mohou sloužit nejenom k pochopení principu činnosti algoritmu, ale i k získaní náhledů na to, jak modifikovat algoritmus pro dosažení konkrétních cílů.



Obrázek 2.1 – Základní komponenty MPC (Převzato z [4])

2.1.1 Model procesu

Jak bylo uvedeno v předchozím textu, metoda MPC neustále pracuje s modelem řízeného procesu. Proto, matematický model je nejdůležitější částí celé struktury algoritmu. Průběh řízení je závislý na kvalitě a přesností modelu soustavy. Matematický model musí nejenom zachycovat dynamiku soustavy, ale i jeho zpracování nesmí být časově náročné. Podle prof. Camacho a prof. Bordons [4] nejčastěji se používají lineární matematické modely v následujících tvarech:

- Impulzní charakteristika
- Přechodová charakteristika
- Přenosová funkce
- Stavová formulace

Hlavní teoretické výsledky MPC týkající se stability pocházejí z formulace stavového prostoru, která může být použita jak pro SISO (*Single-Input-Single-Output* – anglický) systémy, tak i pro MIMO (*Multi-Input-Multi-Output* – anglický) systémy a může být snadno rozšířena na nelineární procesy [4]. Pro zachycení dynamiky lineárních systémů v diskrétním čase pomocí stavové formulace se používají následující rovnice:

$$\begin{aligned} x_m[k+1] &= A_m x_m[k] + B_m u[k] \\ y[k] &= C_m x_m[k] + D_m u[k] \end{aligned} \tag{2.1}$$

kde A_m je matice dynamiky, B_m je matice pravých stran, C_m a D_m jsou výstupní matice systému. Index m zdůrazňuje, že se jedná o původní stavovou proměnnou a matice

modelu, protože rovnice z (2.1) budou následně přepsané do vice vhodného pro MPC tvaru.

Z obecné stavové formulace systému (2.1) je patrno, že vstupní veličina procesu $u[k]$ má přímý vliv na hodnotu výstupu $y[k]$. Ale vzhledem k principu postupujícího horizontu řízení, kdy je pro predikci a řízení požadována aktuální informace o stavu procesu, předpokládá se, že vstup $u[k]$ nemůže okamžitě ovlivnit výstup $y[k]$ [2]. To znamená, že pro model řízeného procesu platí $D_m = 0$ a soustavu (2.1) lze přepsat ve tvaru.

$$\begin{aligned} x_m[k+1] &= A_m x_m[k] + B_m u[k] \\ y[k] &= C_m x_m[k] \end{aligned} \tag{2.2}$$

Obvykle, řídicí systémy využívající prediktivní řízení pracují s přírůstky procesních veličin [4]. To znamená, že místo vstupní veličiny procesu $u[k]$ může být použit přírůstek vstupní veličiny $\Delta u[k]$, a místo výstupu $y[k]$ – přírůstek výstupu řízeného procesu $\Delta y[k]$. Pro dosažení přírůstkového modelu ve stavové formulace je potřeba přepsat soustavu rovnic (2.2) do diferenciálního tvaru:

$$x_m[k+1] - x_m[k] = A_m(x_m[k] - x_m[k-1]) + B_m(u[k] - u[k-1]) \tag{2.3}$$

Pak rozdíl stavové proměnné lze vyjádřit pomocí:

$$\begin{aligned} \Delta x_m[k+1] &= x_m[k+1] - x_m[k] \\ \Delta x_m[k] &= x_m[k] - x_m[k-1] \end{aligned} \tag{2.4}$$

A rozdíl vstupní veličiny:

$$\Delta u[k] = u[k] - u[k-1] \tag{2.5}$$

Kombinací rovnic (2.4) a (2.5) lze rovnice stavového prostoru přepsat ve tvaru:

$$\Delta x_m[k+1] = A_m \Delta x_m[k] + B_m u[k] \tag{2.6}$$

Dalším krokem je propojení stavové proměnné $\Delta x_m[k]$ a výstupní proměnný $y[k]$. Pro realizaci tohoto kroku je třeba nejdřív definovat nový vektor stavových proměnných.

$$\boldsymbol{x}[k] = [\Delta x_m[k] \quad y[k]]^T \tag{2.7}$$

Pro výstupní veličinu taky platí:

$$y[k+1] - y[k] = C_m(x_m[k+1] - x_m[k]) = C_m \Delta x_m[k+1] \tag{2.8}$$

Nakonec, spojení rovnic (2.6) a (2.8) vede k novému tvaru stavové formulace systému v diferenciálním vyjádření:

$$\begin{bmatrix}\Delta x_m[k+1]\\ y[k+1]\end{bmatrix} = \begin{bmatrix}A_m & \boldsymbol{o}_m^T\\ C_m A_m & 1\end{bmatrix}\begin{bmatrix}\Delta x_m[k]\\ y[k]\end{bmatrix} + \begin{bmatrix}B_m\\ C_m B_m\end{bmatrix}\Delta u[k]$$
$$y[k] = [\boldsymbol{o}_m \quad 1]\begin{bmatrix}\Delta x_m[k]\\ y[k]\end{bmatrix} \tag{2.9}$$

kde $\boldsymbol{o}_m$ je nulový vektor. Použitím, definovaného podle (2.7), vektoru $\boldsymbol{x}[k]$ a rozšířených matic A, B, C lze stanovit finální verzi stavové formulace řízeného procesu:

$$\boldsymbol{x}[k+1] = A\boldsymbol{x}[k] + B\Delta u(k)$$
$$y[k] = C\boldsymbol{x}[k] \tag{2.10}$$

2.1.2 Účelová funkce a omezení

Účelová funkce

Matematický model soustavy stanovuje, jaký proces je řízen algoritmem, ale zvolená účelová funkce definuje pravidla regulace. V různých variacích MPC algoritmů se objevují účelové funkce v různých tvarech [4]. Nicméně obecným cílem účelových funkcí je, nastavovat budoucí výstup systému $y[k]$ tak, aby následoval stanovenou požadovanou trajektorii $w[k]$. Obecným příkladem je účelová funkce definována podle následujícího vzorce [4]:

$$F_{cost} = \sum_{i=1}^{N_p} (w[k+i] - \hat{y}[k+i])^2 \tag{2.11}$$

kde F_{cost} – účelová funkce, $w[k]$ – požadovaná veličina v diskrétním čase k, $\hat{y}[k]$ – predikovaná trajektorie výstupu systému a N_p je horizont predikce.

Omezení

Řízení procesu na základě pouze účelové funkce může být problematické. Příkladem může být nerealizovatelnost některých hodnot vstupu procesu z důvodu omezení na straně zařízení, nebo zakázané pásmo výstupních hodnot procesu z bezpečnostních důvodu. Proto připadá v úvahu definovat určitá omezení parametrů algoritmu.

$$y[k] \in \langle y_{min}, y_{max}\rangle, pro\ \forall k$$
$$u[k] \in \langle u_{min}, u_{max}\rangle, pro\ \forall k \tag{2.12}$$

Definovat intervaly povolených hodnot pro vstup a výstup řízeného systému lze například podle vzorků (2.12). Ale neexistují žádná pravidla pro návrh omezení regulace, protože tyto omezení jsou striktně závislá na řízeném procesu.

2.1.3 Optimalizátor

Další základní součástí strategie MPC je optimalizátor. Hlavní činnost optimalizátoru spočívá v poskytnutí sekvence optimálních akčních zásahů. Během každého výpočetního cyklu optimalizátor poskytuje matematickému modelu procesu posloupnost akčních signálu a následně, ve spoluprací s účelovou funkcí a definovanými omezeními, hledá optimální kombinaci budoucích vstupních veličin řízeného procesu:

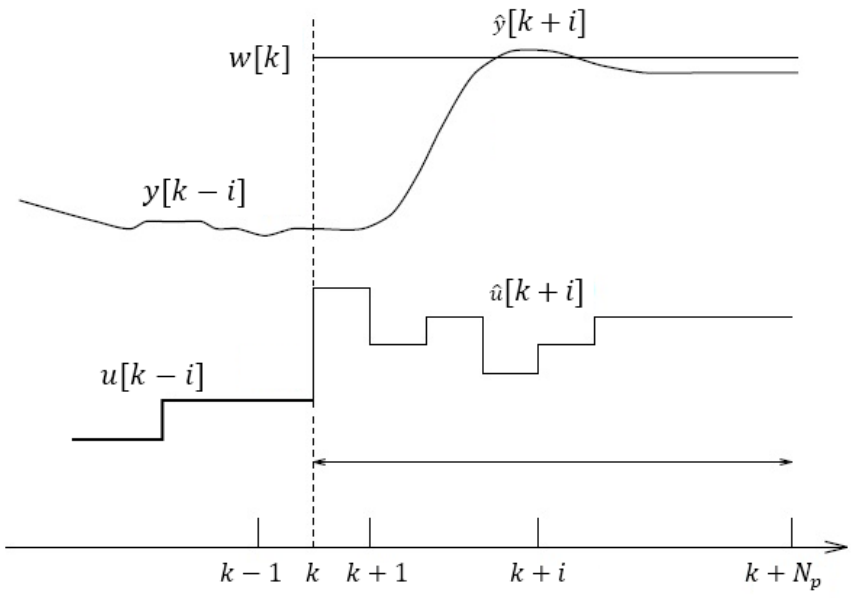
$$\hat{\boldsymbol{u}} = [\hat{u}_{k+1}, \hat{u}_{k+2} \cdots \hat{u}_{k+i-1}], kde\ i = N_p \tag{2.13}$$

kde $\hat{\boldsymbol{u}}$ je posloupnost predikovaných akčních zásahů, N_p je horizont predikce.

Velikost optimalizačního problému závisí na počtu proměnných, velikostí množiny možných hodnot akčních veličin a na délce horizontu predikce. Postupná analýza každé možné kombinace pravděpodobně přivede k nejpřesnějším výsledkům, ale časová náročnost těchto výpočtů způsobí nepoužitelnost dane metody. Proto se pro složitější úlohy používají různé metody globální optimalizace. Řešení problému optimalizace v rámci této diplomové práce je věnována kapitola 3.

2.2 Strategie prediktivního řízení

V předchozí části kapitoly byly popsány základní komponenty, ze kterých se skládá metoda prediktivního řízení. Tyto komponenty jsou: model řízeného procesu, optimalizátor, účelová funkce a definované omezení. Model procesu se využívá k simulace dynamických vlastností systému a vypočtu výstupních hodnot procesu. Ke každé možné hodnotě akčního zásahu se počítá příslušná hodnota účelové funkce, která rozhoduje o kvalitě akčního zásahu. Funkce optimalizátoru spočívá v generovaní nových možných hodnot akčního zásahu a hledaní optimálního. Princip spolupráce těchto komponent je definován strategií prediktivního řízení, která je znázorněna na následujícím obrázku:



Obrázek 2.2 - Strategie prediktivního řízení (Převzato z [4])

Hlavními zdroji, které byly použité pro popis základní strategie metody prediktivního řízení, jsou knihy „Model Predictive Control – 2nd edition" [4] a „Model Predictive Control System Design and Implementation Using MATLAB" [2].

Celý proces prediktivního řízení lze rozdělit na několik důležitých bodů, které se stále cyklicky opakují:

1. V každý časový okamžik k, pomocí stanoveného modelu řízeného procesu, jsou predikovány budoucí hodnoty výstupu systému. Délka posloupnosti budoucích výstupních hodnot je omezena parametrem N_p, kteremu se říká horizont predikce. Tyto predikované výstupní hodnoty $\hat{y}[k+i]$, kde $i = 1, 2 \dots N_p$, jsou závisle na dvou faktorech. Prvním faktorem jsou známé hodnoty vstupních a výstupních proměnných, které byly realizovány a změřeny do časového okamžiku $k-1$. Druhým faktorem jsou budoucí hodnoty akčního zásahu $\hat{u}[k+i]$, kde $i = 0, 1 \dots N_p - 1$, které zpracovává model řízeného procesu.
2. Výpočet budoucích hodnot akčního zásahu se provádí na základě optimalizace definované účelové funkce. Cílem optimalizace je udržení predikovaných hodnot výstupu systému co nejblíže k požadovaným hodnotám $w[k+i]$.
3. Z celé stanovené sekvence budoucích akčních zásahů, v řízeném procesu je realizovaná pouze první hodnota $u[k]$, a ostatní hodnoty řídicího signálu jsou odmítnuty. Důvodem pro realizaci pouze jedné hodnoty je to, že v následující časový okamžik $k+1$ je známa nejenom predikovaná (vypočtená) výstupní veličina $\hat{y}[k+1]$

ale i reálná (změřená) hodnota $y[k+1]$. Proto musí být znovu provedeny operace z prvního kroku a veškeré predikované vstupní a výstupní veličiny musí být aktualizovány.

2.2.1 Optimalizační problém

Jak je zřejmě ze strategie prediktivního řízení, optimalizátor hraje velmi důležitou roli při regulaci procesu – optimalizace účelové funkce. V případě řízení systému účelová funkce reprezentuje kvadratickou odchylku mezi aktuálním stavem procesu a jeho požadovanou veličinou (pro podrobný popis účelové funkce viz kapitola 2.1.2). Proto optimalizace spočívá v hledaní tokových hodnot akčního zásahu, pro které bude hodnota účelové funkci (kvadratické odchylky) minimální.

Hledaní optimální hodnoty z celého intervalu povolených hodnot může být problematické. S rostoucím rozlišením a intervalem povolených hodnot, výrazně roste počet možných akčních veličin, ze kterých je potřeba vybrat optimální. Pro systémy řízení využívající metodu MPC situace s hledáním optimálního řešení je ještě složitější tím, že je potřeba vybrat optimální hodnotu nejenom pro jeden určitý časový okamžik, ale pro každý okamžik z celého horizontu predikce. U lineárních systémů tento optimalizační problém se řeší například využitím kvadratického programování [5]. Ale použití stejných metod i pro nelineární systémy vede k složitějším časově náročným výpočtům. Následkem je výrazné zkrácení délky horizontu predikce. A velmi krátký horizont predikce pak nivelizuje veškeré výhody implementace řídicího systému na základě MPC. Cílem dané diplomové práce je návrh a následné testování řídicího algoritmu s možným řešením problému globální optimalizace MPC pro nelineární systémy.

2.2.2 Návrh řešení

Optimalizační problém řídicích algoritmů využívajících MPC pro nelineární systémy je spojen s množstvím hodnot řídicího signálu, které mohou být predikované a následně realizované. Pro odstranění tohoto problému je navržen nový řídicí systém, do kterého budou implementované dvě řešení. První řešení se týká počtu možných hodnot řídicí veličiny. Druhé řešení je spojeno s metodou hledaní optimálních hodnot akční veličiny bez analýzy každé možné hodnoty.

Omezení změn akčního zásahu

Jak již bylo popsáno v předchozím textu o základních komponentách metody MPC, a hlavně o využití modelu systému, prediktivní řízení nepracuje přímo s hodnotou vstupní veličiny procesu, ale s jejím přírůstkem. Každá další hodnota akční veličiny je stanovena ze vztahu:

$$u[k+1] = u[k] + \Delta u[k] \tag{2.14}$$

Proto lze říct, že hlavním cílem optimalizátora je hledaní optimálních přírůstků akčních zásahů. A výstupem optimalizátoru bude posloupnost těchto přírůstků, kterou lze zapsat ve tvaru:

$$\Delta \boldsymbol{u} = [\Delta u_{k+1}, \Delta u_{k+2}, \dots, \Delta u_{k+i-1}], kde\ i = N_p \tag{2.15}$$

A tady právě lze uplatnit první řešení problému prediktivního řízení – omezit počet hodnot, se kterými bude pracovat optimalizátor. Omezení počtu možných hodnot akčních zásahu pravděpodobně přivede k poklesu kvality regulace. Například, když bude pro regulaci zvoleno pouze 10 možných hodnot řídicího signálu z celého dostupného intervalu, může se stát, že reálná optimální hodnota akčního zásahu bude ležet někde mezi dvou sousedních vybraných možných hodnot.

Ale vhodným omezením pouze počtu možných přírůstků akčních zásahů, se kterými pracuje optimalizátor, lze se tomuto problému vyhnout. Protože i když je počet možných přírůstků je omezen, optimální hodnoty řídicího signálu lze dosáhnout sekvencí několika přírůstků akční veličiny. Pro navržený algoritmus je definovaná sada možných přírůstků řídicího signálu:

$$\begin{aligned} \Delta u[k] \in K\{-10, -5, -2, -1, 0, 1, 2, 5, 10\} = \\ = \{-10K, -5K, -2K, -1K, 0, 1K, 2K, 5K, 10K\} \end{aligned} \tag{2.16}$$

Tato sada možných hodnot byla vybrána analogicky s mincemi. Kombinací těchto devíti hodnot lze získat libovolné celé číslo. A konstanta K slouží pro specifikaci hodnot přírůstků akčních zásahu pro konkrétní řízeny proces. Vhodnou volbou konstanty K lze dosáhnout lepší kvality regulace.

Ze vzorce (2.16) je patrno, že vzdálenost dvou sousedních prvků není konstantní pro celou sadu možných hodnot. To znamená, že pravděpodobnost použití prvků během řízení není rovnoměrná. Řešení problému nerovnoměrností je velmi jednoduché – nastavit optimalizátor tak, aby veškeré optimalizační procesy se prováděly nad indexy

možných hodnot (0, 1, ...). A až v momentě dosažení možných hodnot do modelu procesu, použit hodnotu ze sady (2.16) která odpovídá stanovenému indexu.

Nové metody globální optimalizace

Samotné omezení počtu možných hodnot akčních zásahů neodstraní optimalizační problém pro delší horizont predikce. Analýza všech možných hodnot pro každý bod horizontu predikce vede k časově náročným výpočtům. Proto je nezbytně nutné použit výkonný algoritmus globální optimalizace. Proto dalším řešením implementovaným pří návrhu řídicího systému je použití dvou optimalizačních algoritmů, které patří do skupiny metaheuristických algoritmů.

Kapitola 3

Globální optimalizace

Problémy globální optimalizace se vyskytují ve většině oborů, kterými jsou například strojírenství, matematika, fyzika, ekonomie, a dokonce i společenské vědy. Vzhledem k tomu, že v reálnem světe zdroje, čas a peníze jsou vždy omezeny, je třeba najít způsob, jak optimálně využit tyto důležité zdroje pro řešení definovaného problému. Následkem širokého rozsahu oblastí, ve kterých se řeší problémy globální optimalizace, je fakt, že neexistuje žádná univerzální metoda optimalizace. Ale vývoj informačních technologií ve dnešní době dělá počítačové simulace nepostradatelným nástrojem pro řešení optimalizačních problémů pomocí matematických nástrojů a efektivních vyhledávacích algoritmů.

3.1 Formulace problému

Pro řešení jakéhokoliv problému je nezbytně nutně tento problém definovat. Většinu optimalizačních problémů lze matematické vyjádřit v obecném trávu [6]:

$$minimize\ f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathbb{R} \times n \tag{3.1}$$

kde $f(\boldsymbol{x})$ je funkce vektoru

$$\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \tag{3.2}$$

Vektor $\boldsymbol{x}$ tvoří prostor parametrů a je definován na základě úlohy, která musí být řešena během optimalizačního procesu. Prvky x_i jsou proměnné nebo parametry ovlivňující řešenou úlohu. Jinými slovy každá kombinace hodnot prvků $x_1, x_2, \dots, x_n$ tvoří jednotlivá řešení problému.

Funkce $f(\boldsymbol{x})$ představuje účelovou funkci (*objective function* nebo *cost function* [6]), která slouží k posouzení kvality jednotlivých řešení problému. Ke každé kombinace možných řešení problému se přiřadí vypočtena hodnota účelové funkce. Pak činnost optimalizačního procesu spočívá v hledaní hodnot prvků vektoru $\boldsymbol{x}$ pro které hodnota účelové funkce bude minimální.

3.2 Hledání optima

V případě správného formulování optimalizačního problému, hlavním dalším úkolem je výběr vhodného postupu a metody pro řešení definovaného problému.

Algoritmy globální optimalizace lze klasifikovat mnoha způsoby. Jedním ze základních principů klasifikace je rozdělení algoritmů do dvou kategorií: deterministické a stochastické. Deterministické algoritmy se řídí přísnou procedurou – cesta, funkce a hodnoty proměnných jsou opakovatelné [7]. Na druhé straně jsou stochastické algoritmy, které vždy mají nějakou náhodnost. Genetické algoritmy jsou dobrým příkladem. Možná řešení v populaci se budou lišit při každém spuštění programu, z důvodu použití některých pseudonáhodných čísel. I když v konečných výsledcích nemusí být žádný velký rozdíl, cesty ke každému řešení nejsou přesně opakovatelné [7].

Existuje i třetí typ optimalizačních algoritmů, který je směsí deterministických a stochastických algoritmů. Jsou to tak zvané metaheuristické algoritmy. Je nutně zdůraznit, že neexistuje žádné stanovené definice metaheuristiky [7]. Každopádně je pravda, že většina metaheuristických algoritmů je inspirována živou přírodou.

3.3 Optimalizační algoritmy inspirované živou přírodou

Metaheuristika je metodou pokusu a omylu – cílem je najít dobrá (v ideálnem případě optimální) řešení v přijatelném časovém intervalu, ale nikdy není zaručeno, že podobná řešení existují [7]. Stejné pravidlo platí í v přírodě. Proto existuje celá řada metaheuristických algoritmů inspirovaných živou přírodou.

Tyto algoritmy lze rozdělit na dvě skupiny – algoritmy, které jsou zaměřené na práci s populací, a algoritmy, které jsou zaměřené na práci trajektorii [7]. Zaměření na populaci znamená, že pro generovaní nových řešení je potřeba spolupráce s dalšími jedinci z populace. Pro algoritmy z druhé skupiny platí, že generovaní nových řešení je založeno na pohybu jedinců v definovaném prostoru nezávisle na ostatních řešeních. I když nové řešení je horší než původní, stále existuje nějaká pravděpodobnost, že toto řešení bude přijato. Protože existuje nenulová pravděpodobnost, že tato trajektorie pak může dosáhnout globálního optima.

V rámci zadané diplomové práce bude prostudován jeden optimalizační algoritmus ze skupiny algoritmů zaměřených na práce s populací, a jeden algoritmus ze skupiny pracující s trajektoriemi. Prvním je algoritmus diferenciální evoluce. Druhým použitým algoritmem je netopýří algoritmus.

Kapitola 4

Diferenciální evoluce

Ve 3. kapitole bylo uvedeno, že řešení problémů globální optimalizace je společným bodem jak pro vědu, tak i pro inženýrství. Ale v ideálním případě nemusí samostatné řešení optimalizačních problémů být složité. Například, strukturální inženýr, který má expertní znalosti v oblastí stavebnictví, nemusí mít na stejné vysoké úrovní znalosti i z optimalizační teorie jen proto, aby správně navrhnul a realizoval požadovanou konstrukci. Navíc, optimalizační algoritmus by měl být dostatečně výkonným proto, aby dokázal konvergovat k reálnému optimu. Na druhou stranu, nesmí výpočetní čas algoritmu trvat příliš dlouho. Z uvedených předpokladů plyne, že kvalitní metoda globální optimalizace má být jednoduše implementovatelná, snadno použitelná, spolehlivá a časově nenáročná. Jednou z takových metod je právě algoritmus diferenciální evoluce [8].

4.1 Úvod

Stejně jako téměř všechny evoluční algoritmy, DE algoritmus je optimalizátorem, který začíná řešení problému ve vice, náhodně vybraných, počátečních bodech. Přednastavené omezení parametrů definují oblast, ze které jsou vybrány vektory v této počáteční populaci. Podobně jiným metodám globální optimalizace, nové vektory jsou poruchami (modifikacemi) již existujících vektorů. Ale na rozdíl od těchto metod, tyto odchylky od původních vektorů DE jsou zmenšeným rozdílem dvou náhodně vybraných populačních vektorů. Zkušební vektor se generuje následným součtem zmenšeného rozdílu dvou vektorů a třetího, náhodně vybraného, vektoru z předchozí populace. V případě, že nové řešení, reprezentované zkušebním vektorem, splňuje definovaná kritéria líp než původní, do nové populace na místo původního vektoru postoupí zkušební vektor. Pro popis algoritmu DE a procesů probíhajících během jeho vykonaní byla použita knížka „Differential Evolution. A Practical Approach to Global Optimization“ [7].

4.2 Struktura populace a její inicializace

Diferenciální evoluce pracuje s populacemi stejným způsobem jak i většina optimalizačních algoritmu smíseného typu. Populaci tvoří množina vektorů, kde každý vektor parametrů představuje možné řešení optimalizačního problému:

$$\begin{aligned} P_{\boldsymbol{x}} &= (\boldsymbol{x}_j), \;\; j = 0{,}1 \ldots N_{specimen} - 1 \\ \boldsymbol{x}_j &= (x_{i,j}), \;\; i = 0{,}1 \ldots N_p - 1 \end{aligned} \tag{4.1}$$

kde $P_{\boldsymbol{x}}$ je aktualní populace a $\boldsymbol{x}_j$ je jedinec (vektro) z této populace.

Před inicializací první populace musí být zadány horní a dolní hranice pro každý parametr. Jakmile byly stanoveny hranice inicializace, generátor náhodných čísel přiřadí každému parametru každého vektoru hodnotu z předepsaného rozsahu. Například počáteční hodnota i-tého parametru j-tého jedince z populace může být stanovena podle následujícího vzorce:

$$x_{j,i} = \mathrm{rand}_i(0,1) \cdot \left(b_{i,H} - b_{i,D}\right) + b_{i,D} \tag{4.2}$$

kde $b_{j,H}$ je horní hranice pro vybraný parametr a $b_{j,D}$ je pak dolní hranice.

Funkce $\mathrm{rand}_j(0,1)$ je generátorem náhodných čísel s rovnoměrným rozdělením, která vrací hodnoty z intervalu $[0,1)$. Je také důležité, že tato funkce má index i. Tím se zdůrazňuje fakt, že náhodné číslo se generuje znovu pro každý parametr vektoru $\boldsymbol{x}_j$.

4.3 Mutace

Jakmile je dokončena inicializace, DE začíná proces mutace náhodně vybraných vektorů, aby vytvořily přechodnou populaci $P_{\boldsymbol{v}}$ šumových vektorů $\boldsymbol{v}_j$:

$$\begin{aligned} P_{\boldsymbol{v}} &= (\boldsymbol{v}_j), \;\; j = 0{,}1 \ldots N_{specimen} - 1 \\ \boldsymbol{v}_j &= (v_{i,j}), \;\; i = 0{,}1 \ldots N_p - 1 \end{aligned} \tag{4.3}$$

Zvláštním rysem algoritmu diferenciální evoluce je to, že pro vytváření nového jedince je potřeba ne dvou, ale čtyř vektorů [1]. Prvním vektorem je původní jedinec aktuální populace, další tři – jsou pomocné, náhodně vybrané vektory z aktuální populace. Ze vzorce (4.4) je patrno, jak pomoci těchto tří jedinců stanovit, tak zvány, šumový vektor. Také ze vzorce (4.4) je vidět, že první původní jedinec není použit během mutace, ale hraje důležitou roli v dalších procesech vytváření nové populace.

$$\boldsymbol{v}_j = \boldsymbol{x}_{\mathrm{r}0} + F \cdot (\boldsymbol{x}_{r1} - \boldsymbol{x}_{r2}) \tag{4.4}$$

kde F je mutační konstanta, $\boldsymbol{x}_{\mathrm{r}0}$ až $\boldsymbol{x}_{\mathrm{r}2}$ jsou nahodně vybrane vektory.

Mutační konstanta je kladné reálné číslo, které řídí rychlost vývoje populace. I když neexistuje žádná horní hranice pro hodnoty mutační konstanty, efektivní hodnoty patří většinou do intervalu (0, 1).

Náhodně vybrané pomocné vektory $\boldsymbol{x}_{\mathrm{r0}}$, $\boldsymbol{x}_{\mathrm{r1}}$, $\boldsymbol{x}_{\mathrm{r2}}$, kromě toho, že se liší od původního vektoru, také jsou i navzájem odlišné. Navíc tyto vektory jsou vybrané jenom pro jednoho jedince, jinými slovy – jsou znovu náhodně vybrané ke každému jedinci z aktuální populace.

4.4 Křížení

Další zvláštnost DE je proces křížení, který se uskutečňuje až po mutaci a generovaní šumových vektorů (např. u genetických algoritmů nejdřív se provádí křížení a pak mutace). Každý jedinec ze současné populace je pak rekombinován s příslušným šumovým vektorem za vzniku zkušební populace:

$$\begin{gathered} P_{\boldsymbol{u}} = \left(\boldsymbol{u}_j\right),\ \ j = 0{,}1 \ldots N_{specimen} - 1 \\ \boldsymbol{u}_j = \left(u_{i,j}\right),\ \ i = 0{,}1 \ldots N_p - 1 \end{gathered} \tag{4.5}$$

kde $P_{\boldsymbol{u}}$ je zkušební populace která se skládá ze zkušebních jedinců (vektorů) $\boldsymbol{u}_j$.

Křížením se vytváří zkušební vektory z hodnot parametrů, které byly zkopírovány ze dvou různých vektorů – původního, doposud nepoužitého, vektoru z aktuální populace a příslušného šumového vektoru. Zkušení vektor se vytváří pomoci prahu křížení [1] (konstanta Cr – v angličtině se používá termín *crossover probability*). Par korespondujících parametrů se cyklický vybírá z původního a šumového vektoru (oba první parametry, oba druhé parametry atd) a ke každému páru parametrů se generuje náhodné číslo. V případě, že náhodné číslo je menší než definovaná hodnota prahu křížení, do příslušného parametru ve zkušebním vektoru se zapíše hodnota ze šumového vektoru. V opačném případě se použije hodnota z původního jedince. Lze tento proces zapsat do následujícího vzorce:

$$u_{i,j} = \begin{cases} v_{i,j} & if\ (rand_i < Cr)\ or\ (i = i_{rand}\ ) \\ x_{i,j} & \end{cases} \tag{4.6}$$

Může se ale stát, že při malém počtu definovaných parametrů ve vektorech nebo při velmi malé hodnotě konstanty Cr dojde k duplikace šumového a zkušebního vektorů. Pro odstranění tohoto problému, během generovaní zkušebního vektoru jeden náhodně vybraný parametr z šumového vektoru se přesune do zkušebního i když vygenerované náhodné číslo bude větší než konstanta Cr.

V momentě, kdy jsou následně realizované procesy mutace a křížení pro každého jedince z aktuální populace, přechází se do další fázi algoritmu diferenciální evoluce – selekce.

4.5 Selekce

Během selekce každý zkušební jedinec soutěží s příslušným původním vektorem o místo v nove populaci. Hlavním kritériem pro vyber jednoho z těchto vektorů je jejich hodnota účelové funkce. Pokud zkušební vektor $\boldsymbol{u}_j$ má stejnou nebo nižší hodnotu účelové funkce než hodnota jeho původního vektoru $\boldsymbol{x}_j$, nahradí původní vektor v příští populaci.

$$\boldsymbol{x}_j = \begin{cases} \boldsymbol{u}_j & if\ \left(F_{cost}(\boldsymbol{u}_j) \leq F_{cost}(\boldsymbol{x}_j)\right) \\ \boldsymbol{x}_j & \end{cases} \tag{4.7}$$

Jakmile je nová populace sestavena, proces mutace, křížení a selekce se opakuje, dokud se nenajde optimální řešení, nebo dokud není splněna předem stanovená podmínka ukončení algoritmu (např. počet generací dosáhne přednastaveného maxima).

4.6 Pseudokód algoritmu diferenciální evoluce

Pro lepší pochopení principu činností algoritmu diferenciální evoluce na obrázku 4.1 je znázorněn pseudokód optimalizačního algoritmu.

```
Definice účelové funkce F_cost
Inicializace populace jedinců x_j podle prototypu
Nastaveni počátečních parametrů F, Cr, r_j pro x_j

while (ukončovací podmínka algoritmu)
  ke každému jedinců z populace náhodný výběr třech
  pomocných vektorů z populace

  realizace procesu mutace
  if (rand < Cr) or (i = i_rand)
    vyber parametru z šumového vektoru pro
    zkušebního jedince
  else
    vyber parametru z původního vektoru pro
    zkušebního jedince
  end if
  if ((rand < A_j) and (F_cost(x_j) < F_cost(x_best)))
    uplatnění nových řešení
  end if
  vyhodnocení nových řešení a hledaní nejlepšího jedince
end
```

Obrázek 4.1 - Pseudokód algoritmu diferenciální evoluce (Převzato z [8])

Kapitola 5

Netopýři algoritmus

Metaheuristické algoritmy, jako je optimalizace hejnem částic (*particle swarm optimization*), algoritmus světlušek (*firefly algorithm*) a harmonické vyhledávání (*harmony search*), se nyní stávají účinnými metodami pro řešení mnoha náročných problémů s optimalizací. V roce 2010 pan Xin-She Yang vyvinul nový algoritmus, který se ukázal jako velmi účinný nastroj globální optimalizace – netopýří algoritmus [7]. Netopýří algoritmus (anglický – *Bat Algorithm*[1]) je jedním z nejnovějších algoritmu inspirovaných živou přírodou. Následující kapitola poskytuje podrobný úvod do netopýřího algoritmu, jeho základních procesů a principu činnosti.

5.1 Úvod

Podobně ostatním optimalizačním algoritmům, které jsou inspirované živou přírodou nebo chováním živočichů v hejnech, činnost netopýřího algoritmu je založená na práci s populacemi jedinců – vektorů reprezentujících možné řešení optimalizačního problému. Ale na rozdíl od evolučních algoritmů, ke kterém například patří diferenciální evoluce, ve kterých nová řešení vznikají vzájemnou rekombinací a mutací jedinců z aktuální generace, generovaní nových řešení a hledaní optima v BA je založeno na virtuálním pochybu jedinců. V případě algoritmu DE parametry každého vektoru jsou chápany jako geny, které se mění během procesů mutace a křížení od jedné generace do druhé (podrobně je tomu věnována kapitola 4). Jiným případem jsou vektory netopýřího algoritmu – každý element vektoru je koordinátou v prostoru pro daného jedince. Proto každý vektor z populace lze chápat jako polohu jedince v prostoru, a změnu parametrů těchto jedinců – jako změnu jejich polohy v prostoru. Ale slouží to pouze jako představa o tom, jak funguje BA a neklade to žádné omezeni na velikost vektoru (počet parametrů není omezen rozměrem prostoru). Je dobrou otázkou – když se jedna pouze o pochyb v prostoru, proč pro tento algoritmus jsou vybrány netopýry? Jako odpověď na tuto otázku slouží další část diplomové práce.

[1] V dalším textu diplomové práce, při popisu netopýřího algoritmu bude použitá anglická verze zkratky názvu algoritmu – BA.

5.2 Chování netopýrů

Netopýři jsou fascinující zvířata. Jsou to jediní savci s křídly, které mají také pokročilé schopnosti echolokace. Odhaduje se, že existuje víc než 950 různých druhů netopýrů, které představují až 20 % všech druhů savců ve světě [7]. Jejich velikost sahá od drobného thajského netopýra (s přibližnou vahou 1,5 až 2 g) až po obří netopýry s rozpětím křídla kolem 2 m a váhou do 1 kg. Většina netopýrů do jisté míry používá echolokaci. Mezi všemi druhy jsou ty nejmenší netopýry slavným příkladem, protože používají echolokaci značně, zatímco obří netopýry echolokaci používají málo [7].

Většina malých netopýrů jsou hmyzožravci. Tyto netopýry používají speciální typ sonaru, který se nazývá echolokace, k detekci kořisti, vyhýbání překážkám a lokalizaci jejich trhlin v temnotě. Tyto netopýry vydávají velmi hlasitý zvukový pulz a poslouchají ozvěnu signálu, který se odráží od okolních objektů.

Je zřejmé, že některé netopýry mají dobrý zrak a většina netopýrů má také velmi citlivý čich. Proto ve skutečnosti budou netopýry používat kombinaci všech smyslů pro maximalizaci účinné detekce kořisti a kvalitní navigace. V rámci netopýřího algoritmu globální optimalizace se požívá pouze schopnost netopýrů k echolokaci a jejich chování [6]. Echolokační chovaní netopýrů je pak formulováno tak, aby bylo spojeno s účelovou funkcí, která má být následně optimalizována.

5.3 Virtuální netopýry

Reálné netopýry jsou velmi zajímavé a jejich chovaní v přírodě, jak se schopnosti k echolokaci, tak i bez ní, je neuvěřitelné komplikované a je závisle na mnoha faktorech. Proto některé charakteristiky netopýrů (hlavně ty, které jsou spojeny s echolokací) jsou idealizované. Tyto modifikované pro optimalizační algoritmus netopýry se nazývají virtuální netopýry. Pro každého virtuálního netopýra platí následující soubor pravidel [6]:

1. Všechny netopýry využívají echolokaci ke vnímaní vzdálenosti a také „znají“ rozdíl mezi bariérami a kořisti.
2. Každý netopýr z populace litá náhodně s rychlostí $\boldsymbol{v}_j$, aktuální polohou $\boldsymbol{x}_j$, vysílá signály a hledá kořisti. Netopýří signál má zadanou konstantní frekvencí f_{min}, proměnnou vlnovou délkou λ a počáteční hlasitost A_0. Netopýry mohou automaticky nastavovat vlnovou délku svých vysílaných pulzů a upravovat rychlost emise pulzů $r \in (0, 1]$, v závislosti na blízkosti svého cíle.

3. I když se hlasitost může měnit mnoha způsoby, předpokládáme, že se hlasitost nastavuje na intervalu od počáteční hodnoty A_0 $(kde\ A_0 > 0)$ do minimální konstantní hodnoty A_{min}.

Na základě těchto idealizovaných pravidel lze popsat průběh netopýřího algoritmu pomocí pseudokódu, který je znázorněn na obrázku 5.1.

```
Definice účelové funkce F_cost
Inicializace populace virtuálních netopýrů x_j, v_j
Nastaveni počátečních parametrů f_j, A_j, r_j pro x_j

while (ukončovací podmínka algoritmu)
  vytváření nových řešeni nastavením frekvencí
  a aktualizací rychlosti a polohy
  if (rand > r_j)
    vyber nejlepšího řešení
    vytváření nového lokálního řešeni v okolí nejlepšího
  end if
  if ((rand < A_j) and (F_cost(x_j) < F_cost(x_best)))
    uplatnění nových řešení
    aktualizace hodnot A_j a r_j
  end if
  vyhodnocení nových řešení a hledaní nejlepšího jedince
end
```

Obrázek 5.1 - Pseudokód průběhu netopýřího algoritmu (Převzato z [8])

5.4 Inicializace populace netopýrů

Princip práce algoritmu s populacemi jedinců, který je popsán v kapitole 4 – Diferenciální evoluce, platí i pro netopýří algoritmus. Populace netopýrů P_x se skladá z vektorů definovaných paramterů $\boldsymbol{x}_j$:

$$P_x = (\boldsymbol{x}_j),\ j = 0{,}1 \dots N_{specimen} - 1$$
$$\boldsymbol{x}_j = (x_{i,j}),\ i = 0{,}1 \dots N_p - 1 \tag{5.1}$$

Před inicializací první populace musí být zadány horní a dolní hranice pro každý parametr. Jakmile byly stanoveny hranice inicializace, generátor náhodných čísel přiřadí každému parametru každého vektoru hodnotu z předepsaného rozsahu. Například počáteční hodnota i-tého parametru j-tého jedince z populace může být stanovena podle následujícího vzorce:

$$x_{j,i} = \text{rand}_i(0,1) \cdot \left(b_{i,H} - b_{i,D}\right) + b_{i,D} \tag{5.2}$$

kde $b_{j,H}$ je horní hranice pro vybraný parametr a $b_{j,D}$ je pak dolní hranice.

Na rozdíl od evolučních algoritmu, kterým je například algoritmus diferenciální evoluce, BA nevyžaduje vytvoření přechodové populace šumových vektoru. Proto další etapou netopýřího algoritmu je generování zkušební populace nových možných řešení optimalizačního problému:

$$\begin{gathered} P_{\boldsymbol{u}} = \left(\boldsymbol{u}_j\right),\ j = 0{,}1 \ldots N_{specimen} - 1 \\ \boldsymbol{u}_j = \left(u_{i,j}\right),\ i = 0{,}1 \ldots N_p - 1 \end{gathered} \tag{5.3}$$

kde $P_{\boldsymbol{u}}$ je zkušební populace která se skládá ze zkušebních jedinců (vektorů) $\boldsymbol{u}_j$. Pro sestavení zkušebních vektorů u BA se používá princip pohybu virtuálních netopýrů.

5.5 Pohyb virtuálních netopýrů

V předchozím textu diplomové práce (viz oddíl 5.3) jsou popsaná pravidla, která stanovují chovaní virtuálních netopýrů a jejich vlastností. Z těchto pravidel je také zřejmě, že nejdůležitějšími procesy činnosti BA jsou pochyb netopýrů a jejich schopnost k echolokaci. Proto dalším důležitým bodem je popis pravidel pro změnu polohy netopýrů a nastavení parametrů vysílaného signálu. Proces vytváření nových řešení (polohy netopýrů) se začíná z vypočtu frekvence pro každého jedince z populace [7]:

$$f_j = f_{min} + (f_{max} - f_{min}) \cdot \beta \tag{5.4}$$

kde f_{min} je definovaná dolní hranice pro hodnoty frekvence, f_{max} je naopak horní hranice. Součinitel β je náhodně vygenerované číslo z intervalu [0, 1].

V momentě, kdy je stanovená frekvence vybraného jedince, je třeba stanovit rychlost, se kterou netopýr poletí do nové polohy. Jinými slovy – jak moc se změní parametry j-tého vektoru [7]:

$$\boldsymbol{v}_j{}^k = \boldsymbol{v}_j{}^{k-1} + (\boldsymbol{x}_j{}^{k-1} - \boldsymbol{x}_{best}) \cdot f_j \tag{5.5}$$

kde $v_j{}^{k-1}$ je hodnota rychlostí netopýru v předchozí $(k-1)$ časový okamžik[2], $v_j{}^k$ – je rychlost jedince v následující (k) časový okamžik, $x_j{}^k$ – poloha jedince a x_{best} je aktuální nejlepší poloha (řešení).

[2] Index k, který representuje určitý časový okamžik, je analogem indexu g, který označuje určitou generace jedinců v algoritmu DE (viz kapitola 4). Oba indexy mají stejný smysl – rozlišit různé etapy hledaní optimálního řešení. V případě BA jde o změnu polohy netopýrů s časem, a v případě DE se jedná o změnu genů jedinců od generace ke generaci.

Vzorec (5.6) popisuje, jak vypočítat zkušební polohu j-tého netopýru ($\boldsymbol{u}_j$) na základě aktuální polohy $\boldsymbol{x}_j{}^{k-1}$ a stanovené rychlostí $\boldsymbol{v}_j{}^k$ [7].

$$\boldsymbol{u}_j = \boldsymbol{x}_j{}^{k-1} + \boldsymbol{v}_j{}^k \tag{5.6}$$

Ze vzorků (5.5) a (5.6) je patrno, že v rovnicích se sčítají a násobí se hodnoty rychlostí, polohy, frekvence a další náhodné koeficienty. Proto je třeba upozornit, že tyto terminy neodpovídají reálným fyzickým významům a jsou použité pro jednodušší pochopeni principu činnosti algoritmu a znázornění toho, že algoritmus je inspirován chováním reálných netopýrů.

BA předpokládá další způsob posuvu netopýrů pro nalezení optima – náhodná procházka. Ale tento způsob se používá pouze pro lokální přemístění netopýrů na kratší vzdálenosti okolo dočasného nejlepšího řešení. Následující vzorec (5.7) popisuje náhodné generování nových lokálních řešení $\boldsymbol{x}_{local}$:

$$\boldsymbol{x}_{local} = \boldsymbol{x}_{best} + \varepsilon A^k \tag{5.7}$$

kde $\varepsilon \in [-1,\ 1]$ – náhodné číslo, $A^k =< A_j^k >$ - je průměrná hodnota hlasitostí všech jedinců z populace v časovém okamžiku k [7]. O tom, jestli se do zkušební populace dostane takto vytvořený nový vektor, rozhoduje hodnota rychlosti emise pulzů:

$$\boldsymbol{u}_j = \begin{cases} \boldsymbol{x}_{local} & if\ (rand_i > Cr) \\ \boldsymbol{u}_j \end{cases} \tag{5.8}$$

5.6 Hlasitost a rychlost emise pulzů

Kromě polohy a rychlosti virtuálních netopýrů, musí během iteračního procesu být odpovídajícím způsobem aktualizované i parametry vysílaného signálu – hlasitost A_j a rychlost pulzní emise r_j. Jakmile netopýr najde kořist, hlasitost obvykle klesá, zatímco rychlost pulsní emise se zvyšuje, proto lze hlasitost zvolit jako libovolnou hodnotu, která bude nejvíc odpovídat účelům algoritmu [7]. Například extrémní hodnoty hlasitostí mohou být definovaná jako $A_0 = 100$ a $A_{min} = 1$, nebo pro zjednodušení $A_0 = 1$ a $A_{min} = 0$. Ale je důležité, aby v momentě, kdy netopýr dosáhne svého cílů, hlasitost jeho signálu byla A_{min}. Pravidla, podle kterých jsou aktualizovaná hlasitost a rychlost pulzní emise jsou vyjádřená ve vzorcích (5.9) [7]:

$$\begin{aligned} A_j^k &= \alpha A_j^{k-1}, \alpha \in (0,1) \\ r_j^k &= r_j^0(1 - e^{-\gamma k}), \gamma > 0 \end{aligned} \tag{5.9}$$

kde α a γ jsou součinitele. Pro libovolné hodnoty z přislušných intervalů platí:

$$A_j^k \to 0, r_j^k \to r_j^0, pro\ k \to \infty \tag{5.10}$$

V případě jednoduchosti můžeme použít $\alpha = \gamma$, ale každem konkretním připadě výběr parametrů vyžaduje experimentování. Navic, zpočátku by měl mít každý netopýr různé hodnoty hlasitosti a frekvence vyzařování, což lze dosáhnout přiřazením náhodných veličin z přislušných intervalů.

5.7 Validace nových řešení

Jakmile jsou přemístěny všechny netopýry z populace, buď pomocí aktualizace jejich rychlosti, nebo pomocí náhodného lokálního přesunutí, každá nová řešení se porovnává s původní polohou příslušného netopýru. Hlavním kritériem pro vyber jednoho z těchto vektorů je jejich hodnota účelové funkce. Pokud zkušební poloha netopýru $\boldsymbol{u}_j$ má stejnou nebo nižší hodnotu účelové funkce než hodnota jeho původního vektoru $\boldsymbol{x}_j$, nahradí původní vektor v další iteraci – jinými slovy, netopýr se přesune na nové místo:

$$\boldsymbol{x}_j = \begin{cases} \boldsymbol{u}_j \ \ if\ \left(F_{cost}(\boldsymbol{u}_j) \leq F_{cost}(\boldsymbol{x}_j)\right) \\ \boldsymbol{x}_j \end{cases} \tag{5.11}$$

V momentě, kdy jsou porovnaný všechny zkušební netopýry, proces hledaní nových poloh netopýrů se opakuje znovu, dokud se nenajde optimální řešení, nebo dokud není splněna předem stanovená podmínka ukončení algoritmu (např. počet iterací dosáhne předem definovaného maxima).

Kapitola 6

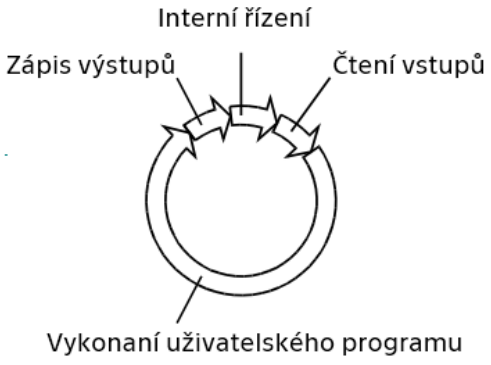
Programovatelné automaty TECOMAT FOXTROT

6.1 Úvod

Programovatelný automat – číslicový elektronický řídící systém určeny pro řízení procesů a pracovních strojů v průmyslovém prostředí [8]. Celosvětové se používá výraz PLC (z anglického *Programmable Logic Controller*). Hlavní funkce PLC spočívá v získávaní a předávaní informace z a do řízeného zařízení prostřednictvím číslicových nebo analogových vstupů a výstupů. Algoritmy řízení jsou uloženy v paměti uživatelského programu, který je cyklicky vykonáván [8].

6.2 Princip vykonávání programu PLC

Řídicí algoritmus programovatelného automatu představuje sekvenci instrukcí, která je uložena v paměti uživatelského programu. Během vykonaní algoritmu, PLC postupně čte jednotlivé instrukce a provádí potřebné operace s daty. Jakmile jsou provedeny veškeré instrukce definovaného algoritmu, programovatelný automat začíná aktualizovat hodnoty na výstupních periferních modulech a přenášet stavy ze vstupních modulech do zásobníkové paměti PLC. Výše uvedené akce se stále opakuji a tím vytvářejí tak zvány cyklus programu. Ve většině literatury, která popisuje princip činnosti programovatelných automatů, pro vysvětleni cyklu programu lze najit následující obrázek:



Obrázek 6.1 – Cyklus programu PLC (Převzato z [8])

Velmi důležitá vlastnost cyklu programu je, že jak aktualizace výstupních hodnot, tak i vstupních stavu se provádí pouze jednou. To znamená, že celý řídicí algoritmus provádí veškeré instrukce pouze s těmi stavy vstupů, které byly přečteny na začátku

programového cyklu. A nastavovaní výstupních hodnot se uskuteční pouze v době, kdy je zaručeno, že tyto hodnoty nebudou dále změněny, tzn po dokončeni všech výpočtů – na konci cyklu programu. Tím se odstraňuje možnost vzniku poruch a nebezpečných stavů při změnách vstupních hodnot během výpočtů.

6.3 Programovaní podle normy IEC 61 131-3

Norma IEC 61 131 pro programovatelné řídící systémy má pět základních částí a představuje souhrn požadavků na moderní řídící systémy [9]. V evropské unii je tato norma přijatá pod číslem EN IEC 61 131. Z hlediska programovacích jazyků pro PLC je nejdůležitější třetí část normy – IEC 61 131-3. Tato norma představuje první vážný pokus o standardizaci programovacích jazyků pro průmyslovou automatizaci [9]. Také specifikuje syntaxe a sémantiky jednotné sady programovacích jazyků, včetně celkového softwarového modelu a struktury jazyka.

6.3.1 Jazyky programovaní PLC

V rámci standardu jsou definovány čtyři programovací jazyky [9]. Podrobný popis detailů každého programovacího jazyku PLC není cílem dané diplomové práce. Proto v dalším textu jsou pouze vyjmenované programovací jazyky uvedené v normě. Také pro základní přehled ke každému druhu jazyku je přiřazen jednoduchý příklad se stejnou funkcionalitou – ukládaní do proměnné **C** výsledku logického součtu proměnné **A** a negované proměnné **B**.

Textové jazyky

- **IL** (anglický *Instruction List*) – jazyk seznamu instrukcí

```
LD    A
ANDN  B
ST    C
```

Obrázek 6.2 - příklad programu v jazyce IL (Převzato z [9])

Programovací jazyk seznamu instrukcí je nízkoúrovňový jazyk typu assembler. Tento jazyk patří mezi řádkově orientované jazyky programovaní [9].

- **ST** (anglický *Structured Text*) – jazyk strukturovaného textu

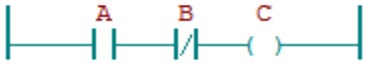
```
C := A AND NOT B;
```

Obrázek 6.3 - příklad programu v jazyce ST (Převzato z [9])

Jazyk strukturovaného textu je velmi výkonný programovací jazyk, který byl vyvinut na základě známého jazyku Pascal. Je také velice podobný programovacímu jazyku C. Je objektově orientován a obsahuje všechny podstatné prvky moderního programovacího jazyka [9].

Grafické jazyky

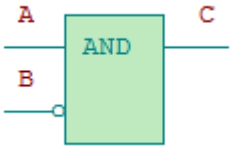
- **LD** (anglický *Ladder Diagram*) – jazyk příčkového diagramu (jazyk kontaktních schémat)



Obrázek 6.4 - příklad programu v jazyce LD (Převzato z [9])

Programovací jazyk příčkového diagramu pochází z elektromechanických reléových obvodů je založen na grafické reprezentaci reléové logiky [9]. Jazyk kontaktních schémat je původně určen ke zpracovaní logických (booleovských) signálů. Proto se často uplatňuje v jednodušších řídicích systémech.

- **FBD** (anglický *Function Block Diagram*) – jazyk funkčního blokového schématu



Obrázek 6.5 - příklad programu v jazyce FBD (Převzato z [9])

Jazyk funkčního blokového schématu je dalším grafickým programovacím jazykem. Programování pomocí FBD je založeno na propojování souboru funkcí a funkčních bloků, které jsou reprezentovaný obdélníky (viz Obrázek 6.5).

6.3.2 Volba vhodného programovacího jazyku

Pravdou je, že jakýkoliv požadovaný program lze naprogramovat pomocí libovolného jazyku, popsaného v normě IEC 61 131-3, se zachováním stejné funkcionality. Ale čas potřebný pro realizaci uživatelského programu a množství kódu (v případě grafických jazyků – velikost schémat a diagramů) jsou závislé na složitosti požadované funkcionality.

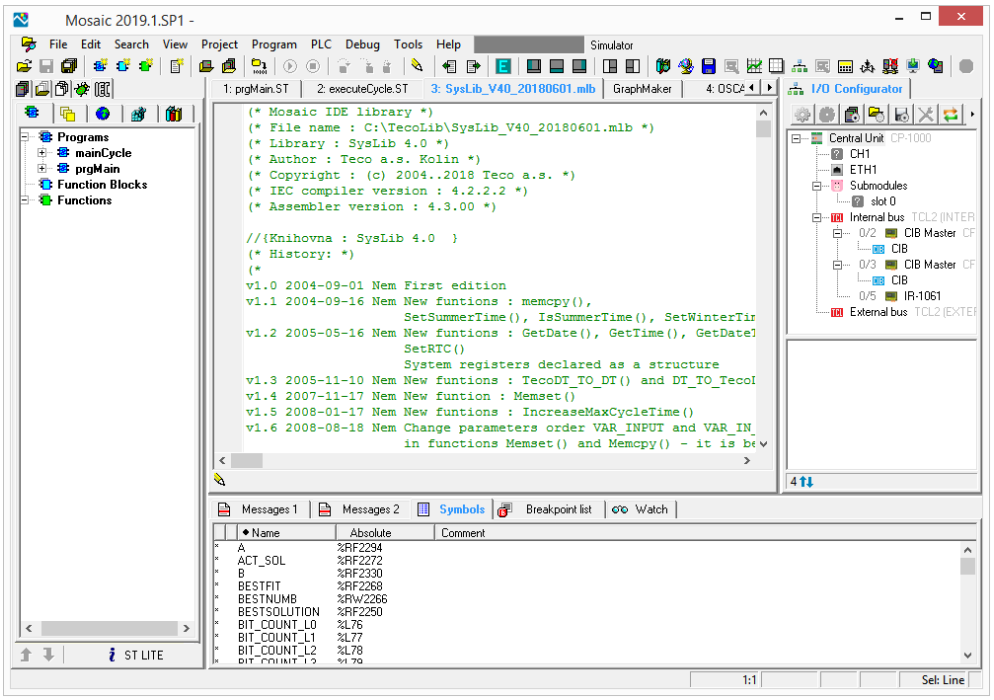
Dalším rozhodujícím faktorem pro výběr programovacího jazyku je vlastní zkušenost programátora. Pravděpodobně, vývojář, který má odborné znalosti a zkušenosti s programováním v assembleru, pro vývoj řídicího systému použije jazyk IL.

Člověk, který se dobře orientuje v oblasti schémat a diagramu, může využit jeden z grafických programovacích jazyků (FBD nebo LD).

Pro návrh řídicího systému na základě metody prediktivního řízení vybral jsem textový programovací jazyk ST. Hlavním důvodem bylo to, že ST je relativně podobny programovacím jazykům MATLAB a C, se kterými mám zkušeností z vysoké školy a z praxe. Během psaní kódu jsem se snažil využit jednu z nejdůležitějších vlastnosti textových jazyků – možnost napsaní dobře čitelného kódu. Vhodné názvy proměnných a funkce, podle kterých lze jednoduše pochopit jejich význam a funkcionalitu, pomáhají při zpětně kontrole již napsaného kódu. Také tento postup je vhodný v případě, kdy je potřeba se vracet k určité částí programu po několika týdnech – smysluplné názvy, doplněné stručnými komentáři, rychle připomenou, o co se jedná v této částí kódu a proč jsou nastavené určité hodnoty.

6.4 Vývojové prostředí Mosaic

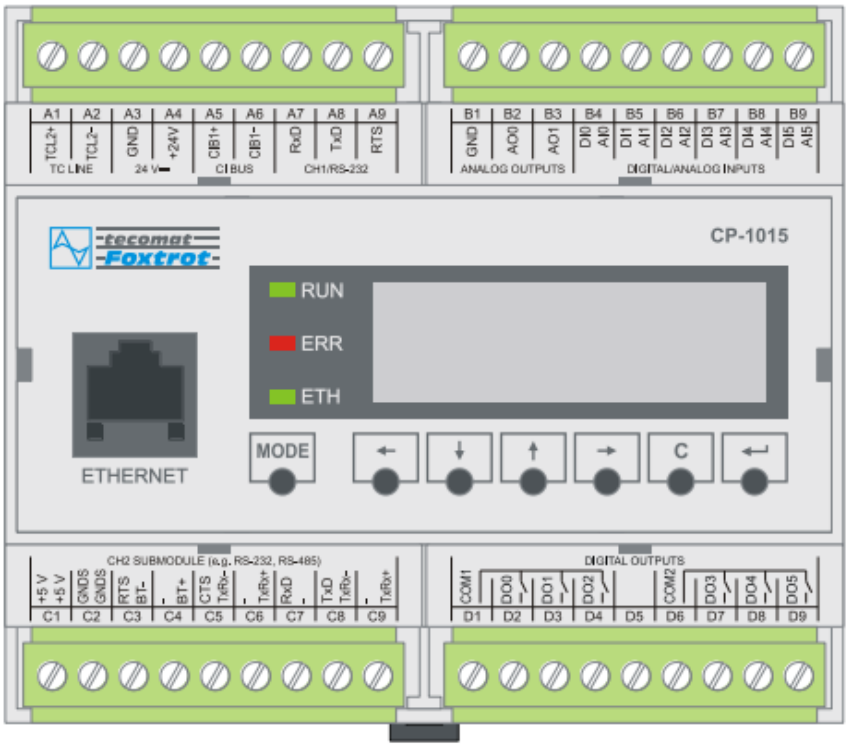
Existuje celá řada vývojových prostředí pro programovaní a práce s PLC. Vyber prostředí hlavně zaleží na typu programovatelného automatu a jeho výrobci. Podle [10] – Mosaic je vývojové prostředí pro tvorbu a ladění programů pro programovatelné logické systémy TECOMAT z produkce firmy Teco a.s. Program Mosaic je dodáván od roku 2000. Prostředí je vyvíjeno ve shodě s mezinárodní normou IEC EN-61131-3, která definuje strukturu programů a programovací jazyky pro PLC.



Obrázek 6.6 - Ukázka vývojového prostředí Mosaic firmy Teco a.s.

6.5 Systém Tecomat Foxtrot CP-1015

Systém Tecomat Foxtrot je kompaktní řídicí a regulační systém s možností modulárního rozšíření. Je určen pro řízení technologií v nejrůznějších oblastech průmyslu i v jiných odvětvích [9].



Obrázek 6.7 - Základní modul CP-1015 (Převzato z [9])

Pro operace se výstupními signály CP-1015 je vybaven šestí reléovými a dvěma analogovými výstupy. Pro vstupní signály lze použit šest analogových vstupů, které lze nakonfigurovat jako vstupy digitální. Při připojení dalších modulů je možné počet vstupů a výstupů výrazně navýšit. Podle dokumentace doba cyklu PLC je 0,2 ms na 1000 logických operace.

Kapitola 7

Návrh řídicího algoritmu

7.1 Názvosloví

Úvodem ke kapitole 7 je celá předchozí část této diplomové práce. V předchozích kapitolách jsou popsané různé metody, techniky a zařízení. A každá metoda je popsána pomocí různých termínů. Některé z těchto terminů mají jasný fyzický význam, některé naopak smysl mají pouze formální. Dokonce, některá stejná slova v různých aplikacích mají opačný smysl. Příkladem jsou výstupy dvou soustav – v případě regulované soustavy se jedná o řízenou veličinu, v případě řidiči soustavy jde o řídicí (akční) veličinu. Proto úvodem této částí diplomové práce je popis terminů, které se v dalším textu hodně používají.

Společné prvky

Regulovaná soustava – je zařízení nebo technologický proces, na kterém se provádí regulace a v němž se ovlivňuje regulovaná veličina.

Regulovaná veličina – proměnná, která zachová informace o aktuálním stavu výstupu řízeného procesu. Regulovaná veličina je výstupní hodnotou regulované soustavy a vstupní hodnotou řídícího systému.

Akční zásah – je proměnná ovlivňující průběh regulované velečiny. Akční zásah (nebo také akční veličina) je vstupem pro regulovanou soustavu a výstupem pro řídící systém.

Řídicí systém – je systém který řídí definovanou regulovanou soustavu (technologický proces). Návrh celého řídícího systému je hlavním cílem diplomové práce. Řídicí systém se skládá ze dvou nejdůležitějších částí. První částí je řídící program (řídicí algoritmus), ve kterém jsou implementovány algoritmy globální optimalizace a metoda prediktivního řízení. Druhou částí je hardware, na kterém řídící program běží. Hardwarem pro navrženy řídící systém je PLC Tecomat Foxtrot.

Prediktivní řízení

Model procesu – matematický model, popisující dynamické chovaní regulované soustavy.

Horizont predikce – určuje počet časových okamžiků pro které bude predikováno chovaní řízené soustavy. Zjednodušeně lze říct, že horizont predikce stanovuje, jak daleko řídicí algoritmus bude schopen pozorovat do budoucna.

Optimalizační problém. Pro navržený řídicí systém optimalizační problém spočívá ve výběru optimální hodnoty akční veličiny pro každý časový okamžik celého horizontu predikce.

Optimalizátor – je nástroj pro řešení optimalizačního problému. Optimalizátory navrženého řídicího systému jsou metaheuristické algoritmy popsané v předchozích částech diplomové práce.

Algoritmy globální optimalizace:

Jedinec – formální název pro vektor parametrů. Jedinec reprezentuje možné řešení optimalizačního problému a každý jeho prvek je hodnotou příslušného parametru řešeného problému. Souborem parametrů, v případě návrhu systému řízení, je posloupnost přírůstků akčních zásahů, které budou přičteny k celkové hodnotě akčního zásahu u v příslušný časový okamžik. Délka vektoru je rovna definované délce horizontu predikce. Společný název *jedinec* platí jak pro vektory z algoritmu diferenciální evoluce, tak i pro vektory netopýřího algoritmu, ve kterém představují polohu virtuálních netopýrů.

Populace – je množina jedinců. Soubor všech dostupných možných řešení optimalizačního problému, mezi kterými se hledá optimální. Velikost populace je omezena konfigurovatelným počtem jedinců $N_{specimen}$.

Iterace. Nová možná řešení pomocí DE algoritmu vznikají tvorbou nových generací a v případě BA – změnou polohy netopýrů v čase. Pro oba procesy v dalším textu se používá termín iterace.

7.2 Struktura řídícího algoritmu

V předchozí kapitole jsem popisoval obecně vlastnosti a princip činnosti programovatelných automatů firmy TECO. Společným rysem všech PLC je cyklické vykonávaní uživatelského programu. Tento uživatelský program se skládá z několika typů uživatelských procesů. Ve výchozím nastaveni celý program může probíhat v základním procesu P0. Pravda je, že vykonaní stejného programu, který se spouští cyklicky, v různých cyklech PLC muže trvat kratší nebo delší dobu. To je ovlivněno složitostí programu, množstvím použité uživatelské pamětí PLC a velikosti vstupních a výstupních hodnot.

Při regulaci dynamických procesů tento jev bude mít negativní vliv na kvalitu řízení. Proto bylo třeba se tomuto jevu vyhnout během návrhu řídicího algoritmu. Lze k tomu využit uživatelské procesy ze skupiny procesů P10 až P40, které se spouštějí přesně v uživatelem definovaný časový okamžik. Proto vykonaní celého algoritmu prediktivního řízení bylo rozděleno do dvou částí.

7.2.1 Hlavní proces P0

Proces P0 je základním procesem PLC Tecomat Foxtrot. Tento proces musí být definován i když uživatelsky program je prázdný (nemá žádnou funkcionalitu).

První důležitá funkcionalita procesu P0 pro navržený algoritmus je definice všech potřebných polí, konstant a proměnných. Příkladem jsou: délka horizontu predikce, počet jedinců v populaci, požadovaná hodnota, různé identifikátory a mnoho dalších. Také jsou definované proměnné pro fyzické vstupy a výstupy PLC. Čtení a zápis jejích hodnot je také realizován v procesu P0. Velmi důležitě v hlavním procesu definovat matice řízeného modelu, které jsou popsány v kapitole 2 (viz strana 3).

Druhá důležitá funkcionalita spočívá ve voláni uživatelského procesu P10. Pro ovládaní procesem P10 byly využity systémové registry S. Vykonaní uživatelského procesu začíná, jakmile je nastaven bit 1 z bajtu S25 na logickou jedničku.

7.2.2 Uživatelský proces P10

Uvnitř uživatelského procesu P10 probíhají nejdůležitější výpočty celého systému řízení. V tomto procesu jsou realizovaný algoritmy prediktivního řízení, využívající buď algoritmus diferenciální evoluce, nebo netopýří algoritmus.

Proces P10 se volá nastavováním bitu na adrese S25.1. Příklad realizace startu procesu P10 je zobrazen na následujícím obrázku:

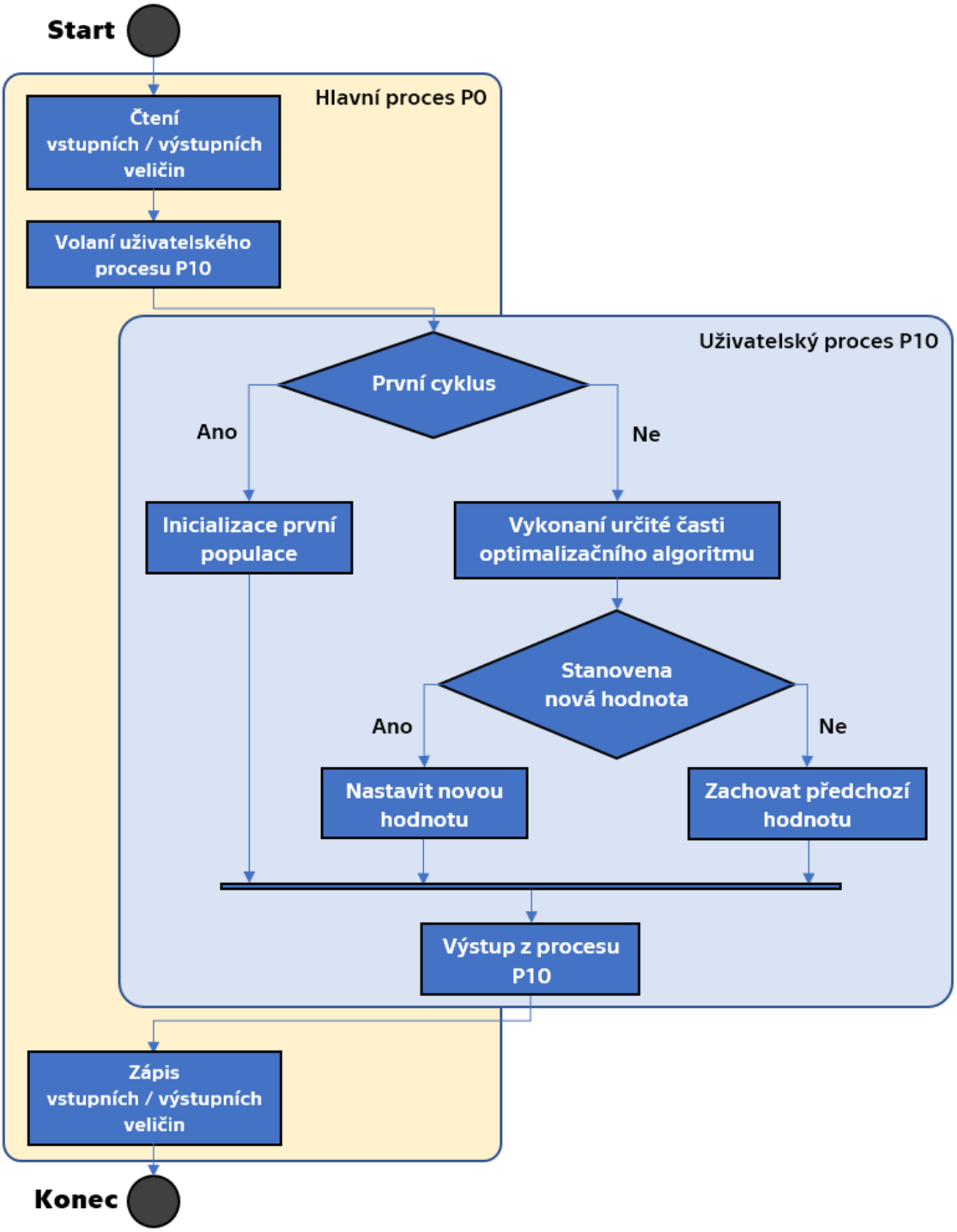
```
// Flag to call P10 process
executeP10  AT %S25.1 : BOOL;
// Call P10 process every 100 ms
executeP10 := System_S.R_EDGE_100MS;
```

Obrázek 7.1 - Voláni PLC procesu P10

Z obrázku 7.1 je patrno, že pro start procesu P10 byly použité další systémové bity. PLC Tecomat foxtrot má několik typu čítačů, které nohou být využité pro zapouštění procesu P10. Pro práci s časovači byla využita Mosaic knihovna od firmy Teco a.s. SysLib.mlb.

7.2.3 Diagram aktivit řídicího algoritmu

Pro znázornění principu vykonaní algoritmu ve dvou různých procesech byl vytvořen diagram aktivit jednoho PLC cyklu:



Obrázek 7.2 - Diagram aktivit jednoho cyklu řídicího algoritmu

7.3 Společná část řídicího algoritmu

Během návrhu algoritmů BA a DE jsem se snažil vytvořit společnou část programu, která se bude stejně vykonávat nezávisle na typu optimalizačního algoritmu. Tento přístup může pomoct při přepínaní mezi programy a algoritmy a také během implementace dalších optimalizačních algoritmů v budoucnu.

Příkladem společné částí programu je definice matic řízeného modelu:

```
//-----------------------------------------------------------------
// Definition of the process model:
// State matrix A
A       : ARRAY[0..,0..2]  OF REAL := [1.726,  -0.737,   0.0,
                                       1.0,     0.0,     0.0,
                                       0.07523, 0.06796, 1.0];
// Input matrix B
B       : ARRAY[0..2]  OF REAL := [0.125,
                                   0.0,
                                   0.0094];
// Output matrix C
C       : ARRAY[0..2]  OF REAL := [0,0,1];
//-----------------------------------------------------------------
```

Obrázek 7.3 - Definice matic řízeného modelu

Další částí programu, která je společná pro oba algoritmy, je inicializace první populace jedinců. Inicializace se provádí pouze v prvním cyklu běhu programu. Pro kontrolu prvního cyklu PLC byla použitá systémová proměnná *CYCLE_COUNTER* z knihovny SysLib.mlb. Ale bylo potřeba také zabezpečit přetečení čítače (znovu se nastaví na nulu) pomocí proměnné *firstCycle*, která uchová informaci, zda již proběhl první cyklus nebo ne. Inicializační hodnota proměnné je *TRUE* a jakmile se inicializuje první populace jedinců, nastaví se hodnota *FALSE*, která pak se nikdy nemění.

Během inicializace každý jedinec z populace se naplní inicializačními hodnotami podle vztahu (4.2) (resp. (5.2)):

```
// Fill all solutions with random values
Sol[i,j] := INT_TO_REAL(RDM2(last := (i * j),
                             low := REAL_TO_INT(lowerLim),
                             high := REAL_TO_INT(upperLim)));
```

Obrázek 7.4 –Inicializace první populace

Zároveň pro každého jedince se počítá hodnota účelové funkce. Přiklad její realizaci je uveden v dalším textu. Ale po inicializačním procesu akční veličina zůstává v počátečním stavu, protože nastavování náhodné inicializační hodnoty z nejlepšího jedince může přivést k neočekávanému chovaní soustavy na začátku řízení.

7.4 Realizace algoritmu diferenciální evoluce

Specifické konstanty a proměnné

Kromě společných polí, proměnných a konstant, které se používají pro oba DE a BA algoritmy, před spouštěním výpočtů je nutně definovat specifické proměnné pro algoritmus diferenciální evoluce.

```
//-----------------------------------------------------------------------
// Special variables for Differential Evultion
VAR
    // Array of actual intermediary population of mutant vectors
    Mutant      : ARRAY [1..Npred,1..pop_size] OF REAL;
    // Array of fitness function values for each mutant vector
    mutantFit   : ARRAY [1..pop_size] OF REAL;
END_VAR
//-----------------------------------------------------------------------
VAR CONSTANT
    // Contants for the Differential Evolution
    // Scale factor
    F           : REAL := 1.6;
    // Crossover probability
    crossover   : REAL := 0.3;
END_VAR
//-----------------------------------------------------------------------
```

Obrázek 7.5 - Specifické konstanty a proměnné pro DE algoritmus

Na obrázku 7.5 jsou zobrazené pouze nejdůležitější proměnné pro DE algoritmus. Dočasné proměnný, které se používají pro uchování náhodně vygenerované hodnoty nebo různých indexů, nejsou zobrazený na předchozím obrázku.

Inicializace populace

První populace je tvořená stejným způsobem jak pro netopýří algoritmus, tak i pro algoritmus diferenciální evoluce. Proto inicializace je popsána pouze v textu pro společnou část programu.

Stanovení nových řešení

Programový cyklus PLC je časově omezen a celý výpočet nových jedinců je delší než maximální povolený čas jednoho programového cyklu. Proto proces aktualizace jedinců byl rozdělen do čtyř kroků pomocí příkazu *CASE*. Každý krok se spouští ve zvláštním programovém cyklu.

1. **Mutace jedinců**

   V prvním kroku je realizován proces mutace každého jedinců z populace podle vzorce (4.4). Pro každého jedince z populace je potřeba vybrat další tři náhodné jedince. V případě programování PLC tento proces je komplikovanější, protože pro

generovaní odlišných náhodných čísel během jednoho programovacího cyklu je nutně volat funkci *RDM2* s různými hodnotami vstupního parametru *last*:

```
// Compute 3 defferent random numbers
rndNumber0 := RDM2(last := 1 * j,
                   low := 1,
                   high := pop_size - 1);
// If the computed one is equal to the actual, take the nearest
IF (rndNumber0 = j) AND (rndNumber0 = REAL_TO_INT(upperLim)) THEN
    rndNumber0 := rndNumber0 - 1;
ELSIF (rndNumber0 = j) THEN
    rndNumber0 := rndNumber0 + 1;
END_IF;
```

Obrázek 7.6 - vyber náhodného jedince z populace

Navíc nesmí se náhodně vybraný jedinec rovnat aktuálnímu jedinci. Proto v případě, kdy náhodné číslo se rovna indexu aktuálního jedince, vybere se jiný nejbližší jedinec. Hraniční případy (první a poslední jedinec z populace) jsou zabezpečené. V momentě, kdy jsou vybraný tři jedinci z populace, lze stanovit šumový vektor:

```
// Loop over whole prediction horizon
FOR i := 1 TO Npred DO
    // Create a mutant vektror for the actual one
    Mutant[i, j] := Sol[i, rndNumber0] +
                    F * (Sol[i, rndNumber1] - Sol[i, rndNumber2]);
END_FOR; // Loop over whole prediction horizon
```

Obrázek 7.7 - výpočet šumového vektoru

2. Křížení šumových vektorů

Dalším krokem algoritmu diferenciální evoluce je proces křížení. Důvod, proč je tento proces realizován, je popsán v kapitole 4. Křížení pro každý parametr každého šumového vektoru se provádí podle vzorce (4.6). Příklad programové realizace pro jeden šumový vektor je znázorněn na následujícím obrázku:

```
// Loop over whole prediction horizon
FOR i := 1 TO Npred DO
    rndCrossover := RDM(INT_TO_REAL(i * j));
    IF (FALSE = ((rndCrossover < crossover) OR (rndParameter = j))) THEN
        Mutant[i, j] := Sol[i, j];
    END_IF;
    // Replace an invalid solution with random values
    IF Mutant[i, j] < lowerLim OR Mutant[i, j] > upperLim THEN
        // Replace an invalid solution with random values
        Mutant[i,j] := INT_TO_REAL(RDM2(last := (i * j),
                                        low := REAL_TO_INT(lowerLim),
                                        high := REAL_TO_INT(upperLim)));
    END_IF;
END_FOR; // Loop over whole prediction horizon
```

Obrázek 7.8 - Proces křížení v algoritmu DE

Po dokončení procesu křížení vzniká nova populace zkušebních jedinců.

3. **Aktualizace účelové funkce pro aktuální jedince**

   Po realizaci předchozí hodnoty akčního zásahu je známá nová aktuální hodnota regulované veličiny. Proto hodnoty účelové funkce jedinců z aktuální populace již nejsou platné. Přepočet hodnot účelové funkce je znázorněn na obrázku 7.9.

```
// Loop over whole prediction horizon
FOR i := 1 TO Npred DO
    deltaU := Ko * Uo[REAL_TO_INT(Sol[i, j])];
    // Calculate one step of the model
    FOR n := 0 TO 2 DO
        FOR m := 0 TO 2 DO
            X[n] := X[n] + A[n, m] * X[n];
        END_FOR;
        X[n] := X[n] + B[n] * deltaU;
    END_FOR;
    FOR n := 0 TO 2 DO
        y := y + C[n] * X[n];
    END_FOR;
    // Calculate a standard deviation
    Fit[j] := Fit[j] + (W - y) ** 2;
END_FOR; // Loop over whole prediction horizon
```

Obrázek 7.9 - Výpočet účelové funkce

4. **Výpočet účelové funkce pro zkušební jedinci**

   Dalším krokem algoritmu je výpočet hodnoty účelové funkce pro každého jedince. Výpočet predikované trajektorii každého jedince pro celý horizont predikce N_p trvá relativně dlouho, proto tento proces je vyřazen do zvláštního kroku. Realizace výpočtů je stejná jako i v předchozím kroku.

5. **Výběr nejlepšího jedince**

   Posledním krokem algoritmu diferenciální evoluce je výběr nejlepšího jedince z nové populace. Celý nejlepší jedinec se uloží do proměnný *bestSolution*, ale na vystup z PLC se nastaví pouze první parametr z tohoto jedince.

Pátý krok je posledním krokem algoritmu diferenciální evoluce. Po jeho provedení se proměnná *step* nastaví na hodnotu 1 a tím se uzavře celý výpočetní cyklus algoritmu.

7.5 Realizace netopýřího algoritmu

Specifické konstanty a proměnné

Prvním krokem při návrhu algoritmu je zavedení specifických pro BA proměnných a konstant. Tyto definované proměnné a konstanty jsou znázorněny na následujícím

obrázku 7.10. Dočasné proměnné, které byly použité například jako indexy v „*for*" cyklech, nejsou zobrazeny na tomto obrázku.

```
//-------------------------------------------------------------------
// Special variables for Bat Algorithm
VAR
    // Array to store actual "frequency" for every bat
    Freq        : ARRAY [1..pop_size] OF REAL;
    // Array to store actual "speed" for every bat
    Vel         : ARRAY [1..Npred,1..pop_size] OF REAL;
    // Array of loudness for every bat
    Loudness    : ARRAY [1..pop_size] OF REAL;
    // Array of pusle rates for every bat
    PulseRate   : ARRAY [1..pop_size] OF REAL;
END_VAR
//-------------------------------------------------------------------
VAR CONSTANT
    // Contants for Bat Algorithm
    // "Frequency" bounds
    maxFreq   : REAl := 3;
    minFreq   : REAL := 0;
END_VAR
//-------------------------------------------------------------------
```

Obrázek 7.10 - Specifické konstanty a proměnné pro BA algoritmus

Inicializace populace

První populace je tvořená stejným způsobem jak pro netopýří algoritmus, tak i pro algoritmus diferenciální evoluce. Proto inicializace je popsána pouze v textu pro společnou část programu.

Stanovení nových řešení

Stejně jako i v případě algoritmu diferenciální evoluce, celý proces stanovení nových řešení optimalizačního problému je rozdělen na několik kroku. Na rozdíl od DE algoritmu, pro netopýří algoritmus stačí pouze čtyři kroky. Číslo aktuálního kroku je uloženo v proměnné *step* a na konci každého kroku se přepíná do dalšího. Procesy, který probíhají v příslušných krocích jsou popsaný v následujícím seznamu:

1. **Posuv netopýrů**

Nejdřív pro každého jedince z populace se aktualizuje frekvence podle vzorku (5.4):

```
//-----------------------------------------------------------
1:
    // BAT ALGORITHM - STEP 1
    // Loop over the whole population
    FOR j := 1 TO pop_size DO
        // Compute a new "frequency" value for every bat
        Freq[j] := minFreq + (maxFreq - minFreq) *
            RDM(last := INT_TO_REAL(j) * 0.01);
```

Obrázek 7.11 - BA – výpočet frekvence

Funkce *RDM* z knihovny *OSCAT_BasicLib* vrací náhodné reálné číslo z intervalu [0, 1]. Ale v případě, kdy je potřeba generovat několik náhodných čísel během jednoho cyklu PLC, musí se zadávat různé hodnoty vstupního argumentu *last*. Proto jako hodnota vstupního argumentu je zvoleno pořadové číslo jednice *j* násobené koeficientem 0,01 pro zabezpečení případu generovaní stejných čísel při dalším volaní funkce *RDM*.

Pak pro každého jedince z populace se aktualizuje příslušná hodnota rychlosti (podle (5.5)) a polohy (podle (5.6)):

```
// Loop over whole prediction horizon
FOR i := 1 TO Npred DO
    // Calculate a "speed" value for each element
    Vel[i,j] := Vel[i,j] + (Sol[i,j] - bestSolution[i]) * Freq[j];
    // Move every bat to a new position according to the calculated "speed"
    Sol[i,j] := Sol[i,j] + Vel[i,j];

    // If the corrent possible solution is out of range, it should be changed
    IF ((Sol[i, j] < lowerLim) OR (Sol[i, j] > upperLim)) THEN
        // Use a valid random index instead
        Sol[i, j] := INT_TO_REAL(RDM2(last := (j * i),
                                      low := REAL_TO_INT(lowerLim),
                                      high := REAL_TO_INT(upperLim)));
    END_IF;
END_FOR; // Loop over whole prediction horizon
```

Obrázek 7.12 - BA – aktualizace jedinců

Během aktualizace jedinců na základě výpočtu frekvence a rychlostí, může dojit k tomu, že stanovená hodnota parametru bude ležet mimo definovaný rozsah. V tomto případě bude vygenerovaná náhodná veličina z definovaného intervalu a zaujme místo nevhodné hodnoty parametru.

2. Generace lokálních řešení a zaokrouhlení parametrů

Dalším způsobem aktualizace jedinců je generovaní nových lokálních řešení okolo aktuálního nejlepšího řešení využitím hlasitostí netopýrů (podle (5.7)).

```
//--------------------------------------------------------------------
2:
    // Loop over the whole population
    FOR j := 1 TO pop_size DO
        // Check condition to generate a new local solution
        IF ((RDM(last := INT_TO_REAL(j) * 0.1)) < PulseRate[j]) THEN
            // Loop over whole prediction horizon
            FOR i := 1 TO Npred DO
                // Create a new local solution
                IF (bestSolution[i] < INT_TO_REAL(nOfChaneges)) THEN
                    Sol[i, j] := bestSolution[i] + Loudness[i];
                END_IF;
            END_FOR; // Loop over whole prediction horizon
        END_IF;
        // Loop over whole prediction horizon
        FOR i := 1 TO Npred DO
            Sol[i, j] := INT_TO_REAL(REAL_TO_INT(Sol[i, j]));
        END_FOR; // Loop over whole prediction horizon
    END_FOR; // Loop over the whole population
    // Move to the next step (3)
    Step := 3;
//--------------------------------------------------------------------
```

Obrázek 7.13 - BA – výpočetní krok 2

3. Výpočet účelové funkce

Výsledkem realizace prvních dvou kroku je to, že je aktualizován každý jedinec z populace. Proto v dalším kroku lze stanovit hodnoty účelové funkce pro každého jedince.

```
// Loop over whole prediction horizon
FOR i := 1 TO Npred DO
    deltaU := Ko * Uo[REAL_TO_INT(Sol[i, j])];
    // Calculate one step of the model
    FOR n := 0 TO 2 DO
        FOR m := 0 TO 2 DO
            X[n] := X[n] + A[n, m] * X[n];
        END_FOR;
        X[n] := X[n] + B[n] * deltaU;
    END_FOR;
    FOR n := 0 TO 2 DO
        y := y + C[n] * X[n];
    END_FOR;
    // Calculate a standard deviation
    Fit[j] := Fit[j] + (W - y) ** 2;
END_FOR; // Loop over whole prediction horizon
```

Obrázek 7.14 - Výpočet účelové funkce

Pro každý parametr jedince, který reprezentuje změnu akčního zásahu, počítá se jedna hodnota regulované veličiny. Realizace všech parametrů jedince tvoří predikovanou trajektorii regulované veličiny, která se porovnává s žádanou veličinou.

4. **Výběr nejlepšího jedince**

   Posledním krokem netopýřího algoritmu je výběr nejlepšího jedince z cele populace. Celý jedinec se uloží do proměnný *bestSolution*, ale na vystup z PLC se nastaví pouze první parametr z nejlepšího jedince.

Krok 4 je posledním krokem pro realizace netopýřího algoritmu. Po jeho provedení se do proměnný *step* nastaví hodnota 1 a veškeré procesy se znovu opakují.

Kapitola 8

Testování algoritmů na simulované soustavě

Prvotní testování navrženého řídicího systému a ověření správné funkcionality bylo provedeno simulačně, využitím vývojového prostředí Mosaic. Pro simulace byla zvolená soustava druhého řadu.

$$\begin{aligned} x_m[k+1] &= A_m x_m[k] + B_m u[k] \\ y[k] &= C_m x_m[k] + D_m u[k] \end{aligned} \tag{8.1}$$

Prvky matic původního modelu ve trávu diskrétní stavové formulace jsou uvedený v Tabulka 8.1.

Tabulka 8.1: Matice původní simulované soustavy

A_m	B_m	C_m	D_m
$\begin{bmatrix} 1{,}215 & -0{,}7358 \\ 0{,}5 & 0 \end{bmatrix}$	$\begin{bmatrix} 0{,}5 \\ 0 \end{bmatrix}$	$[0{,}4757 \quad 0{,}4732]$	0

Dalším krokem bylo sestavení diskrétní stavové formulace soustavy v diferenciálním tvaru.

$$\begin{aligned} \begin{bmatrix} \Delta x_m[k+1] \\ y[k+1] \end{bmatrix} &= \begin{bmatrix} A_m & \boldsymbol{o}_m^T \\ C_m A_m & 1 \end{bmatrix} \begin{bmatrix} \Delta x_m[k] \\ y[k] \end{bmatrix} + \begin{bmatrix} B_m \\ C_m B_m \end{bmatrix} \Delta u[k] \\ y[k] &= [\boldsymbol{o}_m \quad 1] \begin{bmatrix} \Delta x_m[k] \\ y[k] \end{bmatrix} \end{aligned} \tag{8.2}$$

Postup pro získání modelu ve trávu (8.2) je popsán ve 2. kapitole - Prediktivní řízení (viz strana 3). Obdržené matice jsou uvedené v následující tabulce:

Tabulka 8.2: Výsledné matice simulované soustavy

A	B	C
$\begin{bmatrix} 1{,}215 & -0{,}7358 & 0 \\ 0{,}5 & 0 & 0 \\ 0{,}8146 & -0{,}35 & 1 \end{bmatrix}$	$\begin{bmatrix} 0{,}5 \\ 0 \\ 0{,}2379 \end{bmatrix}$	$[0 \quad 0 \quad 1]$

Získány model soustavy byl následně zadán do řídicího algoritmu a simulačně otestován ve vývojovém prostředí Mosaic.

8.1 Analýza výsledků simulačních testů

Testovaní navrženého řídicího systému bylo provedeno pro oba optimalizační algoritmy – diferenciální evoluce a netopýří algoritmus. Nejdůležitějšími parametry algoritmů z hlediska rychlosti a kvality regulace jsou horizont predikce N_p a počet jedinců v populaci $N_{specimen}$.

Během prvních testů s algoritmy se ukázalo, že maximální délka horizonu predikce, pro kterou výpočty nového akčního zásahu netrvají moc dlouho, pro algoritmus diferenciální evoluce je $N_{p\,DE} = 5$. Pro netopýří algoritmus maximální hodnota horizontu predikce činí $N_{p\,BA} = 7$. Při větších hodnotách se prodlužuje programový cyklus PLC a výrazně se zhoršuje kvalita řízení. Výhodou BA je to, že pro stanovení nové hodnoty akčního zásahu je potřeba čtyř výpočetních kroků. V případě algoritmu diferenciální evoluce počet potřebných kroků je pět. Další krok se spouští každých 100 ms.

Také simulační testy ukázaly, že počet jedinců v populaci ovlivňuje to, jak rychle optimalizační algoritmy dokážou stanovit optimální hodnotu celkového akčního zásahu. V případě, že parametr je $N_{specimen}$ nastaven na menší hodnoty, čas potřebný pro nastavení optimální hodnoty akčního zásahu je delší. To, jak se tato vlastnost algoritmů projevuje na procesu řízení je znázorněno na obrazcích 8.1 a 8.2.

Parametry řídicích algoritmů s menším počtem jedinců jsou uvedené v Tabulka 8.3.

Tabulka 8.3: Parametry algoritmů (malé $N_{specimen}$)

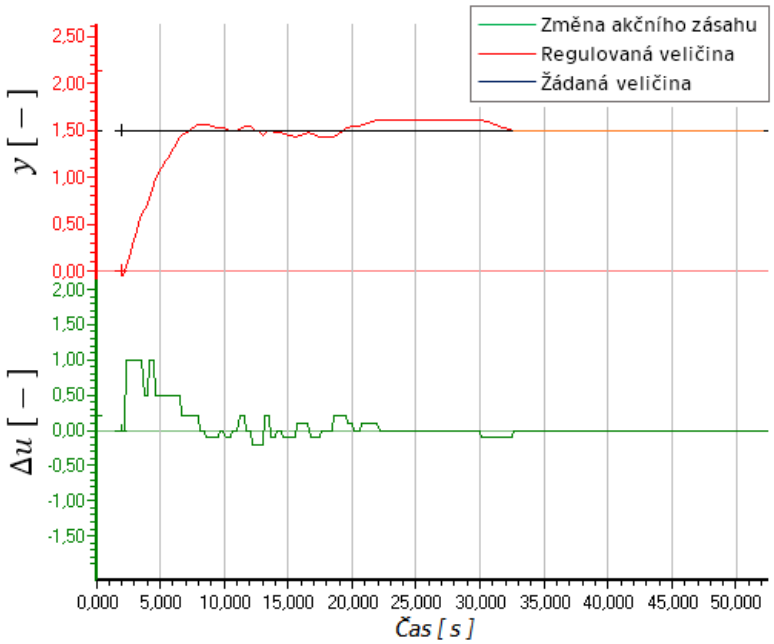
Algoritmus	N_p [-]	$N_{specimen}$ [-]	K [-]
BA	7	15	0,1
DE	5	15	0,1

Parametry řídicích algoritmů s větším počtem jedinců v populace jsou uvedené v Tabulka 8.4.

Tabulka 8.4: Parametry algoritmů (větší $N_{specimen}$)

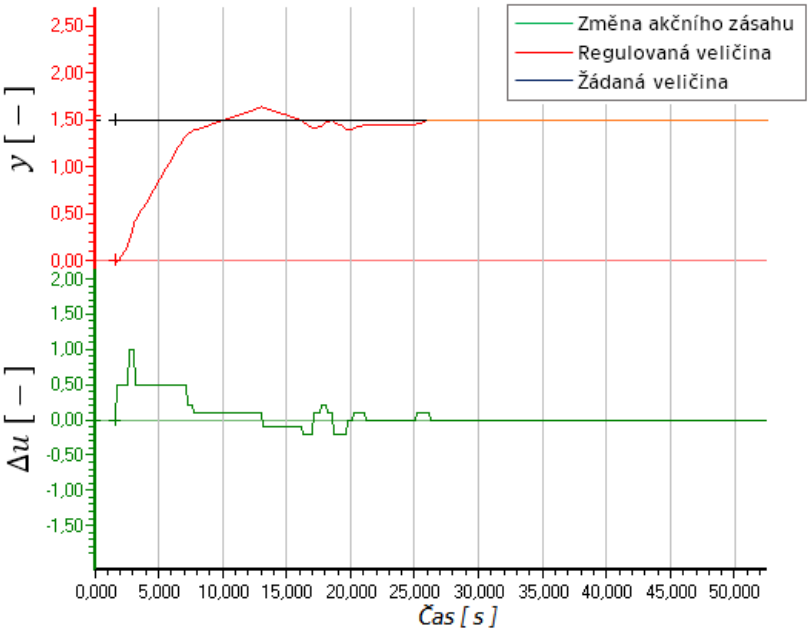
Algoritmus	N_p [-]	$N_{specimen}$ [-]	K [-]
BA	7	30	0,1
DE	5	30	0,1

Průběh simulačního řízení s implementovaným BA algoritmem pro menší počet jedinců:



Obrázek 8.1 – Simulační řízení BA ($N_{specimen} = 15$)

Průběh simulačního řízení s implementovaným DE algoritmem pro menší počet jedinců:

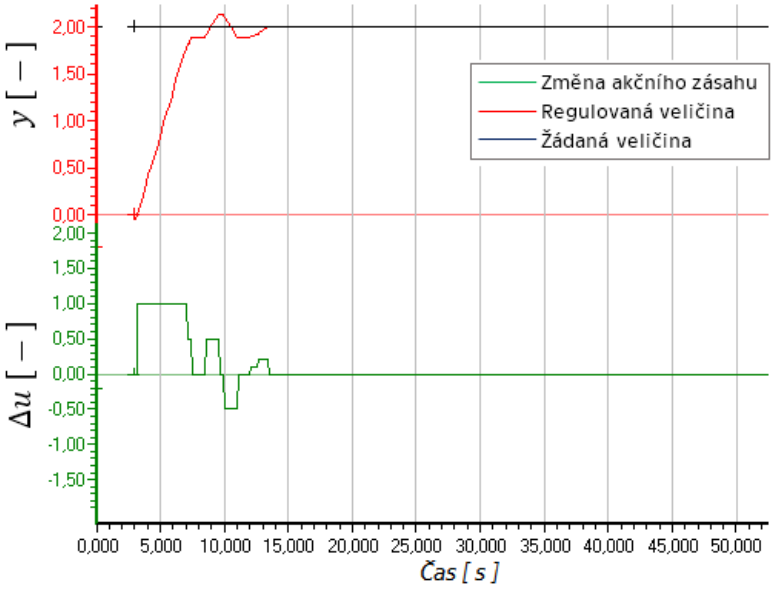


Obrázek 8.2 – Simulační řízení DE ($N_{specimen} = 15$)

Z průběhů simulačního řízení, které jsou zobrazené na obrazcích 8.1 a 8.2, je patrno, že řídicí systém s BA algoritmem potřebuje méně času na to, aby se řízená veličina dostala do blízkého okolí žádané veličiny. Ale algoritmus DE dokáže rychleji najit optimální ustálenou hodnotu akčního zásahu.

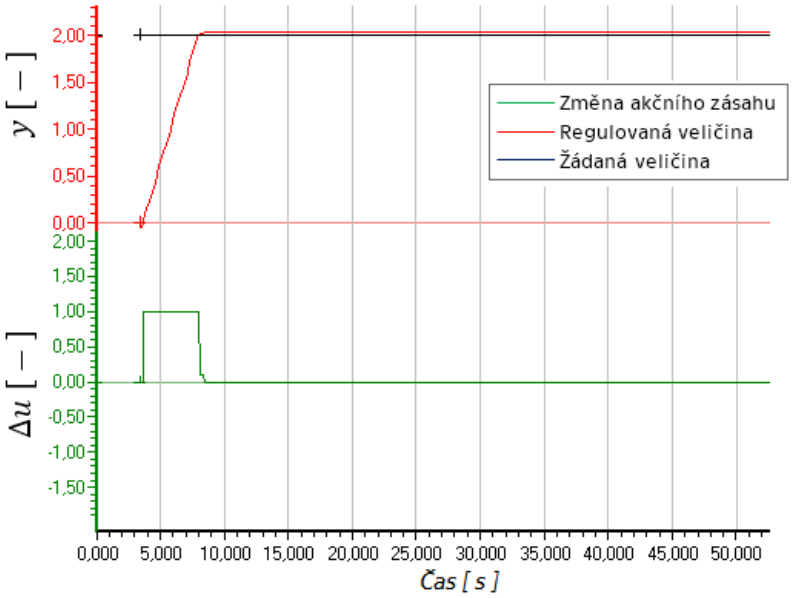
Další průběhy simulačního řízení znázorňují zlepšení kvality řízení při větším nastaveném parametru počtu jedinců $N_{specimen}$.

Průběh simulačního řízení s implementovaným BA algoritmem pro větší počet jedinců:



Obrázek 8.3 – Simulační řízení BA ($N_{specimen} = 30$)

Průběh simulačního řízení s implementovaným DE algoritmem pro větší počet jedinců:



Obrázek 8.4 – Simulační řízení DE ($N_{specimen} = 30$)

Z obrázku 8.4 je vidět, že pomocí algoritmu diferenciální evoluce lze dosáhnout optimálního řízení s minimálním počtem změn akčního zásahu provedených v správný čas. Ale pak během celého průběhu regulace se neodstranila trvalá regulační odchylka.

Netopýří algoritmus dokázal najit optimální hodnotu akčního zásahu. Na jednou stranu bylo provedeno vice změn akční veličiny a spotřebováno vice času něž v případě DE algoritmu, ale na druhou stranu byla odstraněna trvalá regulační odchylka.

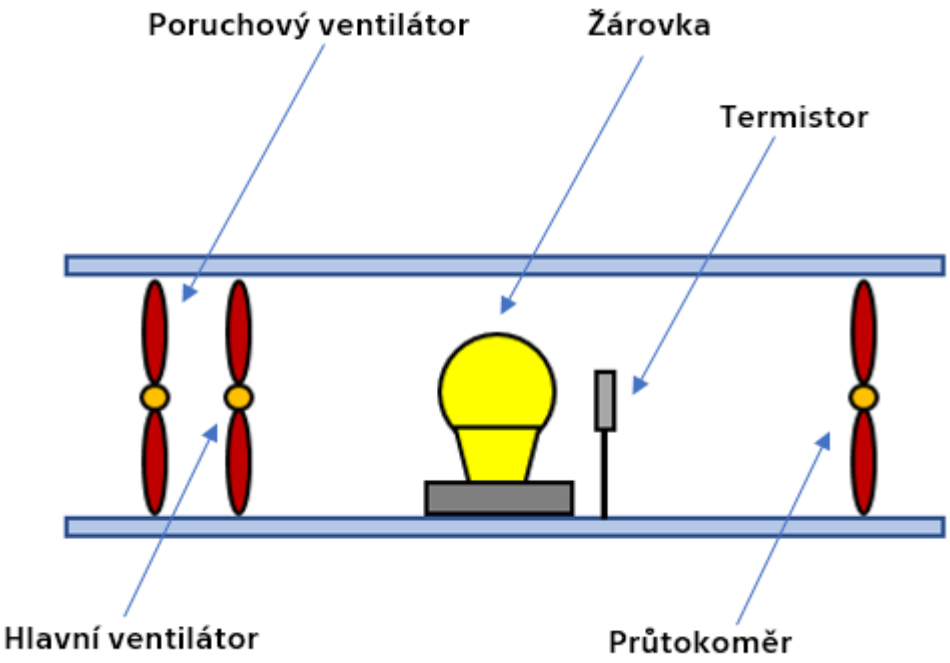
Kapitola 9

Testování algoritmu na reálné soustavě

Testy popsané v předchozí kapitole byly provedeny pouze simulačně. Ale pravda je, že chovaní řídicího systému během simulace se může lišit od chovaní systému při řízení reálné soustavy. Proto dalším krokem ověření správné funkcionality algoritmu je testovaní algoritmu na reálné laboratorní úloze.

9.1 Popis úlohy

Reálnou soustavou zvolenou pro testovaní navržených algoritmů prediktivního řízení je laboratorní soustava "Teplovzdušný model", která se nachází v laboratoři 111 Fakulty Strojní.



Obrázek 9.1 – Schéma laboratorní úlohy „Teplovzdušný model"

Laboratorní úloha je dána krytým tunelem, ve kterém jsou umístěny: dva ventilátory (hlavní a poruchový), žárovka, termistor a vrtulkový průtokoměr. Zjednodušena schéma soustavy je zobrazena no obrázku 9.1.

Existuje několik variant volby vstupních a výstupních veličin soustavy. V rámci zadané diplomové práce, jako akční veličina byla zvolena hodnota napětí na hlavním

ventilátoru $(u \in [0,10]\,V)$. Regulovanou veličinou je napětí na průtokoměru $(y \in [0,10]\,V)$.

9.2 Identifikace modelu soustavy

Prvním krokem po seznámení s laboratorní úlohou byla identifikace řízené soustavy. Pro řídicí systémy využívající metodu MPC kvalita modelu soustavy hraje rozhodující roli.

Identifikace dynamického modelu se prováděla pomocí speciálního nástroje „System Identification Toolbox", který je součásti vývojového prostředí MATLAB. Pro identifikaci laboratorní úloha teplovzdušného modelu byla odpojena od PLC Tecomat Foxtrot a následně připojena k počítači s nainstalovaným nástrojem „System Identification Tollbox". Hlavním problémem pro identifikaci modelu soustavy bylo to, že řízená soustava teplovzdušného modelu je velice závislá na okolních podmínkách. Při změnách teploty, tlaku nebo vlhkostí v laboratoří, také se měnily i parametry identifikovaného modelu. Proto bylo potřeba opakovat celý proces identifikace před každým testem navrženého systému řízení.

Výstupem z nástroje „System Identification Tollbox" jsou parametry pro přenosovou funkci druhého řadu s dopravním zpožděním ve tvaru:

$$G(s) = \frac{K_u}{a_2 s^2 + a_1 s + 1} \cdot e^{-sT_d} \tag{9.1}$$

Příklad parametrů, obdržených během procesu identifikace, je uveden v následující tabulce:

Tabulka 9.1: Parametry identifikovaného modelu soustavy.

a_2	a_1	K_u	T_d
12,637	9,643	1,642	0,936

Získaný model byl následně převeden do diskrétního času s periodou vzorkování 0,4 sekundy a pak do tvaru modelu ve stavové formulaci. Matice identifikovaného systému jsou uvedené v Tabulka 9.2.

Tabulka 9.2: Původní matice identifikovaného systému

A_m	B_m	C_m	D_m
$\begin{bmatrix} 1{,}726 & -0{,}737 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0{,}125 \\ 0 \end{bmatrix}$	$[0{,}07523 \quad 0{,}06796]$	0

Dalším krokem bylo sestavení diskrétní stavové formulace systému v diferenciálním tvaru:

$$\begin{bmatrix}\Delta x_m[k+1]\\ y[k+1]\end{bmatrix} = \begin{bmatrix}A_m & \boldsymbol{o}_m^T\\ C_m A_m & 1\end{bmatrix}\begin{bmatrix}\Delta x_m[k]\\ y[k]\end{bmatrix} + \begin{bmatrix}B_m\\ C_m B_m\end{bmatrix}\Delta u[k]$$
$$y[k] = [\boldsymbol{o}_m \quad 1]\begin{bmatrix}\Delta x_m[k]\\ y[k]\end{bmatrix} \tag{9.2}$$

Převodem systému do stavové formulace v diferenciálním tvaru byly stanoveny následující matice soustavy:

Tabulka 9.3: Výsledné matice reálné soustavy

A	B	C
$\begin{bmatrix}1{,}726 & -0{,}737 & 0\\ 1 & 0 & 0\\ 0{,}0752 & -0{,}068 & 1\end{bmatrix}$	$\begin{bmatrix}0{,}125\\ 0\\ 0{,}0094\end{bmatrix}$	$[0 \quad 0 \quad 1]$

Obdržené výsledky pak byly naprogramovány do navrženého systému řízení. Následně celý algoritmus prediktivního řízení jsem nahrál do PLC a vyzkoušel funkcionalitu pro dvě konfigurace řídicího systému – s algoritmem diferenciální evoluce a s netopýřím algoritmem.

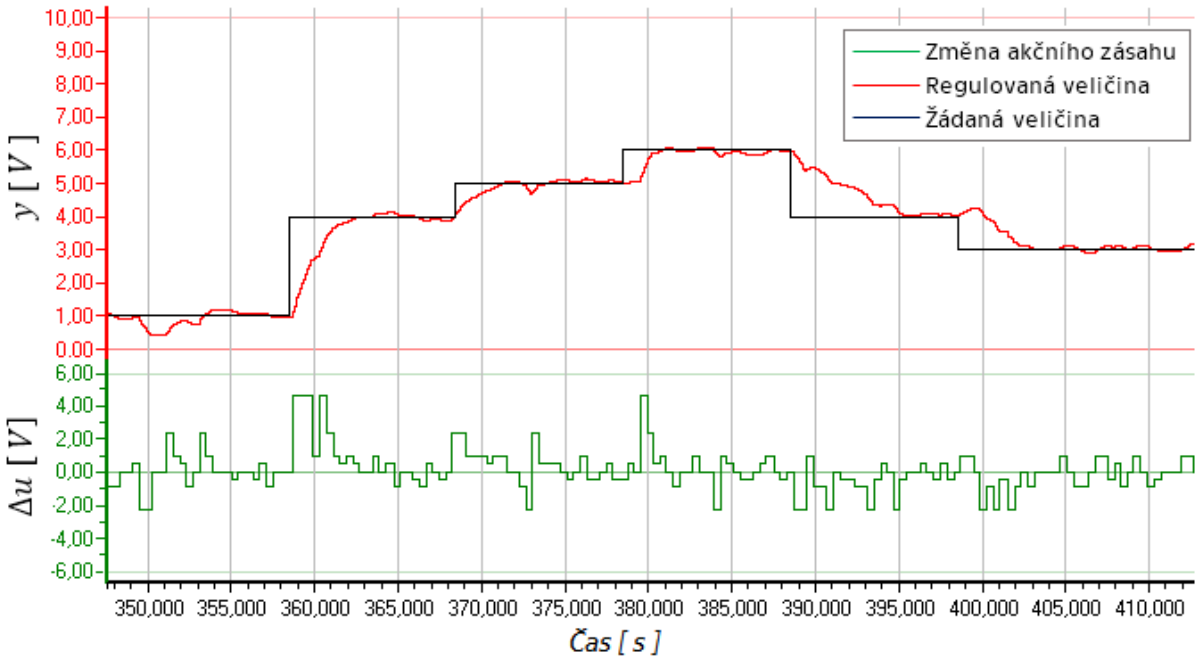
9.3 Analýza výsledků testů reálné soustavy

Pro porovnaní algoritmu diferenciální evoluce a netopýřího algoritmu řízení reálné laboratorní úlohy bylo provedeno pro oba algoritmy. V následující tabulce jsou uvedené parametry řídicího systému pro každý algoritmus:

Tabulka 9.4: Parametry algoritmů pro laboratorní úlohu

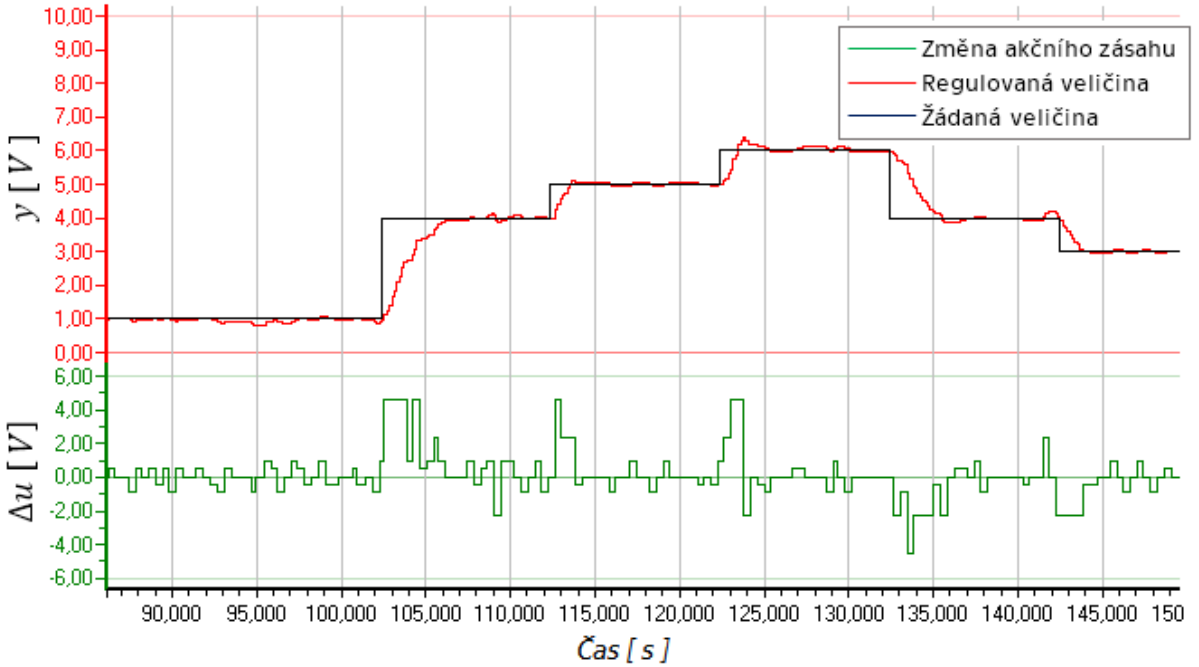
Algoritmus	N_p [-]	$N_{specimen}$ [-]	K [-]
BA	7	35	0,4
DE	5	30	0,4

Průběh řízení reálné laboratorní úlohy s implementovaným BA algoritmem



Obrázek 9.2 – Řízení reálné laboratorní úlohy (Netopýří algoritmus)

Průběh řízení reálné laboratorní úlohy s implementovaným DE algoritmem



Obrázek 9.3 – Řízení reálné laboratorní úlohy (Diferenciální evoluce)

Při porovnaní obrázků 9.2 a 9.3 je vidět, že průběhy regulace různých algoritmů nejsou stejné, ale rozdíl v kvalitě regulace není velký. Na druhou stranu je patrno, že

algoritmus DE dokáže udržovat hodnoty regulované veličiny okolo žádané veličiny lépe, něž BA algoritmus. Pravděpodobně algoritmus diferenciální evoluce hledá optimální hodnoty akčních zásahů lépe, přestože výpočet nové hodnoty akčního zásahu pro DE algoritmus trvá o 100 milisekund díl něž pro BA a nastavena délka horizontu predikce je kratší něž pro netopýří algoritmus.

Závěr

V rámci této práce seznámil jsem se s pokročilou metodou řízení procesů – prediktivní řízení. Tato metoda má zajímavou strategii řízení s výhledem do budoucna pomocí predikce chovaní systémových procesů. Právě proto strategie prediktivního řízení požaduje dostatečně přesný model řízené soustavy. Kromě toho, pro predikci chovaní dynamických modelu je potřeba analyzovat a vyhodnocovat relativně velké množství dat za nejkratší možnou dobu. U lineárních modelu lze tento problém vyřešit například použitím kvadratického programovaní. Ale v případě nelineárních soustav použití podobných metod vede k prodloužení výpočetního času a následně k výraznému zkrácení délky horizontu predikce.

Pro nalezení možného řešení optimalizačního problému prediktivního řízení prostudoval jsem některé algoritmy globální optimalizace inspirované živou přírodou. Detailně byly prostudovaný dva metaheuristické algoritmy – algoritmus diferenciální evoluce a netopýří algoritmus. Tyto algoritmy jsou podobny z hlediska práce s množstvím jedinců, reprezentujících možná řešení optimalizačního problému. Ale strategie zpracovaní definovaného prostoru a generovaní nových řešení se u těchto algoritmů jsou odlišné. Právě pro porovnaní a analýzu chování algoritmů bylo rozhodnuto navrhnout a naprogramovat oba algoritmy.

Algoritmy byly naprogramované a vyzkoušené pomocí vývojového prostředí Mosaic pro PLC Tecomat Foxtrot. Ještě při studovaní algoritmu DE všimnul jsem se, že pro výpočty se hodně používají náhodná čísla. V případě použití PLC to muže být problematické. Navíc pří návrhu se ukázalo, že pro DE algoritmus je potřeba vice cyklu PLC na stanovení nové hodnoty akčního zásahu. Ale překvapivě bylo to, že jak během simulačních testů, tak i během ověřování algoritmů na reálné laboratorní úloze kvalita regulace DE algoritmu byla lepší oproti algoritmu BA.

Na druhou stranu, netopýří algoritmus lépe pracuje s definovaným prostorem a analyzuje více různých trajektorií ve všech směrech prohledávaného prostoru. Tato vlastnost může být výhodou v případě řízení nelineárních systémů.

Dva systémy řízení s různými optimalizačními algoritmy byly otestované pouze na lineárních soustavách. A podle výsledků simulačních testů a testů na reálné laboratorní úloze lze předpokládat, že algoritmy mohou být použité pro realizaci systému prediktivního řízení nelineárních systémů.

Literatura

[1] I. ZELINKA, Umělá inteligence v problémech globální optimalizace, Praha: BEN-technická literatura, 2002.

[2] P. Liuping Wang, Model Predictive Control System Design and Implementation Using Matlab, Londýn: Springer, 2009.

[3] M. B.Kouvaritakis, Model Predictive Control - Classical, Robust and Stochastic, Londýn: Springer, 2016.

[4] E.F.CAMACHO, C.BORDONS, "Model Predictive Control. 2nd ed.," 2007. [Online]. Available: http://een.iust.ac.ir/profs/Shamaghdari/MPC/Resources. [Accessed 17 4 2019].

[5] J. Rossiter, Model-Based Predictive Control. A Practical Approach, Florida: CRC Press LLC, 2000 N.W., 2005.

[6] Andreas Antoniou, Wu-Sheng Lu, Practical Optimization: Algorithms and Engineering Applications, New York: Springer Science+Business Media, LLC, 2007.

[7] YANG Xin-She, Nature-inspired metaheuristic algorithms. 2nd ed., Frome: Luniver Press, 2010.

[8] R. M. S. J. A. L. Kenneth V. Price, Differential Evolution. A Practical Approach to Global Optimization, Berlin: Springer-Verlag , 2005.

[9] TECO Advanced Automation, "PROGRAMOVATELNÉ AUTOMATY TECOMAT FOXTROT," 22 Leden 2017. [Online]. Available: https://www.tecomat.cz/modules/DownloadManager/download.php?alias=txv00410__01__general__foxtrot. [Accessed 9 6 2019].

[10] Teco a.s., "Programování PLC podle normy IEC 61 131-3," Listopad 2007. [Online]. Available: https://www.tecomat.cz/modules/DownloadManager/download.php?alias=txv00321__01__mosaic__progiec__cz. [Accessed 16 7 2019].

[11] Teco a.s., "ZAČÍNÁME V PROSTŘEDÍ MOSAIC," Duben 2010. [Online]. Available: https://www.tecomat.cz/modules/DownloadManager/download.php?alias=txv00320__01__mosaic__progstart__cz. [Accessed 15 3 2019].

Seznam použitých zkratek a symbolů

A_0 – počáteční hodnota hlasitosti signálu virtuálních netopýrů

A_m – matice dynamiky modelu ve stavové formulaci

A_{min} – minimální hodnota hlasitosti signálu virtuálních netopýrů

B_m – matice pravých stran modelu ve stavové formulaci

C_m – výstupní matice systému modelu ve stavové formulaci

D_m – výstupní matice systému modelu ve stavové formulaci

F_{cost} – účelová funkce

N_p – horizont predikce

$N_{specimen}$ – počet jedinců v populaci

$P_{\boldsymbol{u}}$ – zkušební populace jedinců

$P_{\boldsymbol{v}}$ – přechodová populace jedinců

$P_{\boldsymbol{x}}$ – aktuální populace jedinců

f_{min} – minimální frekvence signálu

$x_m[k]$ – stavová proměnná v diskrétním čase

$\hat{y}[k]$ – Predikovaný výstup systému v diskrétním čase

$\boldsymbol{u_j}$ – zkušební vektor

$\boldsymbol{v}_j$ – pro DE – šumový vektor, pro BA – rychlost netopýru

$\boldsymbol{x}_j$ – možné řešení optimalizačního problému (jedinec)

$\Delta u[k]$ – přírůstek akčního zásahu v diskrétním čase

A – matice dynamiky modelu v diferenciálním tvaru

B – matice pravých stran modelu v diferenciálním tvaru

BA – netopýří algoritmus (*Bat Algorithm* – anglický)

C – výstupní matice systému modelu v diferenciálním tvaru

DE – diferenciální evoluce (*Differential Evolution* – anglický)

MIMO – *Multi-Input-Multi-Output* – anglický

MPC – Prediktivní řízení (*Model Predictive Control* – anglický)

PLC – programovatelný automat (*Programmable Logic Controller* – anglický)

SISO – *Single-Input-Single-Output* – anglický

Cr – práh křížení

F – mutační konstanta

K – součinitel přírůstků akčních zásahů

f	–	frekvence signálu virtuálních netopýrů
r	–	rychlost emise pulsu signálu virtuálních netopýrů
$u[k]$	–	akční veličina v diskrétním čase
$w[k]$	–	žádaná veličina v diskrétním čase
$y[k]$	–	řízená veličina v diskrétním čase

Seznam obrázků

Seznam tabulek