SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

EVIDENČNÉ ČÍSLO: FCHPT-123283-77593

ROBOTICKÁ OPTIMALIZÁCIA

BAKALÁRSKA PRÁCA

2017 **Alexey Morozov**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

Evidenčné číslo: FCHPT-123283-77593

ROBOTICKÁ OPTIMALIZÁCIA

BAKALÁRSKA PRÁCA

Študijný program: B-AIMCHP automatizácia, informatizácia a manažment v chémii a potravinárstve
Študijný odbor: 5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky (FCHPT)
Vedúci záverečnej práce: doc. Ing. Michal Kvasnica, PhD.
Konzultant: Ing. Martin Kalúz, PhD

Bratislava 2017 **Alexey Morozov**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2016/2017
Evidenčné číslo: FCHPT-123283-77593

STU FCHPT

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Alexey Morozov**
ID študenta:	77593
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Vedúci práce:	doc. Ing. Michal Kvasnica, PhD.
Konzultant:	Ing. Martin Kalúz, PhD.

Názov práce: **Robotická optimalizácia**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je implementovať hardvérový systém na riešenie optimalizačných úloh. Takýto systém sa bude skladať z jedného alebo z viacerých robotických vozidiel, ktorých úlohou je nájsť miesto v laboratóriu najviac vyhovujúce zvolenej účelovej funkcii. Jednotlivé roboty budú navzájom komunikovať tak, aby poskytli čo najlepši odhad gradientu, ktorý sa využije na nájdenie smeru optimalizácie. Využijú sa taktiež optimalizačné algoritmy založené na náhodnom prehľadávaní priestoru, ako sú napr. Luus-Jaakola alebo Nelder-Mead metódy.

Riešenie zadania práce od: 13. 02. 2017

Dátum odovzdania práce: 14. 05. 2017

Alexey Morozov
študent

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Poďakovanie

Chcel by som poďakovať môjmu školiteľovi Michalovi Kvasnicovi za jeho rady a návrhy počas vypracovávania daného projektu. Taktiež ďakujem mojej spolužiačke Sone Čaklošovej za jej pomoc v kontrole a opravovaní slovenskej gramatiky. Ďakujem.

Abstrakt

Cieľom práce je použitie hardvérového systému, v našom prípade popísaného nelineárnym matematickým modelom na riešenie optimalizačných úloh. Tieto úlohy riešime bezgradientovými optimalizačnými metódami. Jadrom tejto práce je tvorba algoritmov na ovládanie matematického modelu vacerými metódami. Celkovo sú použité tri metódy: Luus-Jaakolova metóda, Nelder-Meedova metóda a metóda optimalizácie rojom častíc. Každá metóda je popísaná v jednotlivej podkapitole. Celé programovanie je uskutočnené v programovacom prostredí „Matlab“. Výsledkom tejto práce sú tri funkčné programy na riešenie optimalizačných úloh bezgradientovými metódami.

Kľúčové slová: Optimalizácia, bezgradientová optimalizačná metóda, optimálne riešenie, robotické vozidlo.

Abstract

The purpose of that project is using a hardware system, in our case nonlinear mathematic model, for solving optimization problems. Those problems are solving by using gradient-free optimization methods. The main part of this project is creating programs for different optimization methods. There are three methods in this project. They are Luus-Jaakol method, Nelder-Mead method and Particular Swarm Optimization method. Each method is described in separate chapter. The whole programing is in multi-paradigm numerical computing environment „Matlab“. The result of this project is three fully functional programs for solving optimization problems by using gradient-free optimization methods.

Key words: Optimization, gradient-free optimization method, optimal solution, robot.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Optimalizácia je vo všeobecnosti proces hľadania extrémov danej funkcie *f* na nejakej množine *M*. V prípade, že poznáme tvar optimalizovanej funkcie, používame takzvané gradientové optimalizačné metódy, ako napríklad analytická metóda hľadania optima, eliminačná metóda, metóda Lagrangeových násobičov a ďalšie. Spoločným znakom týchto metód je, že hodnoty premenných pri ktorých má optimalizovaná funkcia svoje minimum sa získavajú zostrojením gradientu účelovej funkcie a zistením, pri akých hodnotách premenných je jeho hodnota rovná nule. Bezgradientové metódy sa používajú vtedy, keď nemôžeme použiť gradientové metódy. Napríklad, keď účelová funkcia nie je k dispozícii alebo sú jej matematické úpravy príliš náročné. Pri bezgradientových metódach porovnávame postupne získané hodnoty kritérií optimality.

Cieľom tejto práce je skúmanie robotickej optimalizácie. Je to spôsob optimalizácie, ktorá rieši optimalizačné úlohy za použitia hardwerového systému tým, že hodnota účelovej funkcie v nezávislom bode sa nezisťuje na základe matematických úprav účelovej funkcie, ale odčítava sa zo senzora, ktorý je umiestnený na robotovi. Využitie takého spôsobu optimalizácie je možné na vyhodnotenie vlastností reálneho prostredia, do ktorého sa napríklad človek nemôže dostať. Prípadom takéhoto prostredia môže byť okolie jadrovej elektrárne, v ktorej nastala havária a v ktorej potrebujeme nájsť najznečistenejšie miesto alebo jazero, kde potrebujeme zistiť maximálnu hĺbku pre vedné výskumy.

Aby sme mohli riešiť optimalizačné problémy metódou robotickej optimalizácie potrebujeme zostrojiť program, ktorý bude minimalizovať funkciu pohybom robotov. Následne vieme problém robotickej optimalizácie rozdeliť na jednotlivé kroky, ktorých spracovanie tvorí ciele tejto práce. Jednotlivé kroky sú nasledovné:

1. Tvorba algoritmu na koordináciu robotov tak, aby našli minimum funkcie
2. Lokálne riadenie robota, aby išiel na zvolenú pozíciu
3. Implementácia riadiaceho algoritmu v Simulinku
4. Implementácia minimalizačného algoritmu v Matlabe
5. Prepojenie minimalizačného a riadiaceho algoritmu
6. Testovanie funkčnosti na 3 rôznych účelových funkciách

Prvým krokom je tvorba algoritmov, ktoré riešia optimalizačné úlohy bezgradientovými metódami. V druhom kroku použijeme grafické prostredie imitačného modelovania Simulink a jeho súčasť interaktívny inštrument Stateflow, ktorým vieme generovať príkazy pre pohyb robota. V treťom kroku vytvárame schému v Simulinku, kde riešime pohyb robota z jedného bodu do druhého. Robota, ktorý je popísaný matematickým modelom spojíme s blokom Stateflow vytvoreným v predošlom kroku. V štvrtom kroku pre každú optimalizačnú metódu vytvárame program v Matlabe. V piatom kroku modifikujeme programy zo štvrtého kroku, aby nový skúmaný bod nebol ten, ktorý nám vypočítal program, ale ten, v ktorom sa zastavil robot. V poslednom kroku používame vytvorené programy na riešenie optimalizačných úloh a na základe rýchlosti a presnosti hľadania minima funkcie posudzujeme efektívnosť a vhodnosť optimalizačných metód.

V našom prípade pri vyhodnotení funkčnosti a efektívnosti bezgradientových metód budeme používať tri dvojrozmerné funkcie: $x^2 + y^2$, Himmelblauho a Cross-in-tray. Tieto funkcie vyzerajú nasledovne:

$$z = x^2 + y^2 \tag{1}$$

$$z = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \tag{2}$$

$$z = -0.0001 \left(\left| sin(x)\, sin(y)\, e^{\left(\left| 100 - \frac{\sqrt{x^2+y^2}}{\pi} \right| \right)} \right| + 1 \right)^{0.1} \tag{3}$$

Minimá týchto funkcií sú uvedené v Tab 1:

Tab. 1 Riešenia optimalizačných problémov

Funkcia	$[x_{min}, y_{min}]$	$f(x_{min}, y_{min})$
1	[0 0]	0
2	[3 2] [-2.805118 3.131312] [-3.779310 -3.283186] [3.584428 -1.848126]	0 0 0 0
3	[1.34941 -1.34941] [1.34941 1.34941] [-1.34941 1.34941] [-1.34941 -1.34941]	-2.06261 -2.06261 -2.06261 -2.06261

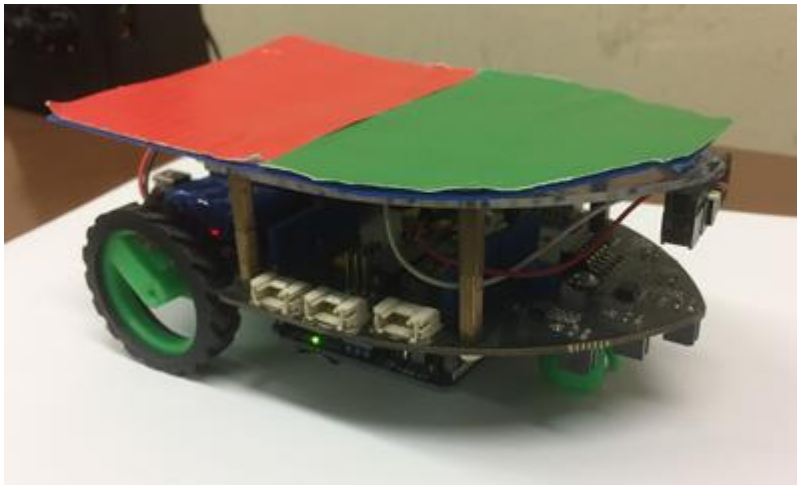
Tieto funkcie sú našimi optimalizačnými problémami, ktoré budeme riešiť pomocou robotickej optimalizácie. Sú v tvare $z = f(x, y)$. Preto budeme ďalej hovoriť o preskúmavaní x, y oblasti alebo o x,y ploche. V tejto práci využívame matematický model robotického vozidla alebo robota. Výhodou použitia matematického modelu oproti reálnemu robotovi je, že model dokáže spoľahlivo komunikovať s programom. Z toho vyplýva, že celou logikou programu je určovanie x a y súradníc bodu, do ktorého ide robot. Keď robot dosiahne cieľový bod, program vyhodnotí, či je v tomto bode hľadané optimum a či je potrebné poslať robota do ďalšieho bodu.

Štruktúra tejto práce je nasledovná: v prvej kapitole riešime problém ovládania robotického vozidla. V druhej kapitole riešime optimalizačné problémy robotickou optimalizáciou. V podkapitole 2.1 na riešenie optimalizačných úloh aplikujeme Luus-Jaakolovu

metódu. V podkapitole 2.2 skúmame Nelder-Meadovu optimalizačnú metódu. V podkapitole 2.3 skúmame metódu optimalizácie rojom častíc. Podmienkou nájdenia globálneho optima bude nemožnosť nájdenia ďalšieho bodu, v ktorom by hodnota účelovej funkcie bola menšia ako v bode predchádzajúcom.

1. Ovládanie robotického vozidla

Pri riešení optimalizačných úloh robotickou optimalizáciou potrebujeme robota, na ktorom bude umiestnený senzor. Náš robot je uvedený na obrázku 1. Na konštrukcii robota je zaujímavé, že každé koliesko je ovládané osobitným motorom a môžeme tak komunikovať s každým motorom nezávisle.



Obr. 1 Robot

V rámci tejto práce nebudeme narábať s reálnym robotom, ale s jeho matematickým modelom, ktorý je opísaný rovnicami 4 až 6.

$$\frac{d\,x(t)}{dt} = \frac{r}{2}\cos\big(\theta(t)\big)\,(\omega_L(t) + \omega_R(t)) \tag{4}$$

$$\frac{d\,y(t)}{dt} = \frac{r}{2}\sin\big(\theta(t)\big)\,(\omega_L(t) + \omega_R(t)) \tag{5}$$

$$\frac{d\,\theta(t)}{dt} = \frac{r}{L}(\omega_R(t) - \omega_L(t)) \tag{6}$$

Kde:

r - polomer kolieska = 3e-3 [m]

L - vzdialenosť kolies = 1e-2 [m]

ω_L - uhlová rýchlosť ľavého kolieska [rad/s]

ω_R - uhlová rýchlosť pravého kolieska [rad/s]

θ - uhol natočenia robota voči x-ovej osi[rad]

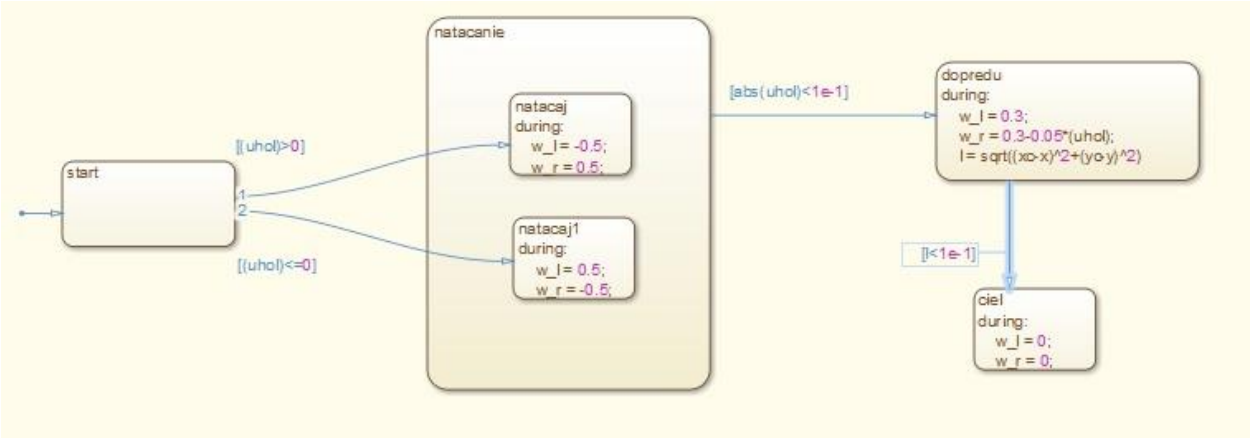
Tento model sa správa spôsobom: vstupmi do modelu sú hodnoty uhlových rýchlosti pre kolieska v čase a ako výstupy dostávame zmenu polohy robota voči začiatočnej polohe. Táto poloha obsahuje hodnoty x-ovej a y-ovej súradnice a taktiež uhol natočenia robota vzhľadom k osi x.

Hodnotu účelovej funkcie v danom bode budeme zisťovať iba na základe polohy robota v priestore. Aby sme mohli vyhodnocovať funkciu v rôznych bodoch priestoru, potrebujeme vedieť pohybovať robotom. Tento pohyb je v tvare úseku z jedného bodu do druhého. Takýto pohyb robot vykonáva v dvoch fázach: v prvej fáze sa natáča do vhodného smeru a v druhej ide rovno do cieľového bodu.

To vyriešime v dvoch krokoch: V prvom si pomocou interaktívneho inštrumentu Stateflow graficky vytvoríme postupnosť príkazov na pohyb a zastavenie robota. V druhom kroku vytvárame v Simulinku schému, pomocou ktorej vieme presunúť robota z bodu A do bodu B.

1.1 Stateflow

Stateflow je interaktívny inštrument modelovania systémov, ktoré sú riadené aktivitami. Túto vlastnosť využijeme na vytváranie programu potrebného pre riadenie pohybu robota. Program v Stateflowe vytvárame graficky a je uvedený na obrázku 2.



Obr. 2 Stateflow Chart

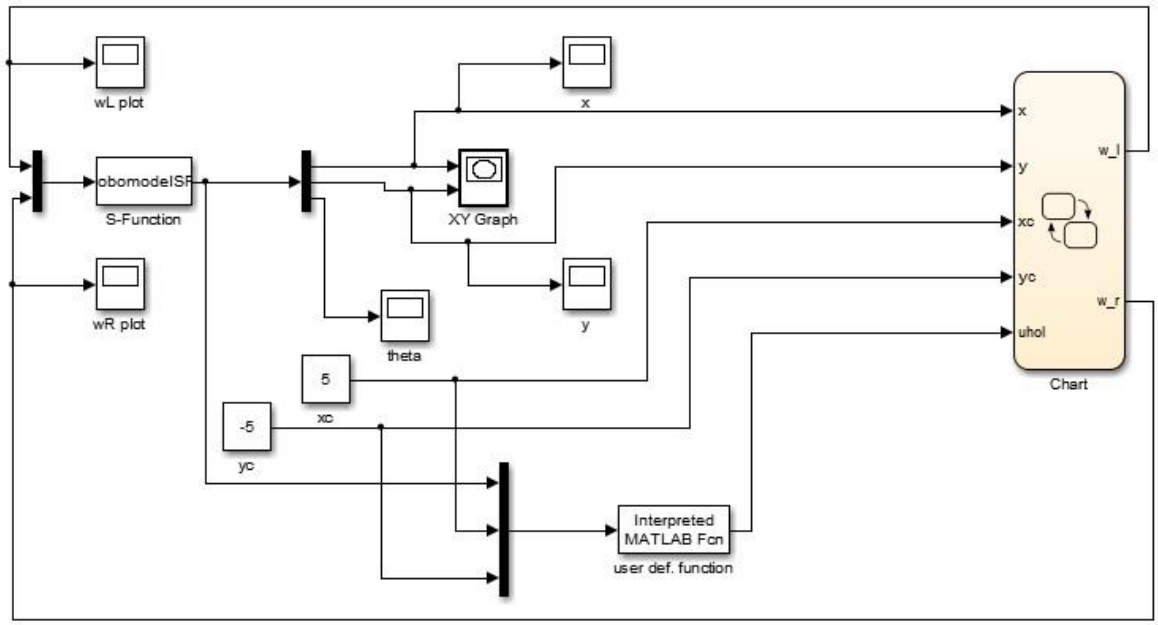
Program Stateflow berie ako vstup aktuálnu polohu robota označenú ako *x* a *y*, polohu cieľového bodu *xc* a *yc* a uhol, o ktorý sa má robot otočiť. Úlohou programu je na základe analýzy vstupujúcich dát generovať v čase uhlové rýchlosti *w_r* a *w_l* pre kolieska. Program pracuje nasledujúcim spôsobom: v prvej fáze natáčame robota tak, že najprv vyhodnotí akým smerom treba robota natočiť. Na to slúži vstup *uhol*, ktorý môže nadobúdať kladnú aj zápornú hodnotu. Ak je *uhol* kladný, robota natáčame proti smeru pohybu hodinových ručičiek. V prípade, že je *uhol* záporný robota natáčame v smere hodinových ručičiek. Robota natáčame tým, že posielame pre jedno koliesko kladnú hodnotu uhlovej rýchlosti a pre druhé zápornú dovtedy, pokiaľ uhol nebude menší ako 0,1 rad. Uvažujeme túto toleranciu, pretože model neudáva dáta spojito ale diskrétne. Ak by sme chceli natočiť robota presne, bola by veľká pravdepodobnosť, že schéma nestihne zastaviť robota a on by pokračoval v otáčaní. V druhej fáze nútime robota ísť rovno tak, že jednému koliesku posielame stále konštantnú rýchlosť a druhému takú istú rýchlosť zmodifikovanú o uhol, ktorý v aktuálnom čase zvierajú vektor natočenia robota a vektor od robota k cieľovému bodu. Takúto modifikáciu zavádzame, aby bolo možné korigovať robota počas jeho pohybu. Robota necháme pohybovať sa týmto spôsobom, kým nebude splnená podmienka na dosiahnutie cieľového bodu taká, že vzdialenosť *l* medzi robotom a cieľovým bodom je menšia ako 0,1. Vzdialenosť *l* sa vypočítava za použitia rovnice 7.

$$l = \sqrt{(xc - x)^2 + (yc - y)^2} \qquad (7)$$

V tomto prípade opäť uvažujeme chybu vyhodnotenia diskrétnych dát.

1.2 Simulink

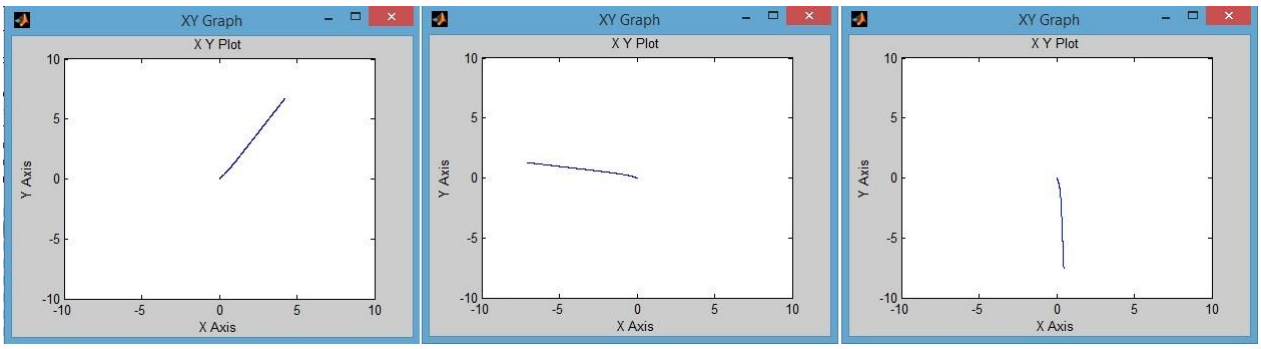
V tomto kroku vytvárame schému v Simulinku, pomocou ktorej vieme presunúť robota z bodu A do bodu B. Cela schéma je uvedená na obrázku č. 3.



Obr. 3 Simulink Schéma

V tejto schéme spájame matematický model robota a Stateflow Char. Hlavnými blokmi tejto schémy sú model robota spracovaný do tvaru S-Funkcie, blok Stateflow Chart a blok obsahujúci funkciu, ktorá vypočítava, o aký uhol sa má robot otočiť.

Do schémy najskôr inicializujeme začiatočnú polohu robota a polohu cieľového bodu. Ďalej sa vypočíta uhol, o ktorý sa má robot otočiť a to uhol medzi vlastným vektorom natočenia robota a vektorom, ktorý smeruje od robota k cieľovému bodu. Potom začne pracovať blok Stateflow Chart, ktorý v čase generuje uhlové rýchlosti a posiela ich do modelu ktorý bude meniť svoju polohu. Táto poloha je zložená z x-ovej a y-ovej polohy robota a uhla jeho natočenia voči osi x. Následne sa pomocou bloku *XY Graph* vykresľuje trajektória pohybu robota. Na obrázku č. 3 je vidieť, že pomocou tejto schémy je naozaj možné posunúť robota do ľubovoľného bodu na ploche.



Obr. 4 Výsledky simulácií

2. Robotická optimalizácia

Robotická optimalizácia je veľká časť práce, v rámci ktorej skúmame rôzne optimalizačné metódy. Pre každú metódu budeme realizovať ciele 1, 4, 5 a 6.

2.1 Luus-Jaakolova metóda

Ako prvú v tejto práci skúmame metódu Luus-Jaakola. Je to heuristická metóda pre globálnu optimalizáciu. Tato metóda rieši optimalizačné problémy za využitia jedného robota.

2.1.1 Minimalizačný algoritmus

Tato metóda vyhodnocuje dáta podľa nasledujúceho princípu. Najprv náhodne vyberieme bod v sústave súradníc a ďalší bod náhodne hľadáme v okolí prvého bodu. Ak má účelová funkcia v novom bode lepšiu hodnotu ako v predchádzajúcom bode, hodnota môže byť väčšia aj menšia v závislosti od toho, či hľadáme maximum alebo minimum funkcie. Teda ak má funkcia v novom bode lepšiu hodnotu, potom tretí bod hľadáme v okolí druhého. V opačnom prípade sa vrátime do prvého bodu a zvolíme nový náhodný bod v okolí prvého bodu a zároveň zmenšíme okolie od okolia pri prvom pokuse.

2.1.2 Implementácia v Matlabe

Na začiatku písania programu definujeme skúmanú funkciu, hranice na x-ovej a y-ovej osiach, v rámci ktorých vyhodnocujeme našu funkciu a začiatočný rozmer okolia *d*, v ktorom hľadáme ďalšie body. Ďalej vyberieme prvý bod *x*, potom druhý bod hľadáme v okolí prvého takým spôsobom, že si najprv vytvoríme okolie, presnejšie nájdeme ľavý dolný roh okolia *c* tak, že od súradníc centrálneho bodu odpočítame polovičnú dĺžku hrany okolia, viď rovnica 8.

$$c = \left(x_x - \frac{d}{2}\ ,\ x_y - \frac{d}{2}\right) \tag{8}$$

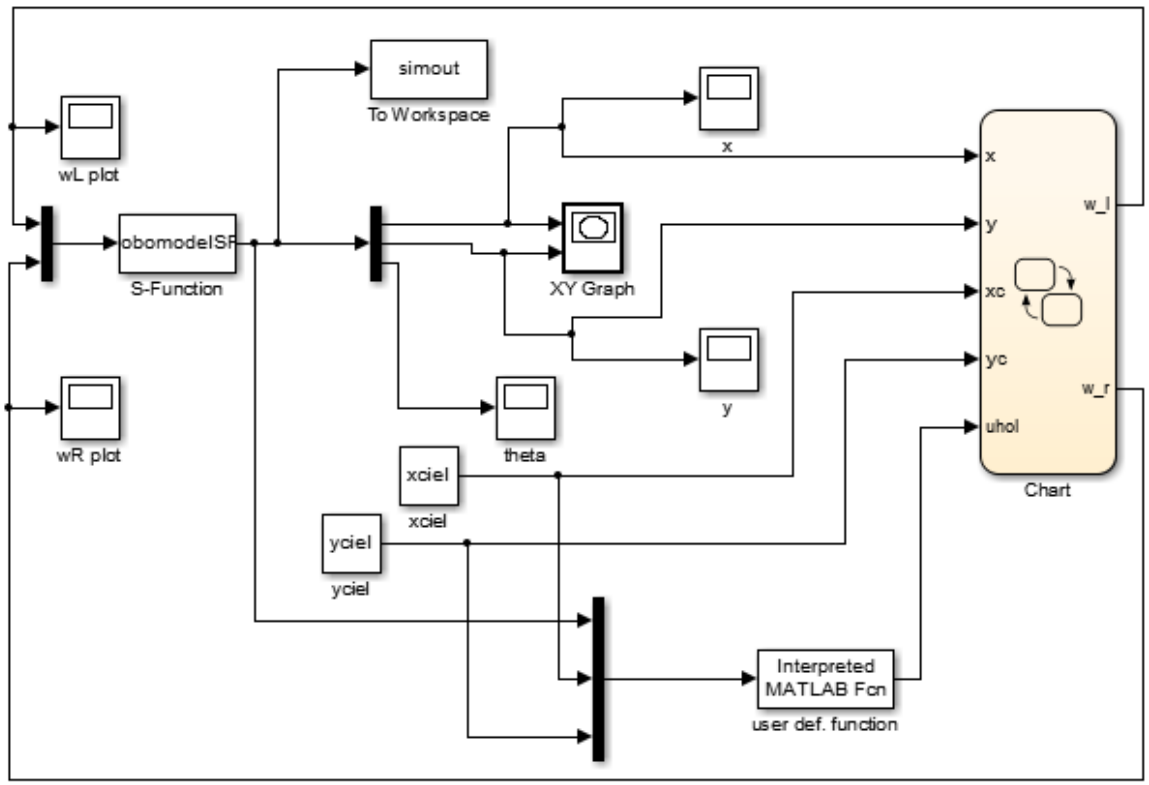
Nový bod *a* nájdeme tak, že k súradniciam ľavého dolného rohu okolia pripočítame dĺžku hrany okolia vynásobenú náhodným číslom v rozmedzí hodnôt nula až jedna (rovnica 9).

$$a = \left(c_x + rand.d,\ c_y + rand.d\right) \tag{9}$$

Uvedeným spôsobom postupujeme, pokiaľ nebude splnená podmienka na zastavenie programu. Pre náš program sú to tri podmienky: prvá – počet iterácií, druhá - rozmer okolia, tretia - konvergencia. Pri každej voľbe nového bodu sa počet iterácii zvyšuje o jeden. Ak prekročí hodnotu 1000 iterácii program sa zastaví. Pri každej nevyhovujúcej voľbe sa dĺžka hrany zmenšuje o 5 %. Ak bude hrana menšia ako 0.001, program sa ukončí. Podmienka konvergencie je, že rozdiel hodnôt funkcii v dvoch za sebou nájdených bodov je menší ako 10^{-6}.

2.1.3 Prepojenie minimalizačného a riadiaceho algoritmu

Nakoľko potrebujeme riešiť optimalizačné úlohy pomocou robotického vozidla, musíme spojiť tento minimalizačný algoritmus a schému pre ovládanie robotického vozidla vytvorenú v kapitole 1. A to tak, že program zvolí nový bod do ktorého pošleme robota, po zbehnutí simulácii dostaneme bod, v ktorom sa robot zastaví. Je jasné, že to nebude bod totožný s vypočítaným. Teda vyhodnocujeme hodnotu účelovej funkcie v bode v ktorom sa robot zastavil. Aby sme umožnili komunikáciu medzi Simulinkovou schémou a minimalizačným programom v Matlabe, potrebujeme túto schému zmodifikovať, ako je uvedené na obrázku č. 4.



Obr. 5 Simulink schéma zmodifikovaná pre komunikáciu s programom v Matlabe

Hlavná modifikácia tejto schémy je pridanie bloku *Simout*, pomocou ktorého ukladáme polohu robota v každom okamihu do premennej v programe. Posledná hodnota danej premennej je konečnou polohou robota.

2.1.4 Riešenie optimalizačných úloh

Naše optimalizačné úlohy sú minimalizáciou troch účelových funkcií uvedených na začiatku práce. Výsledky riešení optimalizačných úloh tejto metódy sú uvedené v nasledujúcej tabuľke.

Tab. 2 Výsledky riešenia optimalizačných úloh Luus-Jaakolovou metódou

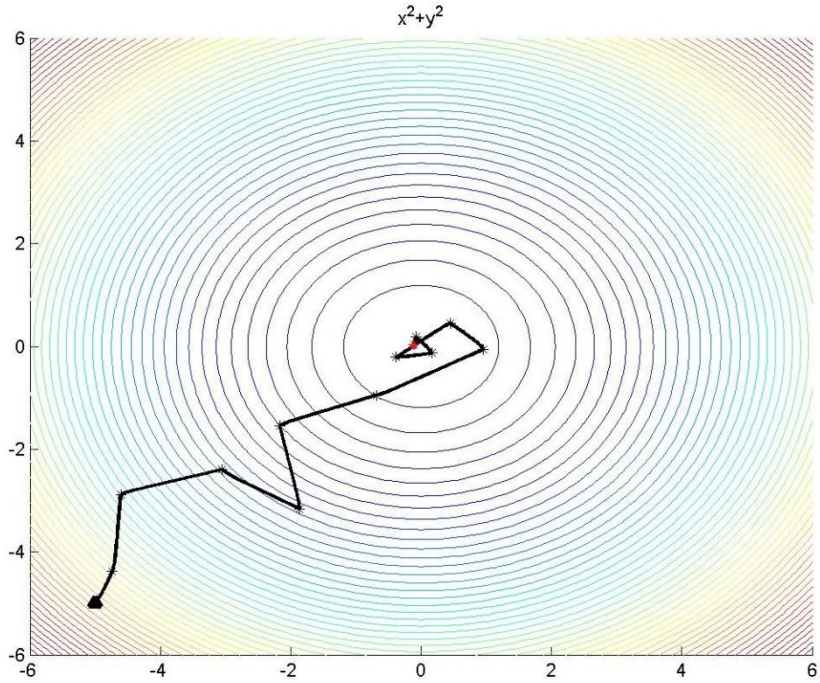
Funkcia	$[x_{opt}, y_{opt}]$	$f(x_{opt}, y_{opt})$	f_{min}	Počet iterácií
$f = x^2 + y^2$	[0.0709 0.1341]	0.0230	0.0000	28
Himmelbauho funkcia	[3.5596 -1.8240]	0.0364	0.0000	28
Funkcia Cross-in-tray	[-1.3811 -1.3634]	-2.0625	-2.0626	30

Ako vidieť z tabuľky a nasledujúcich grafov, je uvedené, ako sa mení hodnota funkcií počas jej optimalizácie. Táto metóda na nájdenie hodnoty funkcie blízkej jej minimu, potrebuje približne 15 iterácii. To je pre nás kritérium rýchlosti metódy. Ak necháme program bežať dlhšie metóda skonverguje k bodu v ktorom vyhodnotí minimum funkcii. Ak porovnáme nájdené hodnoty so skutočnými hodnotami miním vidíme, že majú rozdiel na 2 až 3 desatinnom mieste, čo je mierou presnosti metódy.

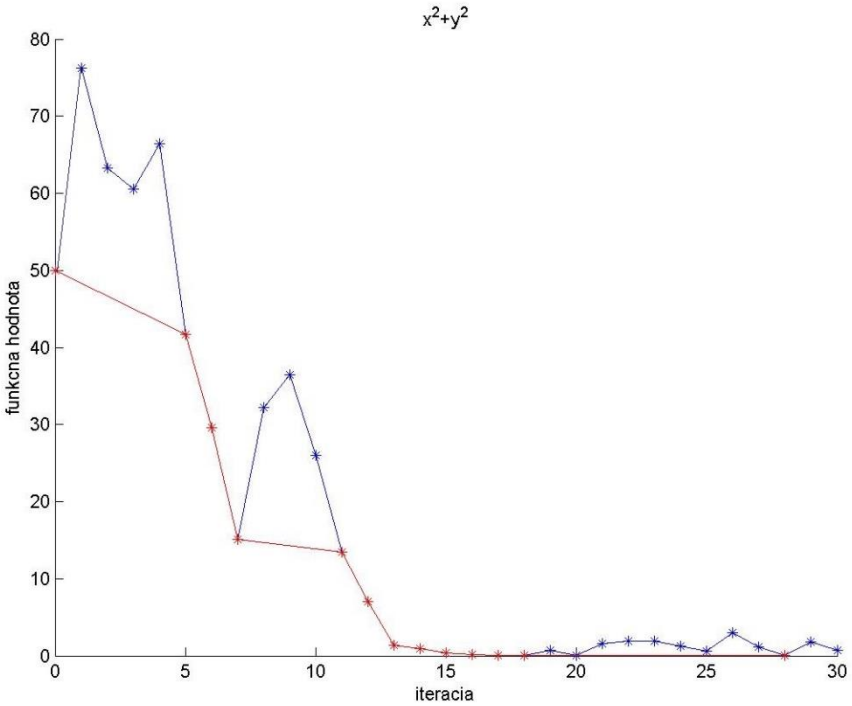
Výsledky riešenia optimalizačných úloh Luus-Jaakolovou metódou sú spracované graficky, na obrazkoch kde je uvedena trajektoria pohybu robota, červenou farbou je vyznačený bod ktorý program vyhodnotil ako minimum. Na obrázkoch pre konvergenciu danej metódy pre danú funkciu, červenou farbou sú vyznačene body v ktorých sa funkcia zmenšovala svoju hodnotu pocas behu programu.

1. **Minimalizácia funkcii $f = x^2 + y^2$**

   Na obrázku 6 je uvedená trajektória pohybu robota pri optimalizácii danej funkcie pričom je znázornený iba pohyb, ktorý vedie k zmenšovaniu hodnoty účelovej funkcie. Na obrázku č. 7 je uvedený trend poklesu hodnoty účelovej funkcii počas behu programu. Pričom červená čiara nám ukazuje body, v ktorých sa zlepšuje hodnota účelovej funkcie oproti predošlému najlepšiemu nájdenému riešeniu. Z obrázku č. 6 je vidieť, že sa funkcia priblížila k minimu v 13-tej iterácii, ale program vyhodnotil, že funkcia dosiahla svoje minimum v 28-tej iterácii.



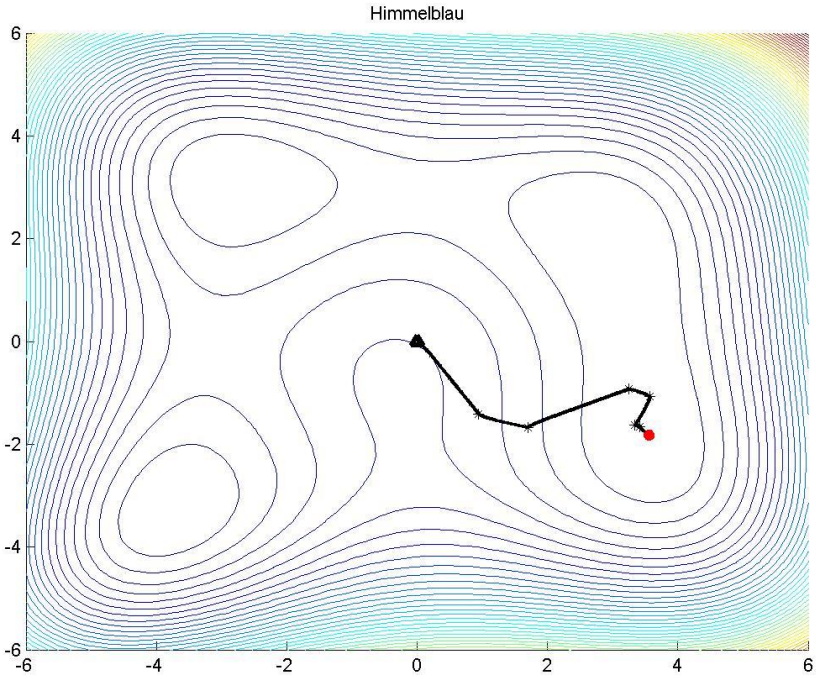
Obr. 6 Trajektória pohybu robota pri Luus-Jaakolovej metóde pre funkciu č. 1



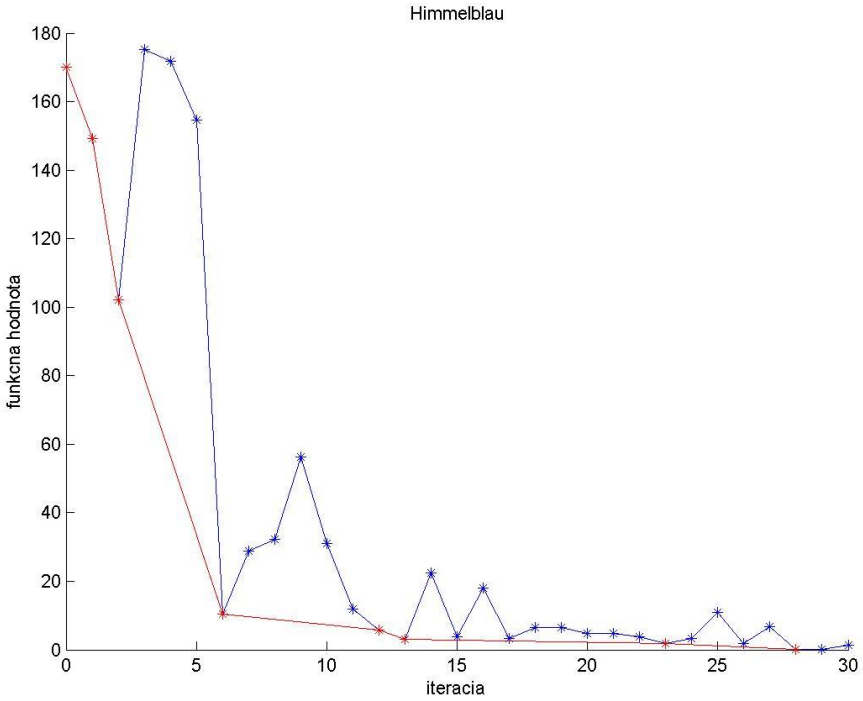
Obr. 7 Konvergencia Luus-Jaakolovej metódy pre funkciu č. 1

2. Minimalizácia Himmelblauho funkcie

Výsledky minimalizácie Himmelblauho funkcie Luus-Jaakolovou metódou sú spracované rovnakým spôsobom ako pri minimalizácii funkcii 1 a sú znázornene na obrázku č. 7 Trajektória pohybu robota a na obrázku č. 8 Konvergencia funkcie. Na obrázku č. 8 vidíme, že sa funkcia priblížila k svojmu minimu už v 13 iterácii, ale pokračovala sa zlepšovať až do 28 iterácie.



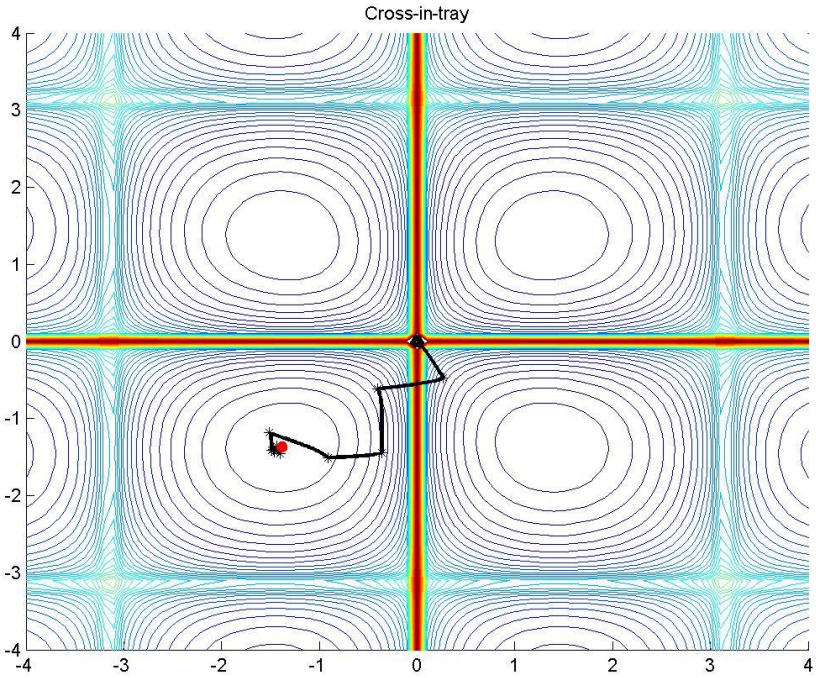
Obr 8 Trajektória pohybu robota pri Luus-Jaakolovej metóde pre funkciu č. 2



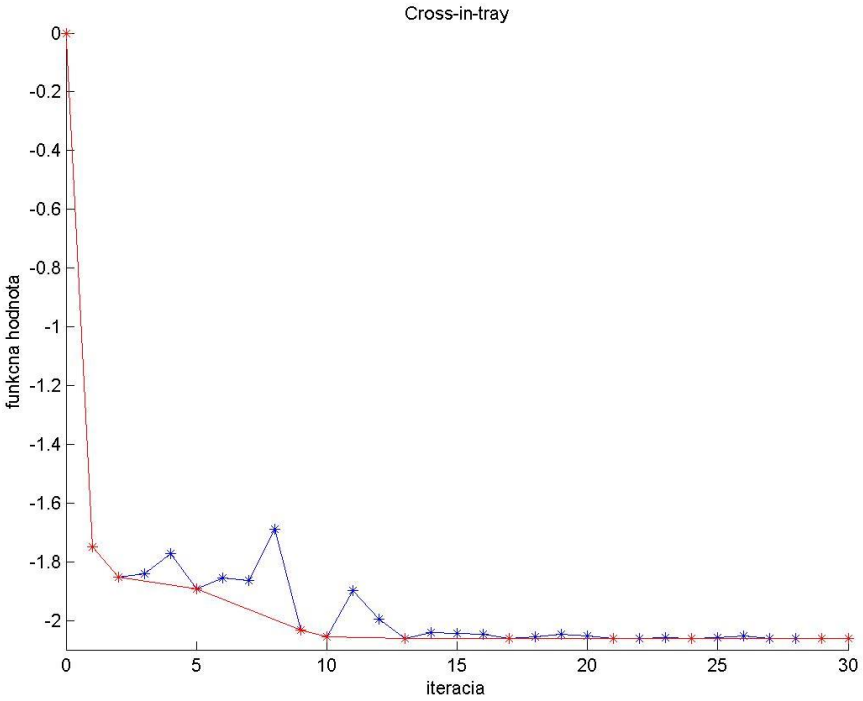
Obr. 9 Konvergencia Luus-Jaakolovej metódy pre funkciu č. 2

3. Minimalizácia funkcie Cross-in-tray

Spracovanie funkcie Cross-in-tray prebieha rovnakým spôsobom ako spracovanie dvoch predošlých funkcií. Dostávame obrázok č. 9, kde je uvedená trajektória pohybu robota a obrázok č. 10 s konvergenciou funkcie. Z tohto obrázku vieme odčítať, že táto metóda skonvergovala k minimu v iterácii 10 a že dosiahla minimum v iterácii 30.



Obr. 10 Trajektória pohybu robota pri Luus-Jaakolovej metóde pre funkciu č. 3



Obr. č. 11: Konvergencia Luus-Jaakolovej metódy pre funkciu č. 3

2.2 Nelder-Meadova optimalizačná metóda

Druhá optimalizačná metóda použitá v tejto práci je metóda Neldera-Meada. Pri tejto metóde, na rozdiel od predošlej, používame troch robotov. Označíme ich robot A, robot B a robot C.

2.2.1 Minimalizačný algoritmus

V tomto prípade funkciu optimalizujeme spôsobom, že naraz odčítame funkčné hodnoty z troch robotov. Následne posúvame robota, ktorý udáva najhoršiu hodnotu účelovej funkcie. Toto je základná logika Nelder-Meadovej optimalizačnej metódy. Jej realizácia je nasledovná: v prvom kroku umiestnime robotov, aby boli vrcholmi približne rovnostranného trojuholníka. Potom posunutím robota s najhoršou hodnotou účelovej funcie posunieme cez protiľahlú stranu trojuholníka a vytvoríme zrkadlový obraz. Teda robot A prejde cez stranu vytvorenú robotom B a robotom C. Na nájdenie súradníc bodu do ktorého má prejsť robot používame nasledujúci vzorec:

$$R = 2M - A \tag{10}$$

kde R je cieľový bod, M je stred na hrane, ktorú vytvárajú dva roboty, ktorých zatiaľ neposúvame a A je bod, ktorý chceme presunúť. Všetky tieto body sú vektory ktoré obsahujú x-ovú a y-ovú súradnicu príslušného bodu.

2.2.2 Implementácia v Matlabe

Na začiatku programu definujeme účelovú funkciu, jej hranice a umiestnenie robotov. Potom pomocou jednoduchého porovnania nájdeme robota, ktorý udáva najhoršiu hodnotu účelovej funkcie a aplikujeme naň uvedený vzorec. V prípade, že v novom bode má funkcia lepšiu hodnotu, robota necháme tam a znova vyhodnotíme všetkých troch robotov. V opačnom prípade robota vrátime naspäť a hľadáme nový bod o niečo bližšie ku protiľahlej hrane. Aby sme to mohli zabezpečiť používame modifikovaný vzorec č. 11 na nájdenie nového bodu:

$$R = (k + 1)M - k * A \tag{11}$$

Aby sme mohli zmenšovať vzdialenosť, ktorú má prejsť robot, budeme pri neúspešnom pokuse zmenšovať parameter *k*. Na začiatku sa parameter *k* rovná jednej, postupne sa zmenšuje až na hodnotu blízku nule. Na dosiahnutie robotom žiadaného bodu používame simulačnú schému vytvorenú na začiatku práce. Do nej inicializujeme začiatočnú a cieľovú polohu robota a spúšťame simuláciu. Je zrejmé, že robot nedôjde presne do cieľového bodu a preto nám vznikne určitá nepresnosť. Avšak tým sa približujeme k správaniu reálneho robota v reálnom prostredí. Podmienky ukončenia programu sú maximálny počet iterácii povolený pre program a hodnota parametru *k* – program sa ukončí ak *k* je menšie ako 10^{-3}. Na konci programu porovnáme hodnoty účelovej funkcie, ktoré udávajú polohy robotov a najmenšia z nich je našim riešením optimalizačného problému.

2.2.3 Prepojenie minimalizačného a riadiaceho algoritmu

Prepojenie minimalizačného a riadiaceho algoritmu uskutočňujeme rovnakým spôsobom ako pri Luus-Jaakolovej metóde. Podrobný popis je v kapitole 2.2.3

2.2.4 Riešenie optimalizačných úloh

Výsledky optimalizácie sú uvedené v nasledujúcej tabuľke.

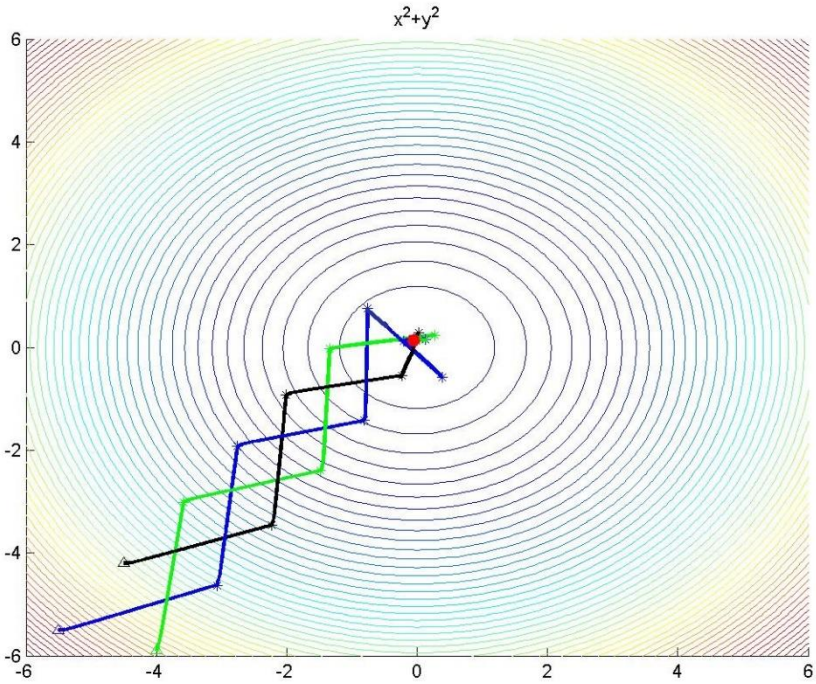
Tab. 3 Výsledky riešenia optimalizačných úloh Nelder-Meadovou metódou

Funkcia	$[x_{opt}, y_{opt}]$	$f([x_{opt}, y_{opt}])$	f_{min}	Počet iterácii
$f = x^2 + y^2$	[-0.0519 0.1340]	0.0206	0.0000	26
Himmelbauho funkcia	[-2.9228 3.1182]	0.4768	0.0000	17
Funkcia Cross-in-tray	[1.3393 1.3949]	-2.0624	-2.0626	19

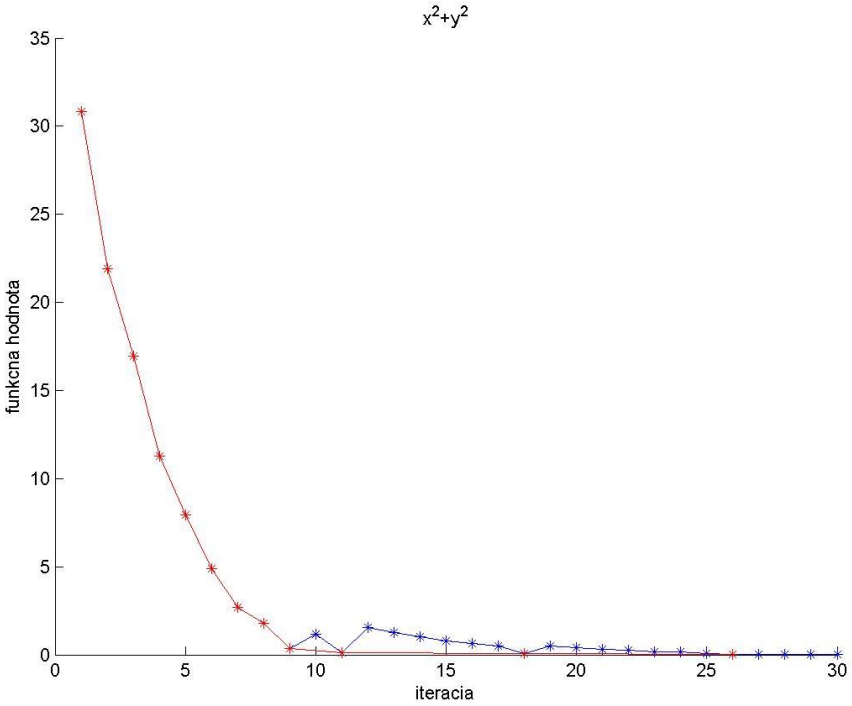
Výsledky minimalizácie Nelder-Meadovou optimalizačnou metódou spracovávame graficky. Pre každú funkciu sú uvedené na obrázkoch č. 12-16. Modrou farbou je vyznačená trajektória pohybu robota A, zelenou robota B a čiernou robota C. Trojuholníkmi je vyznačená začiatočná poloha robotov, červený bod je bod, ktorý program vyhodnotil ako minimum.

1. Minimalizácia funkcii $f = x^2+y^2$

Na obrázku č. 12 je uvedená trajektória pohybu robotov. Na obrázku č. 13 je znázornená konvergencia funkčnej hodnoty k svojmu minimu pri jej optimalizácii Nelder-Meadovou optimalizačnou metódou. Z tohto obrázku vieme odčítať pre nás zaujímavú hodnotu. Je to bod, v ktorom funkčná hodnota prestáva prudko klesať. To znamená, že už je blízko svojho minima. Táto skutočnosť nastane v iterácii číslo 9. Zároveň vidíme, že program pokračuje v minimalizácií funkcie a povie, že nevie nájsť hodnotu funkcie ktorá by bola menšia, ako hodnota nájdená v iterácii číslo 21.



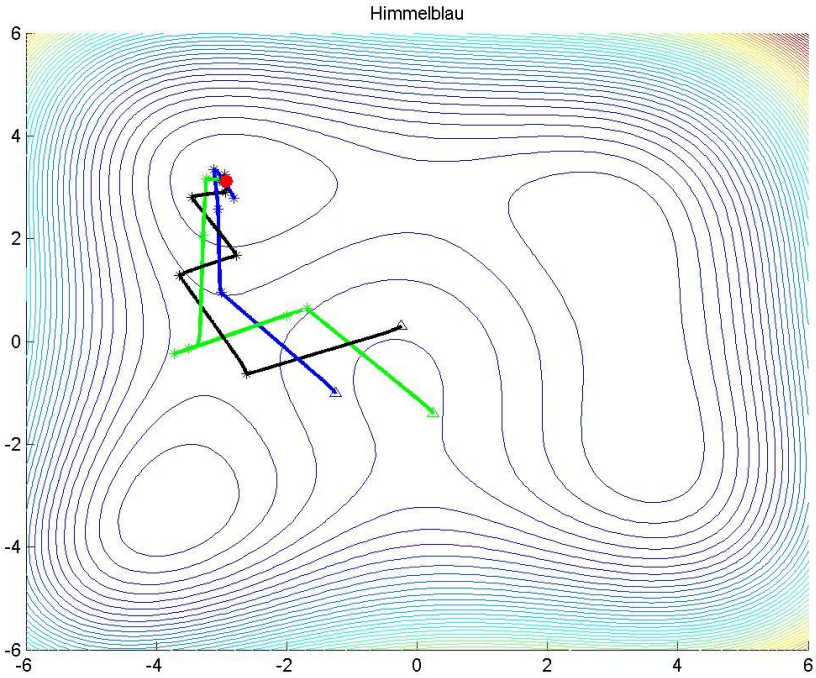
Obr. 12 Trajektória pohybu robota pri Nelder-Meadovej metóde pre funkciu č. 1



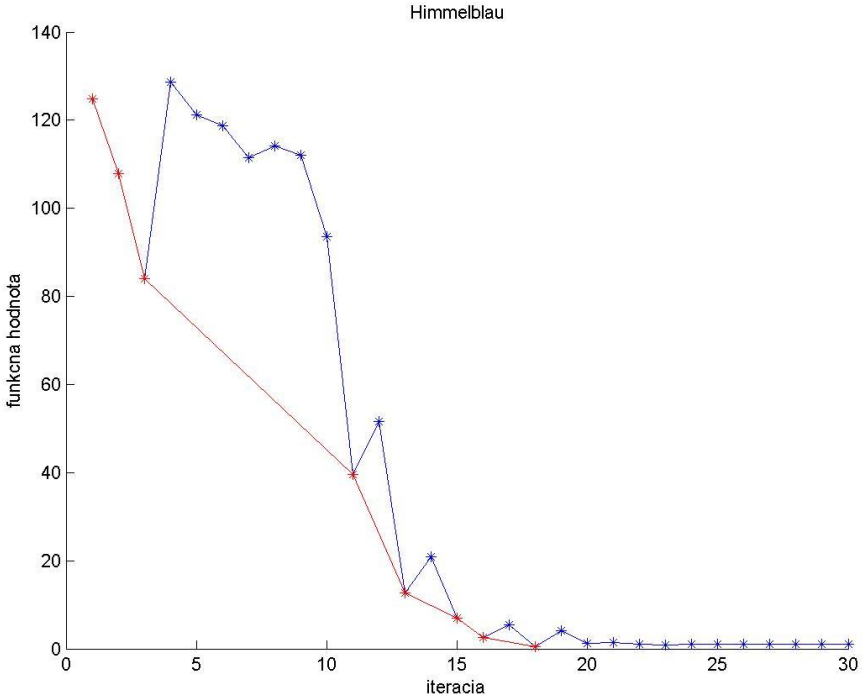
Obr. 13 Konvergencia Nelder-Meadovej metódy pre funkciu č. 1

2. Minimalizácia Himmelblauho funkcii

Výsledky optimalizácie Himmelblauho funkcie Nelder-Meadovou optimalizačnou metódou sú uvedené na obrázku č. 14 a obrázku č. 15. Z obrázku č. 15 je vidieť, že táto metóda optimalizuje funkciu číslo 2 pomerne rýchlo. Už v 16 iterácii je hodnota účelovej funkcii už blízko svojho minima, ale najnižšiu možnú hodnotu účelovej funkcie program nájde v 17 iterácii.



Obr. 13 Trajektória pohybu robota pri Nelder-Meadovej metóde pre funkciu č

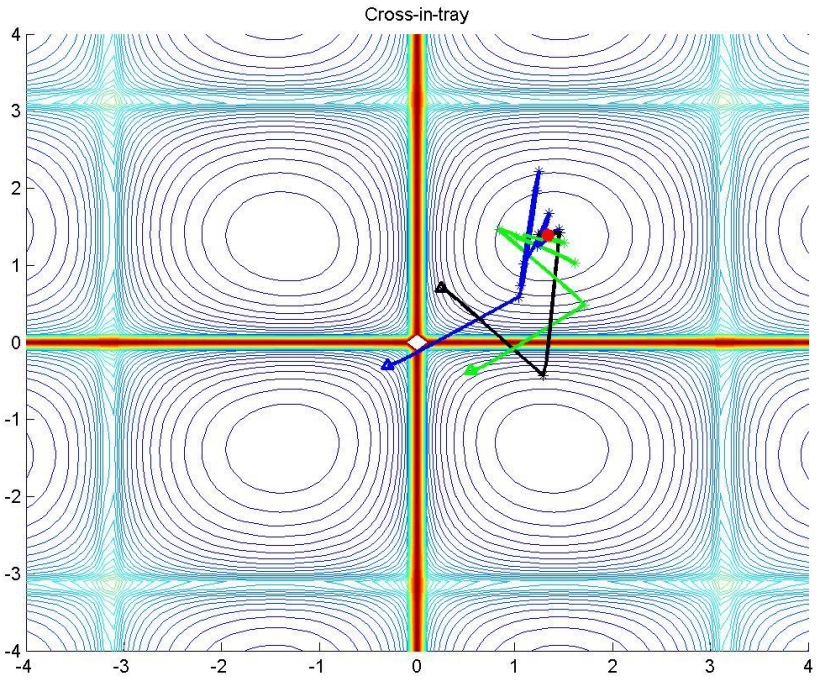


Obr. 14 Konvergencia Nelder-Meadovej metódy pre funkciu č. 2

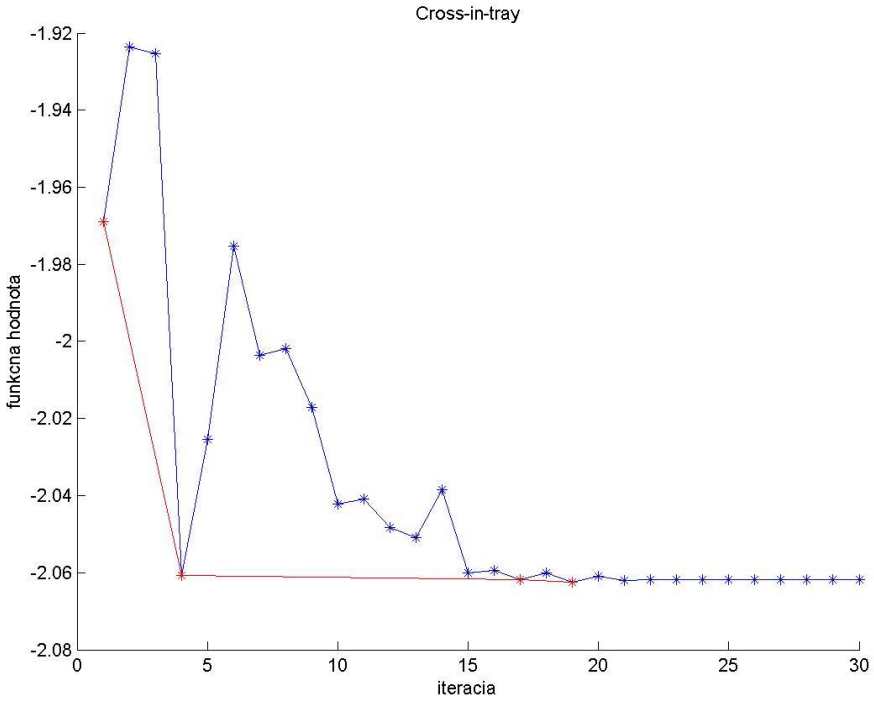
3. Minimalizácia funkcii Cross-in-tray

Priebeh minimalizácie Cross-in-tray funkcie je uvedený na obrázku č. 16 a obrázku č. 17.

Pre nás je zaujímavý obrázok č. 17, pomocou ktorého vieme posúdiť rýchlosť minimalizácie funkcie touto metódou. Vidíme, že hodnotu účelovej funkcie blízku jej minimu program nájde v 4-tej iterácii a hodnotu maximálne blízku minimu nájde v 19-tej iterácii.



Obr. 15 Trajektória pohybu robota pri Nelder-Meadovej metóde pre funkciu č.3



Obr. 16 Konvergencia Nelder-Meadovej metódy pre funkciu č. 3

2.3 Metóda optimalizácie rojom častíc

Tretia optimalizačná metóda uvedená v tejto práci je metóda optimalizácie rojom častíc. Pri danej metóde používame omnoho viac robotov, ktoré sa pohybujú naraz.

2.3.1 Minimalizačný algoritmus

Táto metóda optimalizuje funkciu tým, že roboty pri svojom pohybe medzi sebou komunikujú. Roboty sa optimalizáciou zhromažďujú v jednom bode, ktorý je našim optimom.

2.3.2 Implementácia v Matlabe

Programová realizácia tejto úlohy je nasledovná. Najprv si vytvoríme roj robotov, ktoré sú rovnomerne rozložené po celej skúmanej ploche. Tým vytvoríme maticu polôh robotov. Každý riadok reprezentuje jedného robota a to jeho x-ovú a y-ovú súradnicu. Je zrejmé, že počet

riadkov tejto matice bude rovný počtu robotov v roji. Hneď si vytvoríme maticu, do ktorej budeme zapisovať najlepšiu nájdenú polohu pre každého robota v roji. Na začiatku bude matica totožná s maticou aktuálnych polôh robotov. Ďalej si preskúmame každého robota a vyhodnotíme globálne najlepšiu polohu pre roj. Tá bude daná robotom, ktorého poloha udáva najlepšiu hodnotu účelovej funkcie. V ďalšom kroku začneme robotmi pohybovať. Teda pre každého robota budeme počítať novú polohu podľa vzorca:

$$p(k+1) = i + r_1\big(g - p(k)\big) + r_2\big(b - p(k)\big) \tag{12}$$

kde *p(k+1)* je poloha v novej iteracii, r_1 je náhodne číslo z intervalu [-1, 1], r_2 je náhodné číslo z intervalu [0, 2], g je najlepšia doteraz známa poloha celého roja, b je doteraz najlepšia známa poloha pre daného robota a *p(k)* poloha robota v terajšej iteracii. Po výpočte novej polohy robota ju vyhodnotíme a porovnáme s doteraz známymi najlepšími polohami pre robota a celý roj a ak je nová poloha lepšia, prepíšeme personálnu najlepšiu polohu alebo polohu roja.

2.3.3 Prepojenie minimalizačného a riadiaceho algoritmu

Na rozdiel od predošlých metód, ak pri vyhodnotení zistíme, že v novom bode má funkcia horšiu hodnotu účelovej funkcie ako v predošlom bode, nevraciame robota do pôvodnej polohy, ale hľadáme ďalší bod podľa uvedeného vzorca. Zároveň túto metódu realizujeme iba pomocou matematických výpočtov, bez použitia simulačnej schémy pre pohyb robota, nakoľko pomocou našej schémy vieme zabezpečiť pohyb iba jedného robota. Program nepokračuje ďalej, pokiaľ celá simulácia nezbehne. Pri optimalizácii rojom častíc potrebujeme, aby sa v jednom kroku premiestnili všetky roboty naraz.

2.3.4 Riešenie optimalizačných úloh

Výsledky optimalizácie metódou roja častíc sú spracované vo forme tabuľky.

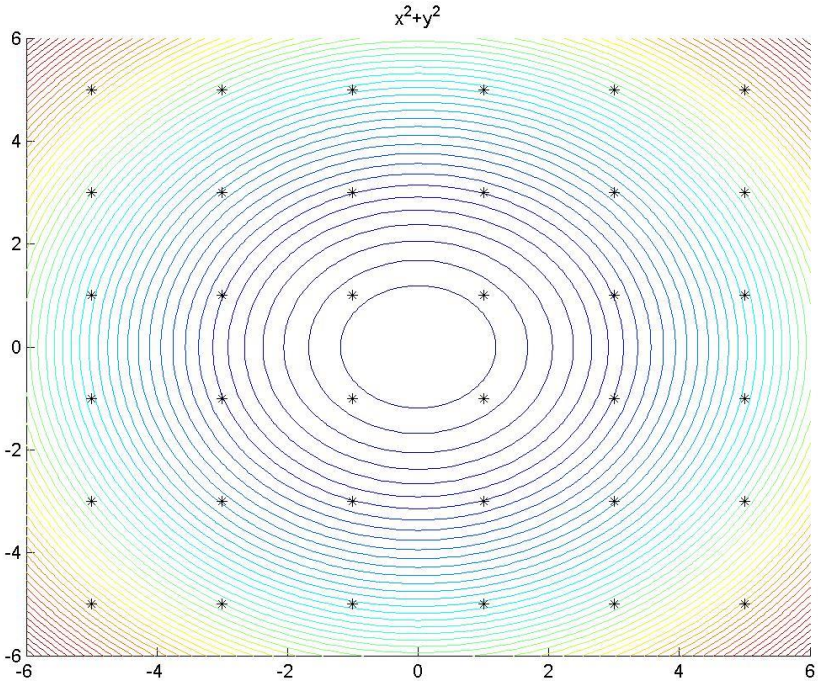
Tab. 4 Výsledky riešenia optimalizačných úloh metódou optimalizácie rojom častíc

Funkcia	$[x_{opt}, y_{opt}]$	$f([x_{opt}, y_{opt}])$	f_{min}	Počet iterácií
$f = x^2 + y^2$	$[1{,}62.10^{-5}\ \ 0.0002]$	$2{,}9417.10^{-10}$	0.0000	19
Himmelbauho funkcia	[3.0002 2.0056]	0.0006	0.0000	19
Funkcia Cross-in-tray	[-1.3480 -1.3516]	-2.0626	-2.0626	20

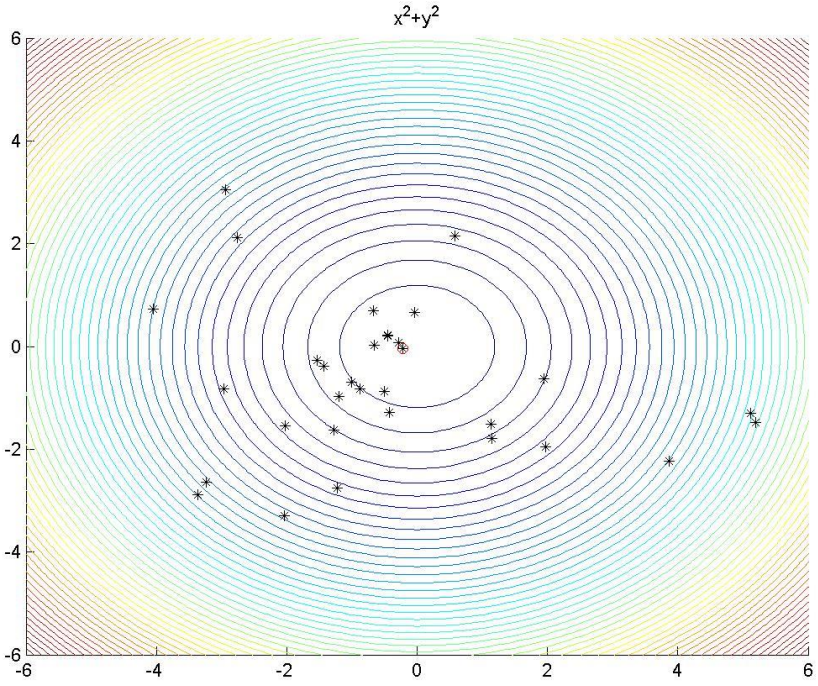
Pri grafickom ukazovaní fungovania optimalizačnej metódy používame trajektóriu pohybu robotov a trend zmeny hodnoty globálneho minima počas behu programu. Pri tejto optimalizačnej metóde je ťažko ukázať trajektórie pohybu všetkých robotov v dôsledku veľkého počtu robotov a veľkého počtu nimi prejdených úsekov. Preto pre znázornenie trajektórie pohybu robotov používame 4 za sebou nasledujúce obrázky, na ktorých je uvedené rozloženie všetkých robotov roja v nultej, tretej, piatej a dvadsiatej iterácii.

1. Minimalizácia funkcii $x^2 + y^2$

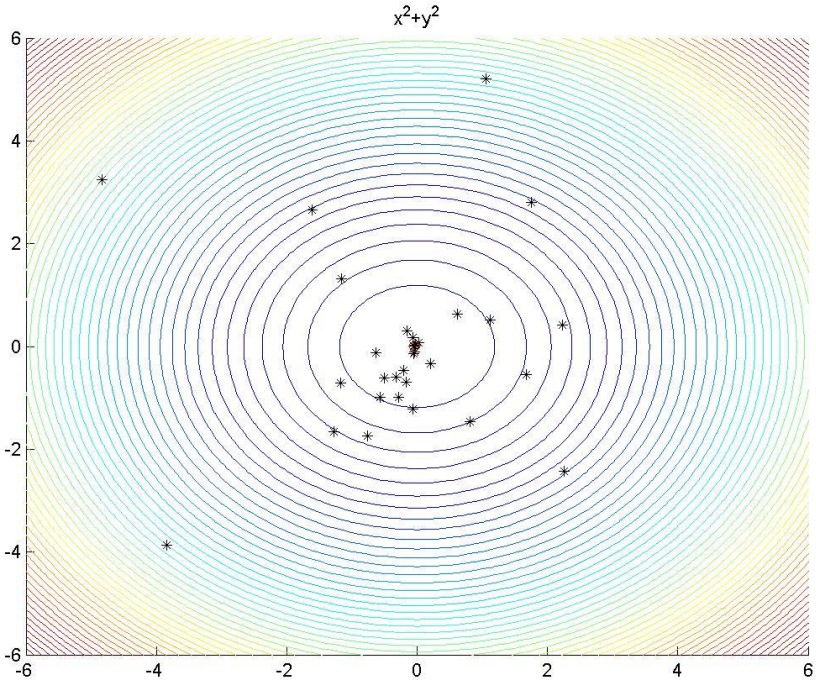
Na obrázkoch č. 18 až 22 je uvedené, ako sa mení poloha robotov v roji pri minimalizácii funkcie. Z obrázku č. 22 vidíme, že už v 3-tej iterácii nájde táto metóda hodnotu účelovej funkcie blízku jej minimu, ale hodnota funkcie ktorá sa najviac približuje k minimu, je nájdená v 19-tej iterácii.



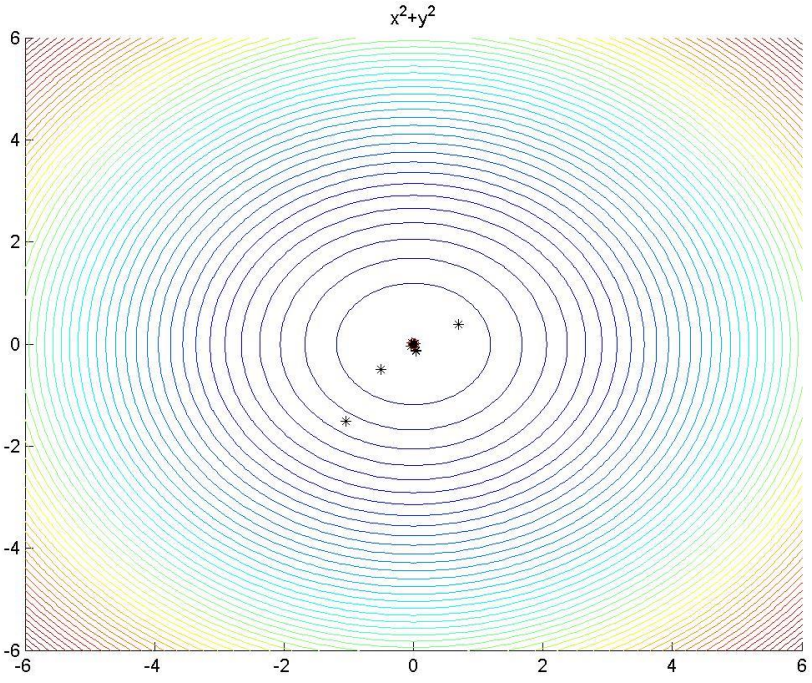
Obr. 17 Rozloženie roja v iterácii 0 pri optimalizácii funkcie č. 1



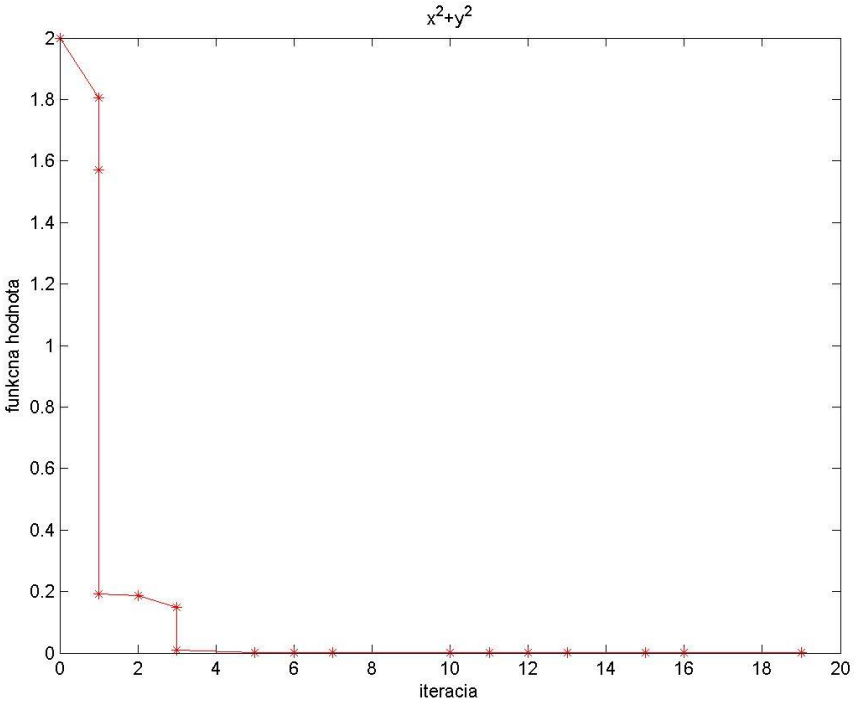
Obr. 18 Rozloženie roja v iterácii 3 pri optimalizácii funkcie č. 1



Obr. 19 Rozloženie roja v iterácii 5 pri optimalizácii funkciu č. 1



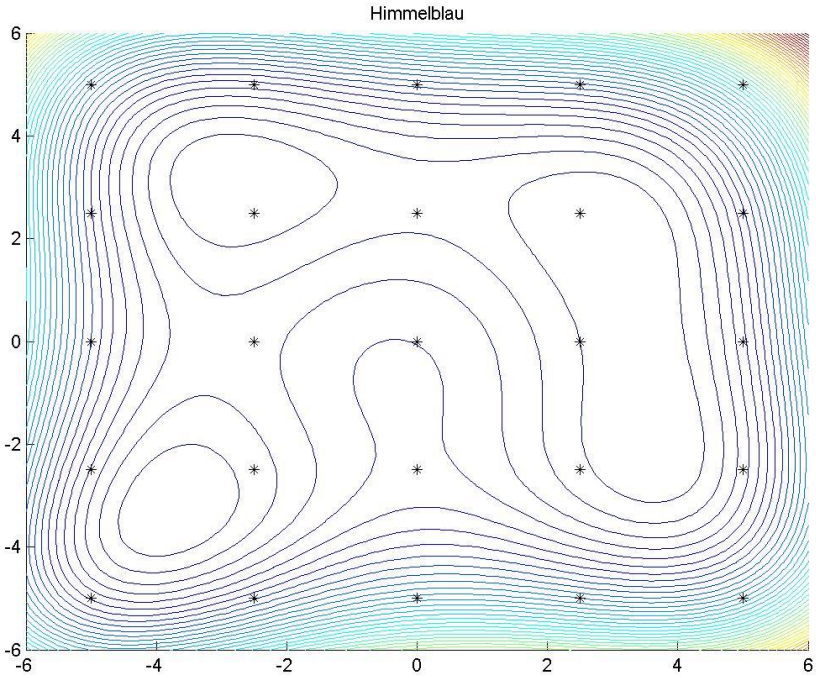
Obr. 20 Rozloženie roja v iterácii 20 pri optimalizácii funkcie č. 1



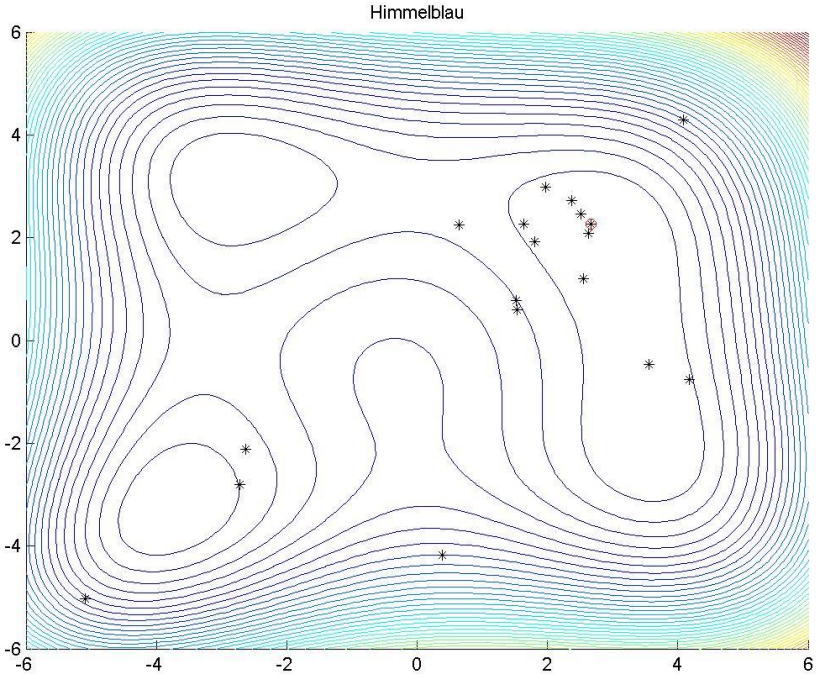
Obr 21 Konvergencia metódy optimalizácie rojom častíc pre funkciu č. 1

2. Minimalizácia Himmelblauho funkcii

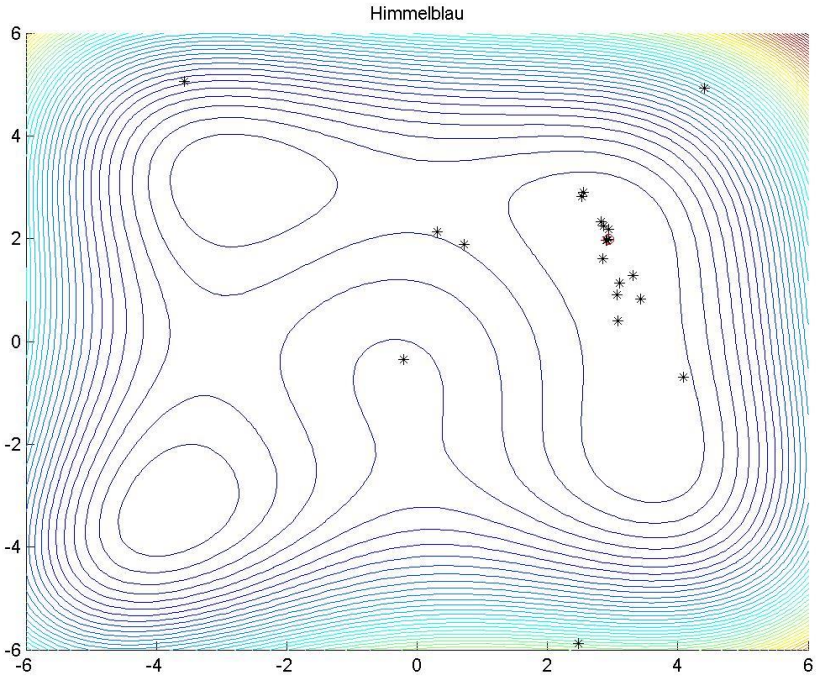
Na obrázkoch č. 23 až 26 je uvedená zmena rozloženia roja počas minimalizácie Himmelbllauho funkcie. Z obrázku pre trend zmeny hodnoty nájdeného globálneho minima vieme zistiť, že program sa priblíži k nájdeniu minima funkcie už v 4-tej iterácii a maximálne sa priblíži k globálnemu minimu funkcie v 19-tej iterácii.



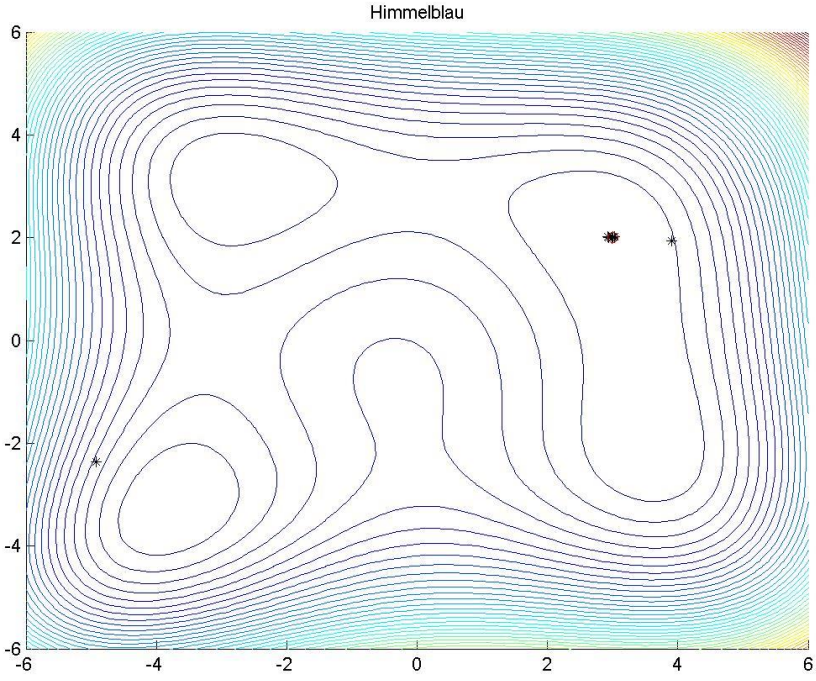
Obr. 22 Rozloženie roja v iterácii 0 pri optimalizácii funkcie č. 2



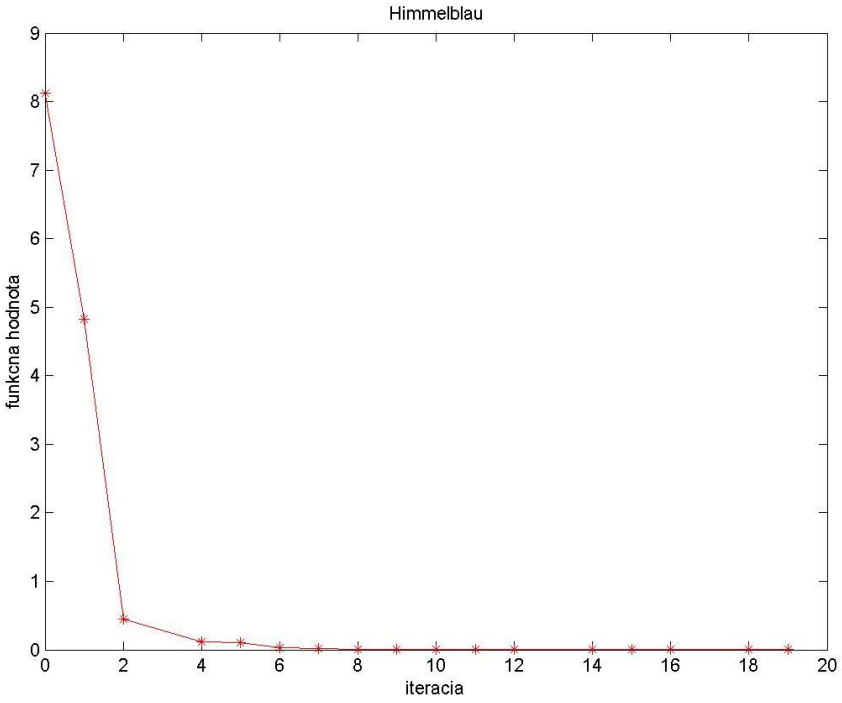
Obr. 23: Rozloženie roja v iterácii 3 pri optimalizácii funkcie č. 2



Obr. 24 Rozloženie roja v iterácii 5 pri optimalizácii funkcie č. 2



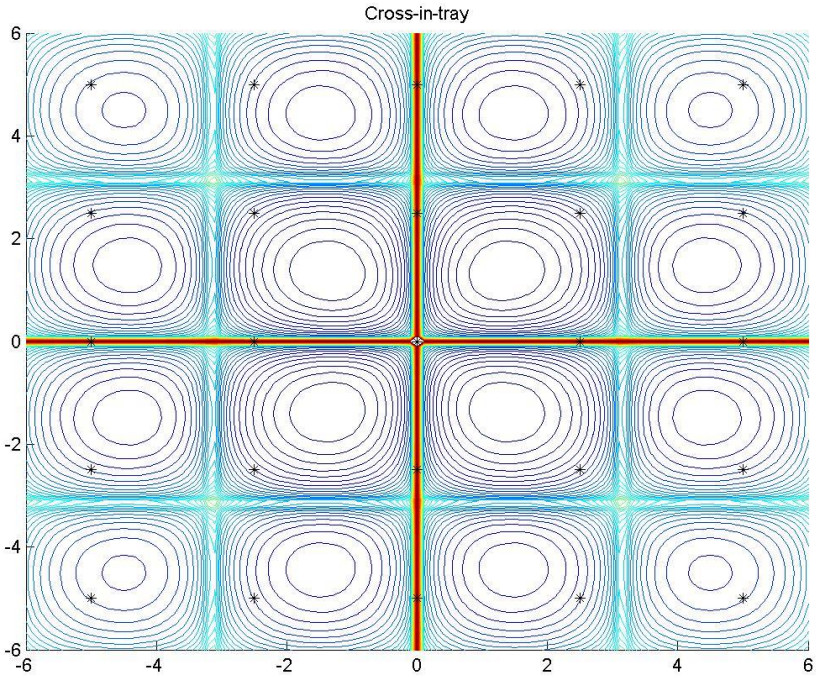
Obr. 25 Rozloženie roja v iterácii 20 pri optimalizácii funkcie č. 2



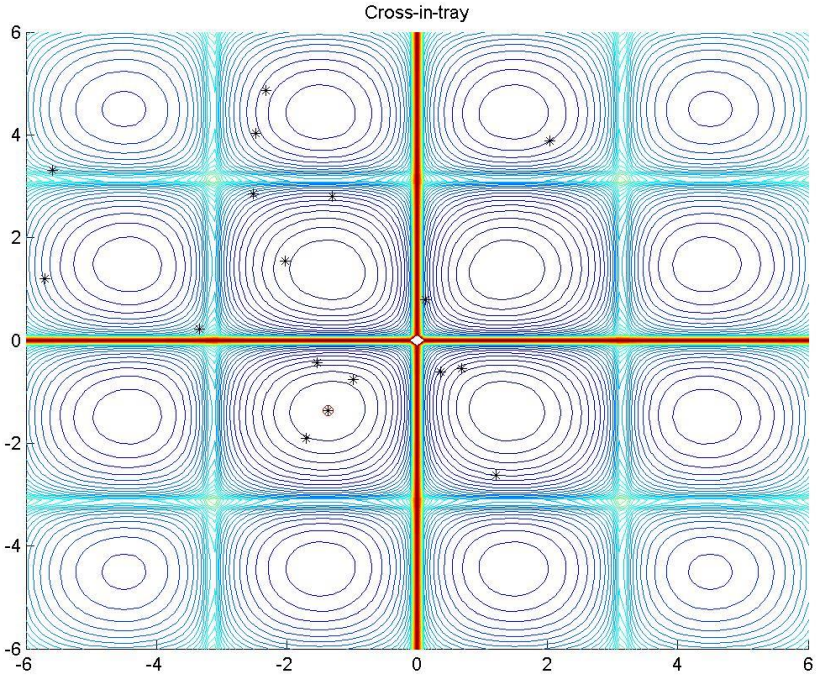
Obr 26 Konvergencia metódy optimalizácie rojom častíc pre funkciu č. 2

3. Minimalizácia funkcii Cross-in-tray

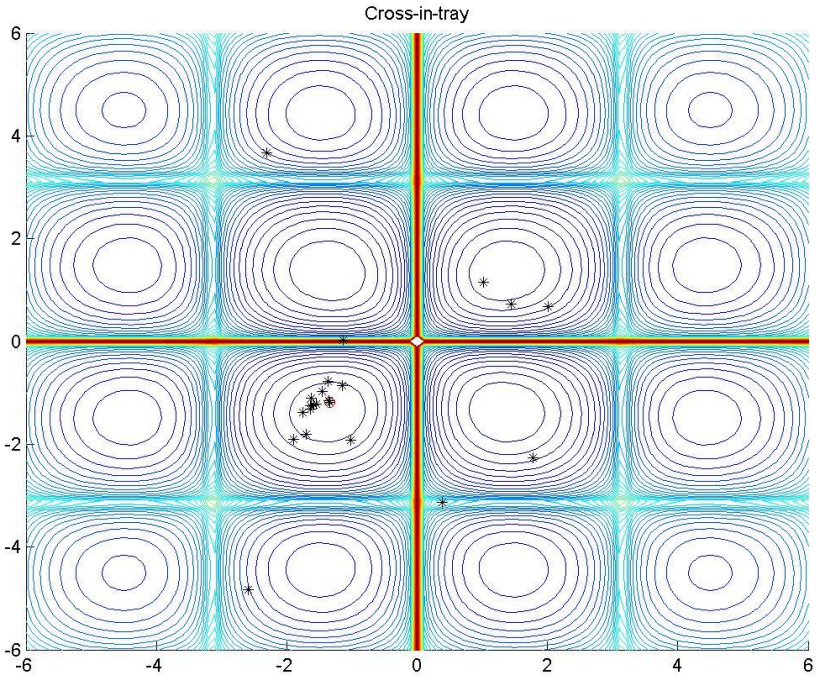
Grafické výsledky získané pri optimalizácii tejto funkcie spracovávame rovnakým spôsobom ako na obrázkoch 28 až 31. Na nich je uvedená zmena rozloženia robotov v roji počas minimalizácie funkcie. Na obrázku 32 je znázornený trend zmeny hodnoty nájdeného minima funkcie. Na ňom vidíme, že program sa priblíži k nájdeniu minima v 3-tej iterácii a najviac sa priblíži v 20-tej iterácii.



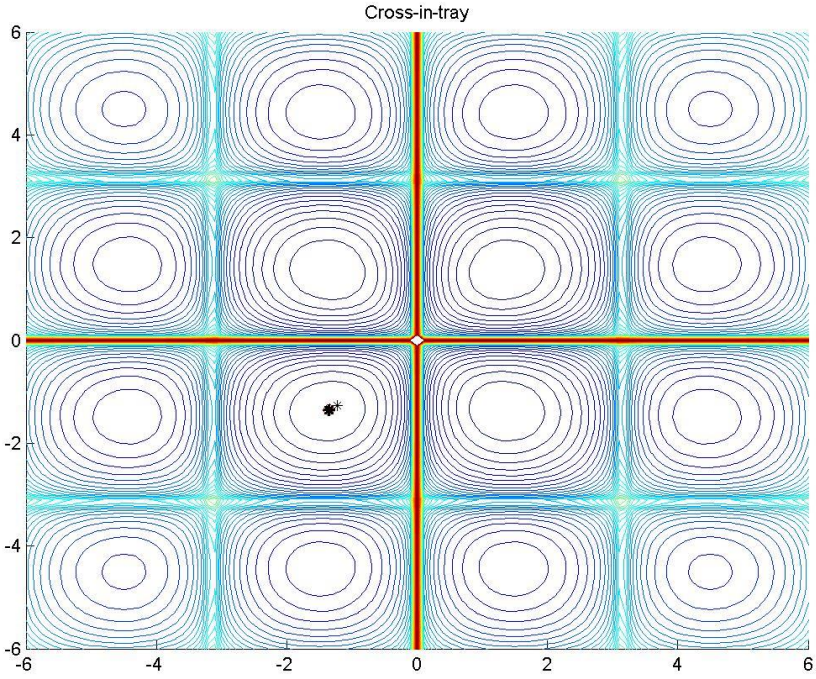
Obr. 27 Rozloženie roja v iterácii 0 pri optimalizácii funkcie č. 3



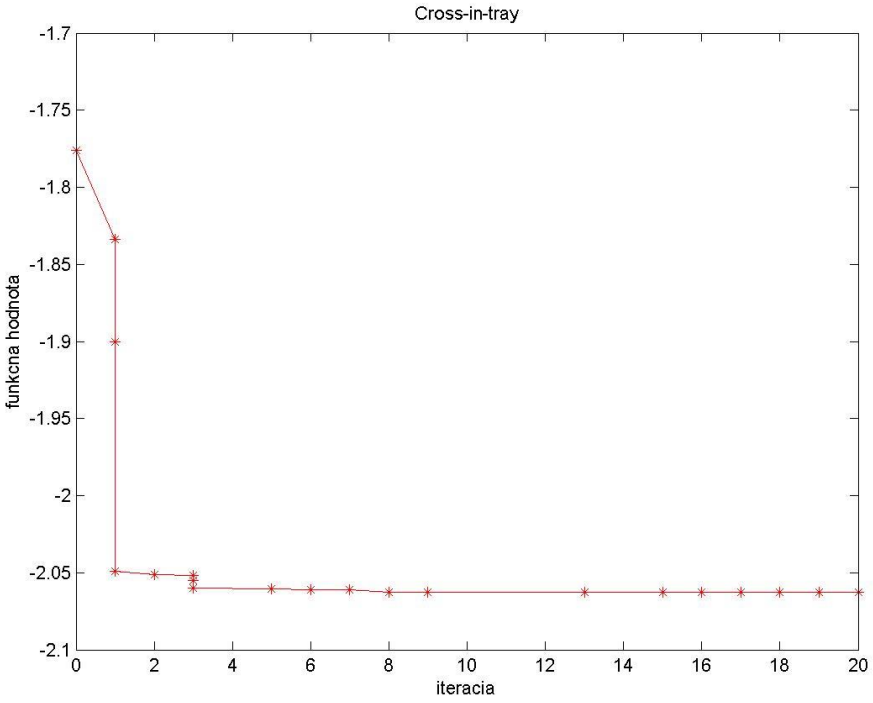
Obr. 28 Rozloženie roja v iterácii 3 pri optimalizácii funkcie č. 3



Obr. 29 Rozloženie roja v iterácii 5 pri optimalizácii funkcie č. 3



Obr. 30 Rozloženie roja v iterácii 20 pri optimalizácii funkcie č. 3



Obr. 31 Konvergencia metódy optimalizácie rojom častíc pre funkciu č. 3

Diskusia

V tejto práci sme riešili optimalizačne úlohy, ktorých cieľom bolo hľadanie minima pre tri rôzne funkcie: suma štvorcov dvoch premenných, Himmelblauho a Cross-in tray. Nájdenie aspoň jedného minima funkcie znamenalo úspešné riešenie optimalizačnej úlohy. Na riešenie týchto úloh sme použili tri bezgradientové optimalizačné metódy: Luus-Jaakolovu, Nelder-Meadovu a metódu optimalizácie rojom častíc. Pre každú metódu v Matlabe bol zostrojený osobitný program, ktorý pomocou jednoduchých matematických operácii hľadal minimum optimalizovanej funkcie.

Výsledky týchto programov sú spracované vo forme tabuliek, kde sa udáva x-ová a y-ová súradnica nájdeného bodu, hodnota optimalizovanej funkcie v danom bode a iterácia v ktorej bol daný bod nájdený. Taktiež sú výsledky z programov spracované aj graficky. Grafy znázorňujú závislosť funkčnej hodnoty optimalizovanej funkcie nájdenú v medzikroku od iterácii. Tieto grafy používame na vyhodnotenie užitočnosti danej metódy voči ostatným.

Postupne zhodnotíme každú optimalizačnú metódu. Najprv sme vytvorili program a preskúmali Luus-Jaakolovu metódu. Z grafických výsledkov je vidieť, že táto metóda na nájdenie minima optimalizovanej funkcie potrebuje minimálne 13 až 15 iterácii a minimum nájde s presnosťou na 1 až 3 desatinne miesto.

Druhou metódou bola metóda Neldera-Meada. Pri tejto metóde program nájde minimum s presnosťou na 1 až 3 desatinne miesto a potrebuje 4 až 7 iterácii.

Tretia metóda optimalizácie rojom častíc nájde minimum najrýchlejšie, konkrétne za 3 iterácie s presnosťou na 4 desatinné miesta.

Je zrejmé, že presnosť a rýchlosť nájdenia bodu v ktorom má funkcia svoju hodnotu minimálnu závisí od počiatočného nástrelu. Kam umiestnime robota v nultej iterácii, od tvaru funkcie, čím je funkcia zložitejšia, tým dlhšie môže trvať nájdenie minima. Takisto závisí od rozmerov skúmanej plochy. Avšak plyv rozmeru plochy vieme odstrániť, ak použijeme vhodné podmienky pre pohyb robota. Pri Luus-Jaakolovej metóde použijeme veľké okolie na hľadanie

nového bodu v okolí prvého, pre Nelder-Meadovu metódu použijeme rozloženie robotov s veľkou vzájomnou vzdialenosťou, pre metódu optimalizácie rojom častíc je určujúcou podmienkou hustota rozloženia robotov. Pre väčšiu plochu použijeme väčší roj.

Záver

Cieľmi tejto práce boli zostrojenie programov na optimalizáciu, rovnako aj preskúmanie a vyhodnotenie vhodnosti optimalizačných metód pre optimalizáciu rôznych funkcií. Z grafických výsledkov vyplýva, že najlepšia metóda je metóda optimalizácie rojom častíc. Hľadá riešenie rýchlejšie a presnejšie ako metódy Luus-Jaakolova a Nelder-Meadova. Zároveň pri optimalizácii rojom častíc nepoužívame simulačnú schému riadenia robotického vozidla a preto nám program beží najrýchlejšie. Táto metóda vyžaduje pohyb každého jedného robota v roji naraz, avšak my vieme pohybovať robotmi iba postupne ako je to potrebné pri optimalizačných metódach Luus-Jaakolovej a Nelder-Meadovej. Keby sme použili rovnakú simulinkovu schému aj pri metóde optimalizácie rojom častíc, trvalo by vyhodnotenie oveľa dlhšie v reálnom čase. V takomto prípade za najlepšiu považujeme metódu Nelder-Meada, ktorej hlavnou nevýhodou je, že nie je vhodná na optimalizáciu funkcií, ktoré majú husté rozloženie a veľké množstvo lokálnych miním.

Zatiaľ boli všetky skúmané funkcie realizované iba virtuálne, teda ostáva priestor na pokračovanie v tejto práci. Teória môže byť realizovaná pomocou reálneho zariadenia tak, že namiesto matematického modelu robota použijeme reálne robotické vozidlo. Na základe tejto sme vytvorili simulačnú schému na ovládanie robotického vozidla a používali ju na hľadanie polohy cieľového bodu. Táto práca bola prípravou na použitie reálneho robota.

Zoznam použitej literatúry

[1] MATHEWS, J. H. and FINK, K. K. 2004. Numerical Methods Using Matlab, 4th Edition, Prentice-Hall Inc. Upper Saddle River, New Jersey, USA, ISBN: 0-13-065248-2

[2] ANESCU, G. and PRISECARU, I. 2015. NSC-PSO, a novel PSO variant without speeds and coefficients Polytechnic University of Bucharest,313 Splaiul Independentei, 060042, Bucharest, Romania

[3] https://www.mathworks.com

Prílohy

Príloha A: Matematický model robota, zapísaný v tvare S-funkcii

Príloha B: Funkcia na výpočet uhla o ktorý sa ma otočiť robot - kód v Matlabe

Príloha C: Program na riešenie optimalizačných úloh Luus-Jaakolovou metódou – kód v Matlabe

Príloha D: Program na riešenie optimalizačných úloh Nelder-Meadovou metódou – kód v Matlabe

Príloha E: Program na riešenie optimalizačných úloh metódou optimalizácii rojom častíc – kód v Matlabe

Príloha A: Matematický model robota, zapísaný v tvare S-funkcii

```
function [sys,x0,str,ts] = robomodelSF(t,x,u,flag,xinit)
r = 3e-2;
L = 1e-1;
%xinit = [0; 0; 0];

switch flag,
    case 0
        [sys,x0,str,ts] = mdlInitializeSizes(xinit);
    case 1
        sys = mdlDerivatives(t,x,u,r,L);
    case 3
        sys = mdlOutputs(t,x,u);
    case {2, 4, 9}
        sys = [];
    otherwise
        error(['unhandled flag = ',num2str(flag)]);
end
end

function [sys,x0,str,ts] = mdlInitializeSizes(xinit)

sizes = simsizes;
sizes.NumContStates = 3; % pocet spojitych stavov
sizes.NumDiscStates = 0; % pocet diskretnych stavov
sizes.NumOutputs = 3; % pocet vystupov
sizes.NumInputs = 2; % pocet vstupov
sizes.DirFeedthrough = 0; % = 0 v pripade, ze v rovniciach vystupu
% nevystupuje u alebo nevystupuje matica D. Inak =1.
sizes.NumSampleTimes = 1; % = 1 pre spojite systemy
sys = simsizes(sizes);
x0 = xinit(:); % zaciatocne podmienky pre dif. rovnice
str = []; % str je prazdna matica
ts = [0 0]; % velkost periody vzorkovania, pre spojite systemy =[0 0]
end

function sys = mdlDerivatives(t,x,u,r,L) % vypocet derivaci - stavove rovnice

sys(1) = (r/2)*cos(x(3))*(u(1)+u(2));
sys(2) = (r/2)*sin(x(3))*(u(1)+u(2));
sys(3) = r/L*(u(2)-u(1));
end

function sys = mdlOutputs(t,x,u) % vypocet vystupov - rovnice vystupu
sys(1) = x(1);
sys(2) = x(2);
sys(3) = mod(x(3)-pi, 2*pi)-pi;
end
```

Príloha B: Funkcia na výpočet uhla o ktorý sa má otočiť robot - kód v Matlabe

```
function uhol = d_uhol_s(sys, xc, yc)

x1 = sys(1);
x2 = sys(2);
alfa = sys(3);

a1 = xc;
a2 = yc;
ubeta = 0;
% Vypocet uhla beta

if a1 > x1 && a2 >= x2
    ubeta = atan((a2 - x2)/(a1 - x1));
elseif a1>x1 && a2<x2
    ubeta = atan((a2 - x2)/(a1 - x1));
elseif a1<x1 && a2>x2
    ubeta = atan((a2 - x2)/(a1 - x1))+pi;
elseif a1==x1 && a2>x2
    ubeta = pi/2;
elseif a1==x1 && a2<x2
    ubeta = -pi/2;
elseif a1<x1 && a2==x2
    ubeta = pi;
elseif a1<x1 && a2<x2
    ubeta = atan((a2 - x2)/(a1 - x1))-pi;
else
    ubeta = 0;
end

uhol = alfa-ubeta;
if uhol > pi
    uhol = -(2*pi - uhol);
end
end
```

Príloha C: Program na riešenie optimalizačných úloh Luus-Jaakolovou metódou – kód v Matlabe

```
function y = luus_jaakola(f,L,d)
close all
% f - optimalizaovana funkcia
% L - rozmer skumaneho prostredia
% d - zaciatocny rozmer okolia pre hladanie noveho bodu
d = 2.5;

x = [rand*L-L/2, rand*L-L/2];
c = [x(1) - d/2, x(2) - d/2];

iters = 0;
converged = false;

while ~converged
    iters = iters + 1;

    if iters > 30
        disp('Maximum number of iterations reached.');
        break
    end

    a = [c(1) + rand*d, c(2) + rand*d];

    assignin('base', 'xinit', [x(:); theta]');
    assignin('base', 'xciel', a(1));
    assignin('base', 'yciel', a(2));
    sim('robomodel_sim');

    [simout.Data(1, :); simout.Data(end, :)];
    a = [simout.Data(end, 1), simout.Data(end, 2)];
    theta = simout.Data(end, 3);

    converged = abs(f(x)-f(a))<1e-6;
    if f(a) < f(x)
        iter = iters;
        x = a;
        c = [x(1) - d/2, x(2) - d/2];
    else
        d = 0.9*d;
        c = [x(1) - d/2, x(2) - d/2];

    end
    if d < 1e-3
        disp('Box is too small.');
        break
    end
end

y = [x, f(x), iter];
end
```

Príloha D: Program na riešenie optimalizačných úloh Nelder-Meadovou metódou – kód v Matlabe

```
function y = nelder_mead(f,x1,y1)
close all
% f - optimalizovana funkcia
% [x1,y1] - zaciatocna poloha prveho robota
A = [x1 y1];
B = [x1+0.85 y1-0.07];
C = [x1+0.55 y1+1];
k = 1;
thetaA = 0;
thetaB = 0;
thetaC = 0;

converted = false;
iters = 0;
while ~converted
    iters = iters + 1;
    if iters > 30
        disp('Maximum number of iterations reached.');
        break
    end
    if k < 1e-3
        disp('Box is too small.');
        break
    end
    if f(A) > f(B) && f(A) > f(C)
        M = (B + C)/2;
        R = (k+1)*M - k*A;

        assignin('base', 'xinit', [A(:); thetaA]);
        assignin('base', 'xciel', R(1));
        assignin('base', 'yciel', R(2));
        sim('robomodel_sim');
        [simout.Data(1, :); simout.Data(end, :)];
        R = [simout.Data(end, 1), simout.Data(end, 2)];
        thetaA = simout.Data(end, 3);

        converted = abs(f(R)-f(A))<1e-2;
        if f(R) < f(A)
            iter = iters;
            A = R;
        else
            k = 0.95 * k;
            assignin('base', 'xinit', [R(:); thetaA]);
            assignin('base', 'xciel', A(1));
            assignin('base', 'yciel', A(2));
            sim('robomodel_sim');
            [simout.Data(1, :); simout.Data(end, :)];
            A = [simout.Data(end, 1), simout.Data(end, 2)];
            thetaA = simout.Data(end, 3);
        end

    elseif f(B) > f(A) && f(B) > f(C)
        M = (A + C)/2;
```

```
    R = (k+1)*M - k*B;

    assignin('base', 'xinit', [B(:); thetaB]);
    assignin('base', 'xciel', R(1));
    assignin('base', 'yciel', R(2));
    sim('robomodel_sim');
    [simout.Data(1, :); simout.Data(end, :)];
    R = [simout.Data(end, 1), simout.Data(end, 2)];
    thetaB = simout.Data(end, 3);
    converted = abs(f(R)-f(B))<1e-2;

    if f(R) < f(B)
        iter = iters;
        B = R;
    else
        k = 0.95 * k;
        assignin('base', 'xinit', [R(:); thetaB]);
        assignin('base', 'xciel', B(1));
        assignin('base', 'yciel', B(2));
        sim('robomodel_sim');
        [simout.Data(1, :); simout.Data(end, :)];
        B = [simout.Data(end, 1), simout.Data(end, 2)];
        thetaB = simout.Data(end, 3);
    end

elseif f(C) > f(B) && f(C) > f(A)
    M = (B + A)/2;
    R = (k+1)*M - k*C;

    assignin('base', 'xinit', [C(:); thetaC]);
    assignin('base', 'xciel', R(1));
    assignin('base', 'yciel', R(2));
    sim('robomodel_sim');
    [simout.Data(1, :); simout.Data(end, :)];
    R = [simout.Data(end, 1), simout.Data(end, 2)];
    thetaC = simout.Data(end, 3);
    converted = abs(f(R)-f(C))<1e-2;

    if f(R) < f(C)
        iter = iters;
        C = R;
    else
        k = 0.95 * k;
        assignin('base', 'xinit', [R(:); thetaC]);
        assignin('base', 'xciel', C(1));
        assignin('base', 'yciel', C(2));
        sim('robomodel_sim');
        [simout.Data(1, :); simout.Data(end, :)];
        C = [simout.Data(end, 1), simout.Data(end, 2)];
        thetaC = simout.Data(end, 3);
    end
end
if f(A) < f(B) && f(A) < f(C)
    y = [[A,f(A)], iter];
elseif f(B) < f(A) && f(B) < f(C)
    y = [[B,f(B)],iter];
```

```
    elseif f(C) < f(B) && f(C) < f(A)
        y = [[C,f(C)],iter];
    end
end
end
```

Príloha E: Program na riešenie optimalizačných úloh metódou optimalizacie rojom častíc – kód v Matlabe

```
function y = PSO(f)
% f - optimalizovana funkcia
a = -5:2.5:5;
swarm_size = 25;
[X,Y] = meshgrid(a,a);
C = cat(2,X',Y');
swarm = reshape(C,[],2);
swarm_b = reshape(C,[],2);
g = [-6,-6];

converged = false;

%Zistime globalne min swarmu
j = 1;
while j <= swarm_size
    i = swarm(j,(1:2));
    j = j + 1;
    if f(i) < f(g)
        g = i;
    end
end
iters = 0;
% Pohyb "agentmi" a hladanie globalneho nim funkcie
figure
while ~converged
    iters = iters + 1;
    if iters > 20
        disp('Maximum number of iterations reached.');
        break
    end
    j = 1;
    while j <= swarm_size
        %Logic: x(k+1) = x(k) + r1*(g - x(k)) + r2*(swarm_local_best - x(k)
        r1 = 2*rand - 1;
        r2 = 2*rand;
        i = swarm(j,(1:2));% aktualna pozicia agenta
        b = swarm_b(j,(1:2));% najlepsia zaregistrovana pozicia agenta
        a = i + r1*(g - i) + r2*(b - i);
        swarm(j,(1:2)) = a;

        if f(a) < f(i)
            swarm_b(j,(1:2)) = a;
            if f(a) < f(g)
                g = a;
            end
        end
        j = j + 1;
    end
end
end
```