

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Ondřej BENDA**
Osobní číslo: **A19B0339P**
Studijní program: **B0714A150005 Kybernetika a řídicí technika**
Specializace: **Automatické řízení a robotika**
Téma práce: **Řízení roje dronů**
Zadávací katedra: **Katedra kybernetiky**

Zásady pro vypracování

- Seznámení se s oblastí rojů dronů.
- Seznámení se s prostředím pro simulaci roje.
- Návrh řídicího algoritmu roje.
- Otestování řídicího algoritmu roje v simulačním prostředí.

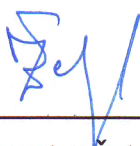
Rozsah bakalářské práce: **30 – 40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

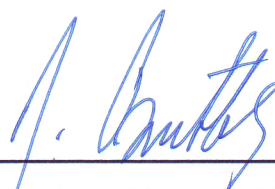
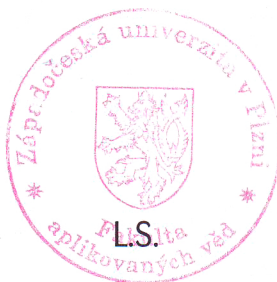
- Chung, S. J., Paranjape, A. A., Dames, P., Shen, S., & Kumar, V. (2018). A Survey on Aerial Swarm Robotics. IEEE Transactions on Robotics, 34(4), 837–855.
- Tahir, A., Böling, J., Haghbayan, M. H., Toivonen, H. T., & Plosila, J. (2019). Swarms of Unmanned Aerial Vehicles —A Survey. Journal of Industrial Information Integration, 16(August), 100106.
- M. Coppola, K. N. McGuire, C. De Wagter, and G. C. H. E. de Croon, 'A Survey on Swarming With Micro Air Vehicles: Fundamental Challenges and Constraints,' Front. Robot. AI, vol. 7, no. February, Feb. 2020.

Vedoucí bakalářské práce: **Ing. Zdeněk Bouček**
Výzkumný program 1

Datum zadání bakalářské práce: **15. října 2021**
Termín odevzdání bakalářské práce: **23. května 2022**



Doc. Ing. Miloš Železný, Ph.D.
děkan



Prof. Ing. Josef Psutka, CSc.
vedoucí katedry



FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA
KYBERNETIKY

BAKALÁŘSKÁ PRÁCE

Řízení roje dronů

Autor:
Ondřej Benda

Vedoucí práce:
Ing. Zdeněk Bouček

15. srpna 2022

Prohlášení

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

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

V Plzni dne 15. srpna 2022:

Abstrakt

Efektivitu práce jednoho dronu může zvýšit použitím roje dronů. Tato bakalářská práce se zabývá řízením takového roje. Čtenáři je popsán dron a je představena možnost řízení skupiny dronů decentralizovaným způsobem, tedy roje. Práce využívá simulačního prostředí SwarmLab, které poskytuje algoritmy řídící pohyb roje, které v kombinaci s algoritmem PSO řídí roj. Parametry těchto algoritmů jsou v experimentální části této práce laděny pomocí genetického algoritmu a jejich funkčnost je testována na řešení úloze, kterou je lokalizace unikajícího plynu.

Klíčová slova

dron, roj, decentralizované řízení, SwarmLab, PSO

Abstract

The efficiency of a single drone can be increased by using a swarm of drones. This bachelor thesis deals with the control of such a swarm. A drone is described to the reader and the possibility of controlling a group of drones in a decentralised way, i.e. a swarm is introduced. The thesis uses the SwarmLab simulation environment, which provides algorithms controlling the swarm movement, which in combination with the PSO algorithm control the swarm. The parameters of these algorithms are tuned in the experimental part of this thesis using a genetic algorithm and their functionality is tested on the assigned task, which is the location of leaking gas.

Keywords

drone, swarm, decentralized control, SwarmLab, PSO

Poděkování

Rád bych poděkoval svému vedoucímu bakalářské práce Ing. Zdeňku Boučkovi za jeho cenné rady, čas a vstřícnost, který mi během vypracování bakalářské práce věnoval.

Obsah

Abstrakt	ii
1 Úvod	1
2 Roje dronů	3
2.1 UAV	3
2.1.1 Popis dronu v prostoru	3
2.1.2 Limitace dronu	6
2.1.3 Skupina dronů	6
Centralizované řízení	6
2.2 Roj	7
2.2.1 Výhody roje	7
2.2.2 Využití roje dronů	8
2.2.3 Řízení roje	10
3 Simulátor SwarmLab	11
3.1 Třída <i>Drone</i>	11
3.1.1 Navigační model	12
3.2 Třída <i>Swarm</i>	14
3.3 Mapa	14
3.4 Algoritmy řízení pohybu roje	18
3.4.1 Vásárhelyiův algoritmus	18
Odpudivá a přitažlivá složka	18
Složka zarovnání rychlosti	19
Složka zajišťující vyhýbání se překážkám	20
Složka zajišťující přibližování se k cíli	20
Výsledná požadovaná rychlost	21
3.4.2 Olfati-Saberův algoritmus	21
3.4.3 Problém vyhýbání se konkávním překážkám	21
3.5 Parametry simulátoru	23
3.6 Práce se simulátorem	25
4 Particle swarm optimization	28
4.1 Stručná historie PSO	28
4.2 Algoritmus PSO	29
4.2.1 Omezení rychlosti	30
4.2.2 Setrvačná složka	30
4.2.3 Ohodnocovací funkce	31
4.3 Prohledávací fáze	31
4.4 Parametry algoritmu PSO	31
5 Testy	33

5.1	Simulovaná úloha	33
5.2	Parametry simulací	33
5.3	Ladění parametrů	37
5.3.1	Algoritmus řízení pohybu roje	37
	Pohyb dronů jako jednotlivců	37
	Pohyb dronů jako roje	38
5.3.2	Algoritmus PSO	41
5.4	Porovnání algoritmů	42
6	Závěr	49
	Bibliografie	50

Seznam obrázků

2.1	Dron typu fixed-wing (Shaer Blog, 2012)	4
2.2	Dron typu quadcopter (Wikipedia, 2022)	4
2.3	Popis dronu v prostoru (Nafia et al., 2018)	5
2.4	Světový rekord největšího počtu UAV ve vzduchu najednou (Guinness World Records, 2021)	7
2.5	Možný tvar roje při letu ve formaci	9
2.6	Možný tvar roje při shlukovém chování	9
3.1	Navigační model využívající strukturu vodopádového modelu	13
3.2	Trojdimenzionální mapa s mřížkovou strukturou objektů . .	15
3.3	Dvou-dimenzionální mapa s náhodným generováním objektů	16
3.4	Ukázka vizualizace waypointů (menší barevné kroužky) jed- notlivých dronů a unikajícího plynu (modrý oblak, sytost modré barvy určuje koncentraci plynu)	17
3.5	Náčrt případu obletění konvexní překážky	22
3.6	Náčrt případu nemožnosti obletění konkávní překážky . . .	22
3.7	Průběh stavových veličin dronu při simulaci	26
3.8	Průběh stavových veličin roje při simulaci	26
3.9	GUI dronu	27
3.10	GUI roje	27
5.1	Startovní pozice roje	34
5.2	Průběh první fáze simulační úlohy - přesun roje	34
5.3	První část druhé fáze simulační úlohy - hledání lokace s de- tekovatelnou koncentrací unikajícího plynu	35
5.4	Druhá část druhé fáze simulační úlohy - hledání zdroje uni- kajícího plynu	35
5.5	Konec simulační úlohy - nalezení zdroje unikajícího plynu .	36
5.6	Vývoj maxima hodnoty <i>fitness</i> funkce v jednotlivých gene- racích, při trénování pohybu dronů jako jednotlivců	38
5.7	Vývoj sumy <i>fitness</i> funkcí v jednotlivých generacích, při tré- nování pohybu dronů jako jednotlivců	39
5.8	Vývoj hodnoty parametru <i>r0_rep</i> , při trénování pohybu dronů jako jednotlivců	39
5.9	Vývoj maxima hodnoty <i>fitness</i> funkce v jednotlivých gene- racích, při trénování pohybu dronů jako roje	40
5.10	Vývoj hodnoty parametru <i>r0_shill</i> , při trénování pohybu dronů jako roje	41
5.11	Porovnání počtu dosažených waypointů drony, řízenými al- goritmem s manuálně a automaticky nastavenými parametry	43
5.12	Porovnání počtu dosažených waypointů rojem, řízeným al- goritmem s manuálně a automaticky nastavenými parametry	44

5.13	Porovnání kompaktnosti roje, řízeným algoritmem s manuálně a automaticky nastavenými parametry	44
5.14	Doba trvání první fáze algoritmu PSO, s manuálně a automaticky nastavenými parametry	45
5.15	Doba trvání druhé fáze algoritmu PSO (s konstantním parametrem ω), s manuálně a automaticky nastavenými parametry	45
5.16	Doba trvání algoritmu PSO (s konstantním parametrem ω), s manuálně a automaticky nastavenými parametry	46
5.17	Doba trvání druhé fáze algoritmu PSO, s konstantní a lineárně časově proměnnou hodnotou parametru ω	46
5.18	Doba trvání celé simulační úlohy (s konstantním parametrem ω), s manuálně a automaticky nastavenými parametry .	47
5.19	Doba trvání celé simulační úlohy (s lineárně časově proměnnou hodnotou parametru ω), s automaticky nastavenými parametry, při lokaci koncentrace unikajícího plynu s hodnotou 1	48

Seznam tabulek

3.1	Parametry roje (soubor param_swarm.m)	23
3.2	Parametry mapy (soubor param_map.m)	24
3.3	Parametry Vásárhelyiovo algoritmu řídicího pohyb roje (soubor param_vasarhelyi.m)	24
3.4	Parametry simulace (soubor param_sim.m)	25
4.1	Parametry algoritmu PSO (soubor param_pso.m)	32
5.1	Manuálně a genetickým algoritmem natrénované parametry Vásárhelyiovo algoritmu	40
5.2	Manuálně a genetickým algoritmem natrénované parametry první fáze algoritmu PSO	42
5.3	Manuálně a genetickým algoritmem natrénované parametry druhé fáze algoritmu PSO, za použití konstantní hodnoty parametru ω	42
5.4	Manuálně a genetickým algoritmem natrénované parametry druhé fáze algoritmu PSO, za použití lineárně časově proměnné hodnoty parametru ω	42

Seznam zkratek

GUI grafické uživatelské rozhraní. 25

PSO particle swarm optimization. ii, 1, 10, 12, 14, 17, 18, 28–31, 33, 36, 41–43, 49

UAV unmanned aerial vehicle. vi, 3, 6, 7, 11

1 Úvod

Drony se staly v moderním světě běžným technickým prostředkem, který je schopen řešit nespočet možných úloh. Je tomu tak díky jejich jednoduchému designu, od kterého se odvíjí i nízké finanční náklady, jejich velmi dobré mobility, schopnosti pohybovat se de facto jakýmkoliv prostředím, a možnosti osadit je nejrůznějšími senzory. To z nich dělá univerzální stroje uplatnitelné například pro monitorování, pořizování vizuálního obsahu, k transportu různých objektů a podobně.

Dron má však i svoje limity. Mohou jimi být například limitovaný čas pobytu ve vzduchu, limitovaná schopnost snímání prostředí, maximální možný náklad, který je možno přepravit, a podobně. Řešením těchto problémů může být použití roje dronů. Paradigma rojové robotiky se snaží překonat omezení jednoho robota pomocí spolupráce ve větších týmech. Inspirace přichází ze zvířecí říše, kde můžeme pozorovat chování zvířat, jež spojí síly ke společnému cíli, který je jinak pro samotného jedince příliš složitý nebo náročný (Garnier, Gautrais a Theraulaz, 2007). Použití více robotů najednou může přinést různé výhody a nové možnosti, jako je: redundance, rychlejší dokončení úkolů díky paralelizaci nebo provádění kolaborativních úkolů (Martinoli a Easton, 2003).

Cílem této práce je simulace řešení praktické úlohy pomocí roje dronů. Zvolenou úlohou je lokalizace unikajícího plynu. Tato úloha představuje typický příklad problému, u kterého je volba řešení pomocí roje dronů velmi vhodná. Klasickým příkladem zadané úlohy je porucha vedení plynu v objektu, v důsledku čehož dochází k úniku plynu. Lze předpokládat, že plyn je životu nebezpečný, a tudíž je vhodné využít k lokalizaci stroj. V objektu se mohou vyskytovat nejrůznější překážky, může se jednat o více patrovou budovu, plyn může unikat u stropu apod. Využití pozemního průzkumného prostředku je tedy nevhodné, naopak dron může tyto překážky hladce překonat. Zároveň použití roje dronů umožňuje čas, potřebný k lokalizaci, rapidně snížit, a tak i překonat limitace jednoho dronu, jako je například zmiňovaný limitovaný čas pobytu ve vzduchu.

Druhá kapitola nejdříve seznámí čtenáře se základními informacemi o dronu a popíše jeho pohyb v prostoru. Následně představí limity dronu a jejich řešení pomocí skupiny dronů. Popíše rozdíl mezi centralizovaným a decentralizovaným přístupem k řízení takové skupiny a dále rozvede druhý zmiňovaný přístup, jehož příkladem je roj.

Ve třetí kapitole je prezentováno simulační prostředí SwarmLab. Jsou zde představeny a popsány základní stavební kameny tohoto simulátoru, kterými jsou třídy *Drone* a *Swarm*, a je popsána práce se simulátorem.

Čtvrtá kapitola popisuje algoritmus PSO, který slouží k samotné lokalizaci unikajícího plynu. Je zde stručně popsána historie vzniku toho algoritmu, dále jsou prezentovány jeho jednotlivé složky a implementace algoritmu pro zadanou úlohu.

V páté kapitole je čtenář seznámen se vzorovou úlohou lokalizace úniku

plynu, která je zde simulována. Pomocí testů a genetického algoritmu jsou následně naladěny parametry jednotlivých algoritmů a je provedeno jejich provázání.

2 Roje dronů

Skupinu dronů řízenou decentralizovaně označujeme jako roj. Nejdříve si popíšeme jednoho jednotlivce roje, tedy dron, neboli UAV.

2.1 UAV

Termín unmanned aerial vehicle (UAV, česky bezpilotní letoun) odpovídá všem strojům, které létají ve vzduchu bez pilota na palubě, který by mohl letoun ovládat (Eisenbeiss et al., 2004). Mohou být ovládány na dálku člověkem nebo mohou být řízeny autonomně. Tato práce se zabývá druhým jmenovaným způsobem. Dalším možným označením těchto strojů je termín dron (z anglického drone), který bude v následujících kapitolách používán. Drony lze dělit do dvou hlavních skupin, typ fixed-wing (česky letadlo s pevnými křídly, obrázek 2.1) a quadcopter¹ (česky kvadrokoptéra, obrázek 2.2). Jedním z hlavních rozdílů mezi těmito typy je jejich schopnost pohybu v prostoru (více v sekci 3.1). Proto si nejdříve nadefinujeme popis dronu v prostoru.

2.1.1 Popis dronu v prostoru

Pro následující popis byl použit článek autora Luukkonen, 2011.

Pro popis dronu v prostoru nadefinujeme dva rámce, inerciální a rámec tělesa. Inerciální rámec F^e je spojený se zemí, systém je pravotočivý, kladný směr osy z_e směřuje ven ze země. Rámec tělesa F^b je pevně spojen s dronem a jeho počátek se nachází v těžišti dronu. Předpokládáme, že dron je typu kvadrokoptéra, osa z_b rámce F^b je tedy rovnoběžná s osou rotace rotorů. Polohu dronu lze pak popsat souřadnicemi počátku rámce F^b v prostoru F^e vektorem

$$\xi = [x, y, z]^T. \quad (2.1)$$

Natočení dronu je definováno vektorem η pomocí Eulerovo úhlů ϕ , θ , ψ jako natočení os F^b vůči inerciálním. Předpokládáme, že standardní pohyb dronu je ve směru osy x_b . Pak rotaci kolem osy x_b označíme ϕ (anglicky označováno jako roll), rotaci kolem osy y_b označíme θ (pitch) a rotaci kolem osy z_b označíme ψ (yaw), tedy

$$\eta = [\phi, \theta, \psi]^T \quad (2.2)$$

Grafické znázornění těchto vztahů zobrazuje obrázek 2.3.

Označme jednotlivé motory kvadrokoptéry čísly 1 až 4 tak, že protilehlé motory jsou 1 a 3, resp. 2 a 4, a motor 1 leží v pozitivním směru osy x_b .

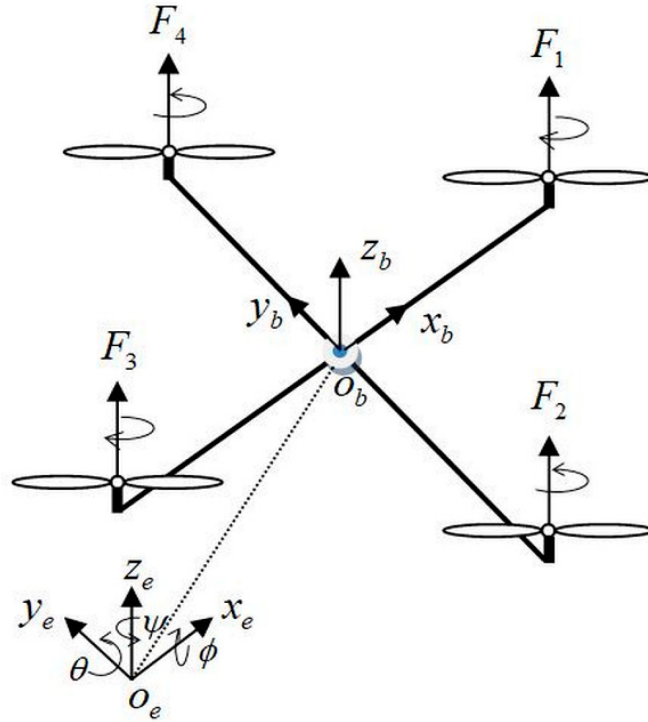
¹Kvadrokoptéra není obecným pojmenováním této skupiny. Obecně se jedná o helikoptéry, tedy stroje s horizontálně rotujícími rotory generujícími zdvih. Kvadrokoptéra, tedy helikoptéra se čtyřmi rotory, je však nejčastějším zástupcem této skupiny, a proto se často používá toto označení.



OBRÁZEK 2.1: Dron typu fixed-wing (Shaer Blog, 2012)



OBRÁZEK 2.2: Dron typu quadcopter (Wikipedia, 2022)



OBRÁZEK 2.3: Popis dronu v prostoru (Nafia et al., 2018)

Úhlovou rychlost motoru i označme ω_i . Při úhlové rychlosti vrtule ω_i je generován tah f_i ve směru osy rotace motoru: $f_i = k\omega_i^2$, kde k je konstanta zdvihu. S úhlovou rychlostí a zrychlením je zároveň generován točivý moment okolo osy rotace motoru: $\tau_{M_i} = b\omega_i^2$, kde b je konstanta odporu. Dopad zrychlení je minimální, a proto byl zanedbán. Součtem sil f_i získáme celkový tah T ve směru osy z_b rámce F^b :

$$T = \sum_{i=1}^4 f_i, \quad \mathbf{T}^b = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \quad (2.3)$$

Vektor točivého momentu $\boldsymbol{\tau}_b$ se skládá z točivých momentů τ_ϕ , τ_θ a τ_ψ , ve směrech odpovídajících orientaci úhlů rámce F^b :

$$\boldsymbol{\tau}_b = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} l \cdot k(-\omega_2^2 + \omega_4^2) \\ l \cdot k(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^4 \tau_{M_i} \end{bmatrix}, \quad (2.4)$$

kde l je vzdálenost mezi motorem a těžištěm kvadroptéry.

Vidíme, že pro změnu úhlu ϕ , tedy rotaci okolo osy x_b , musíme změnit poměr tahu motoru 2 a 4, resp. pro změnu úhlu θ motoru 1 a 3. Z obrázku 2.3 vidíme, že směr rotace protilehlých motorů je stejný, avšak směr rotace sousedních motorů je opačný. Díky tomu, při identickém součtu tahů protilehlých motorů, nedochází k rotaci ve směru osy z_b , tedy změně úhlu ψ . Chceme-li této rotaci dosáhnout, změníme poměr celkového tahu dvojic protilehlých motorů.

2.1.2 Limitace dronu

Drony mají velký potenciál uplatnění vzhledem ke svým vlastnostem. Můžeme je osadit nejrůznějším vybavením jako jsou senzory, aktuátory apod. a díky jejich velmi dobré mobilitě v nejrůznějších prostředích jsou schopny dopravit toto vybavení na požadovanou lokaci. Mají však i své limity, na které se podíváme v následujících odstavcích.

Jednou z výzev, které drony čelí, je skutečnost, že musejí nést svoji vlastní váhu. To vytváří potřebu směřovat k více minimalistickému designu dronu (Coppola et al., 2020). Pokud chceme zvýšit nosnou kapacitu dronu, aby mohl například nést více vybavení, musíme také zvětšit samotné prvky zajišťující let dronu, tedy například baterii a motory. Jejich zvětšením nicméně vzroste hmotnost samotného dronu. Tím se dostáváme do smyčky, často nazývané jako „efekt sněhové koule“ (Obert, 2009).

Další limitací mohou být samotné senzory, které nemusí mít dostatečnou přesnost nebo je jejich cena příliš vysoká.

Čas, za který je dron schopný vykonat daný úkol, je také limitovaný. Dron se může pohybovat omezenou rychlostí, která je dána například výkonem jeho motorů, hranicí bezpečného letu bez kolizí nebo maximální rychlostí snímání senzoru.

Všechny tyto limitace, a mnohé další, můžeme stěží řešit vylepšováním dronu jako jednotlivce. Možným řešením se tedy nabízí použít dronů více.

2.1.3 Skupina dronů

Potřeba překonání limitací dronů (sekce 2.1.2) nás přivádí na jedno z možných řešení tohoto problému, kterým je navýšení počtu dronů. Můžeme tak posunout limity dále, nebo získat úplně nové schopnosti vzniklé skupiny dronů. Vytvoříme tím ale nový problém, potřebu tuto skupinu řídit. Architektura řízení může být dvojího typu, a to centralizované, nebo decentralizované řízení.

Centralizované řízení

Při centralizovaném řízení jsou všechny drony ve skupině řízeny jedním počítačem. Tato „vševědoucí“ entita zná příslušné stavy všech dronů, podle kterých plánuje akční zásahy. Plánování lze provést apriorně² a/nebo proměnlivě v čase. Tento přístup se často používá v kombinaci s externím snímání polohy, to je dosaženo zpravidla s pomocí Globálního družicového navigačního systému (anglicky Global Navigation Satellite System, GNSS), nebo systému snímání polohy (anglicky Motion Capture System, MCS), v závislosti zpravidla na tom, zda se drony nacházejí venku nebo uvnitř (Coppola et al., 2020). Příkladem využití kombinace centralizovaného řízení a externího snímání polohy je aktuální světový rekord největšího počtu UAV ve vzduchu najednou, který se uskutečnil na akci pořádané automobilkou značky Genesis, kde se ve vzduchu najednou nacházelo 3281 dronů (obrázek 2.4) (Guinness World Records, 2021).

Centralizované řízení má však i své nedostatky. Jeden z problémů nastává při absenci zmiňovaného externího snímání polohy, což značně ztěžuje řízení centrálnímu počítači. Druhým, a to zřejmě podstatně větším, problémem je

²Apriorně znamená „předem známé“, nezávisle na zkušenosti získané při prováděném úkonu



OBRÁZEK 2.4: Světový rekord největšího počtu UAV ve vzduchu najednou (Guinness World Records, 2021)

právě podstata centrálního řízení, u kterého vše závisí na jedné centrální entitě. Zvětšováním počtu dronů roste výpočetní náročnost operací, které musí centrální počítač vykonat, což dělá systém obtížně škálovatelný. Zároveň, při chybě centrální entity se tato chyba přenáší na všechny drony, tedy hrozí kolaps celého systému.

Na obdobnou překážku můžeme narazit v oblasti komunikace mezi drony a centrální entitou. Má zpravidla omezenou propustnost, tedy při navyšování počtu dronů musíme buďto snížit objem komunikovaných dat nebo, pokud to vůbec lze, navýšit komunikační kapacity. Tím však může narůst zpoždění v komunikaci mezi dronem a centrální entitou, které je v řízení, nejen dronů, klíčovým faktorem. Riskujeme nárůst zpoždění do takové míry, že skupinu dronů již nejsme schopni řídit.

Možným řešením těchto problémů je decentralizované řízení skupiny, kde každý jednotlivý dron dělá individuální rozhodnutí o svém chování. Takto řízené skupině pak můžeme říkat roj.

2.2 Roj

Roj (anglicky swarm) se skládá z mnoha jednoduchých entit, které interagují mezi sebou i prostředím. Vznik makroskopického chování roje a schopnost dosáhnout významných výsledků jako tým, je důsledkem kombinace jednoduchého nebo mikroskopického chování (Hinchey, Sterritt a Rouff, 2007). Inspirací pro vznik metod řízení, založených na tomto principu, je chování zvířat, které žijí ve velkých společenstvech. Jedním z příkladů je chování mravenců. Jejich kolonie, čítající až milióny jedinců, vykazují fascinující chování, které kombinuje efektivitu s flexibilitou a robustností (Camazine et al., 2020).

2.2.1 Výhody roje

Jedním z hlavních rozdílů mezi výše zmiňovaným centralizovaným řízením a rojovým, tedy decentralizovaným řízením skupiny, je v komunikaci. V

důsledku absence vlastního rozhodování musí drony, v případě centralizovaného řízení, komunikovat s centrální entitou, která tak přijímá informace od všech dronů. Tyto data vstupují do komplexního řídicího algoritmu, který na jejich základě generuje a odesílá požadované chování všech dronů. Lze tedy říct, že všechny drony spolu komunikují navzájem. Důsledkem je již zmiňovaná obtížná škálovatelnost takto řízeného systému. V případě roje je situace jiná. Každý dron rozhoduje o svém chování sám, na základě svého řídicího algoritmu. Ten samozřejmě potřebuje pro svůj chod vstupní data, ty však nemusejí být globálního charakteru, pro požadované chování roje stačí často pouze data lokální. Jednotlivé drony tedy mohou komunikovat jen s blízkými sousedy. Tato vlastnost umožňuje teoreticky libovolnou škálovatelnost takto řízeného systému, bez nutnosti změny řídicího algoritmu, použitých technologií apod.

Další charakteristickou vlastností roje je vysoká redundance, vyplývající z velkého počtu entit, které ho tvoří. Redundance, společně s absencí centralizovaného řízení, předchází existenci bodu selhání, kde selhání jedné části systému může vést ke kolapsu celku. Roj je díky redundanci odolný vůči selhání nebo ztrátě některých entit, stejně tak i odolnější k šumu měření. (Kegeleirs, Grisetti a Birattari, 2021)

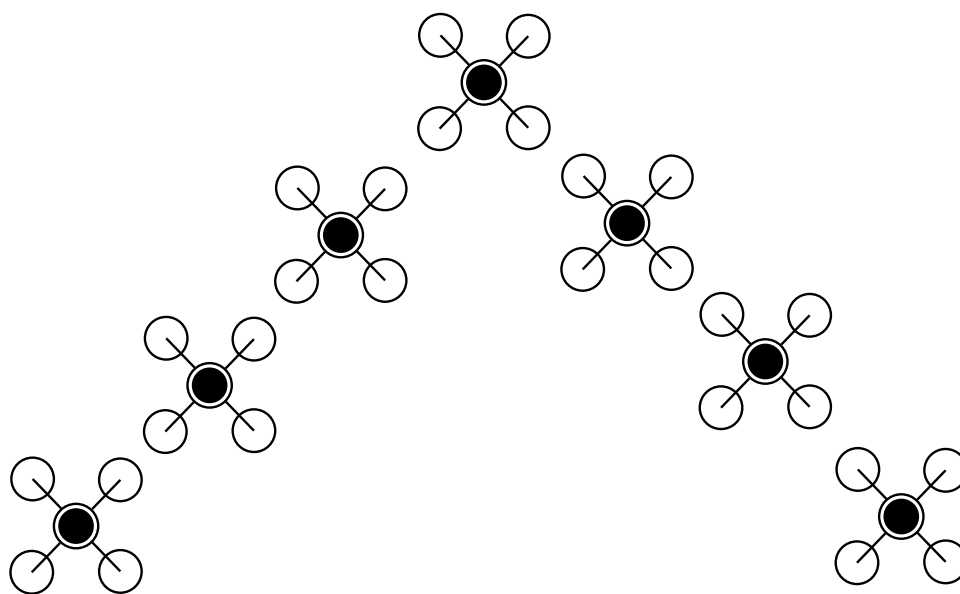
V neposlední řadě je výhodou roje jeho flexibilita. Fakt, že jsou drony řízeny jednoduchým algoritmem, umožňuje nasazení roje s minimální úpravou řídicího algoritmu na množinu úloh, které jsou si podobné (například lokalizace něčeho v prostředí). Stejně tak je díky tomu roj flexibilní v různých prostředích, dokonce i pokud se podmínky mění v čase.

2.2.2 Využití roje dronů

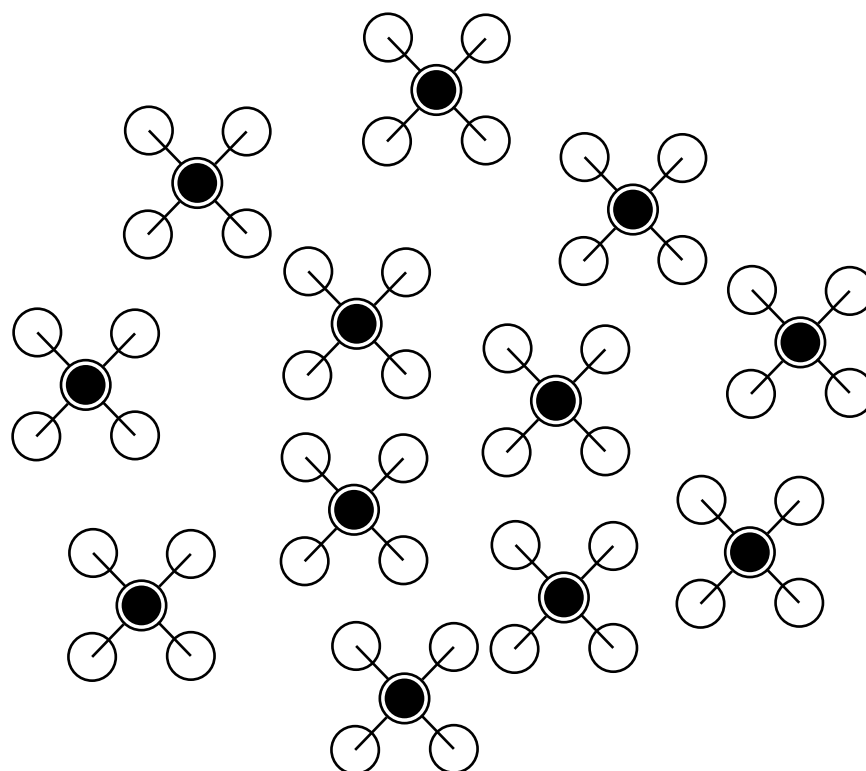
Roj dronů lze použít k řešení různých úloh reálného světa. Podle specifikace úlohy volíme jeden ze základních typů chování roje, jimiž jsou: shlukování (anglicky flocking), let ve formaci a distribuované snímání (Coppola et al., 2020). Na následujících řádkách se blíže podíváme na tyto chování a jejich aplikace.

Při letu ve formaci si jednotlivé drony udržují relativní pozici nebo vzdálenost mezi zvolenými sousedy, a tak se pohybují prostředím jako jeden celek. Výhodou tohoto chování je predikovatelnost roje během operace (Coppola et al., 2020). Let ve formaci je vhodný například pro transport objektu nebo již zmiňované vizuální show. Příklad možného letu ve formaci zobrazuje obrázek 2.5.

Shlukovací chování roje je inspirováno pohybem zvířat v přírodě, například let hejna ptáků (z toho také anglický výraz flocking, překlad slova flock je hejno). Toto chování je typicky charakterizováno kombinací jednoduchých lokálních pravidel, a to: přitažlivé síly, odpudivé síly, zarovnání směru letu se sousedy a zarovnání rychlosti letu se sousedy (Coppola et al., 2020). Tyto pravidla přirozeně předcházejí kolizím mezi agenty prostřednictvím pravidla odpuzování. Zároveň je takto možné dosáhnout bezpečné společné navigace v prostředí s překážkami, kde překážky působí dodatečnou odpudivou silou (Saska, Vakula a Přeučil, 2014). Shlukové chování tak vykazuje lepší schopnost navigace prostředím s překážkami než let ve formaci. Je ho tedy v takovýchto prostředích vhodnější použít, pokud není požadována pevně daná formace roje. Příklad shlukového chování zobrazuje obrázek 2.6.



OBRÁZEK 2.5: Možný tvar roje při letu ve formaci



OBRÁZEK 2.6: Možný tvar roje při shlukovém chování

Při distribuovaném snímání se může roj chovat jedním ze dvou výše zmiňovaných typů chování. Může se však také postupně rozprostírat prostředím. Příkladem takového chování je algoritmus PSO, který popisuje kapitola 4. Možné uplatnění tohoto chování je průzkum nebo mapování prostředí.

2.2.3 Řízení roje

Hlavní obtíží decentralizovaného přístupu řízení skupiny dronů je potřeba vytvoření a naladění řídicích algoritmů roje. Tyto algoritmy lze rozdělit do dvou skupin, resp. vrstev, které dohromady zajišťují jeho požadované chování. První vrstvou je algoritmus řídicí pohyb roje, který zajišťuje jeho pohyb na požadovaná místa, popř. požadovaným směrem. Dva takovéto algoritmy popisuje sekce 3.4. Druhou vrstvou, která je nad vrstvou předešlou, tedy ji předává požadované úkony, je algoritmus, který generuje požadovaný pohyb roje. V této práci je použit algoritmus PSO, který popisuje kapitola 4.

Po výběru algoritmů přichází druhý úkol, kterým je nastavení jejich parametrů. To je možno udělat manuálně nebo automaticky, oba přístupy jsou testovány v kapitole 5. Manuální způsob zahrnuje odhad vhodných počátečních hodnot a následným testováním jejich úpravu, s cílem zlepšení požadovaných kvalit chování roje. Těmi může být rychlost provedení úkolu, jeho bezpečnost a podobně. Automatické ladění parametrů je založeno na hledání optimálních parametrů pomocí algoritmu. Ten pracuje na principu hledání (nebo alespoň přiblížení se) extrému tzv. *fitness* funkce, kterou lze vyhodnotit pro jednotlivé sady parametrů. Tato funkce je zvolena tak, aby odrážela kvalitu řešení úlohy, kterou sada parametrů poskytuje. Jednou ze skupin takovýchto algoritmů jsou *genetické algoritmy*, které heuristicky hledají extrém fitness funkce na základě principů evoluční biologie. Tyto algoritmy napodobují techniky evolučních procesů, které můžeme pozorovat u živých organismů. Pro automatické ladění parametrů řídicích algoritmů v této práci byl použit jednoduchý genetický algoritmus, sestavený na základě knihy M. Mitchella *An Introduction to Genetic Algorithms* z roku 1998.

Algoritmus se skládá z několika fází. V první fázi je náhodně vygenerováno v zadaných intervalech n sad parametrů, kde jednu sadu označujeme jako „chromozom“. Tyto chromozomy tvoří takzvanou první „generaci“. V druhé fázi je každému z nich vyhodnocena fitness funkce. Následně ve třetí fázi je vybrána dvojice chromozomů, přičemž pravděpodobnost výběru chromozomu se odvíjí od hodnoty jeho fitness funkce (vyšší hodnota fitness funkce znamená větší pravděpodobnost výběru). U vybrané dvojice chromozomů, nazývaných „rodiče“, dochází s nastavitelnou *pravděpodobností křížení* ke „křížení“. Vznikají tak dva potomci, kteří vlastní kombinaci³ rodičovských chromozomů. Pokud nedojde ke křížení, potomci jsou kopiemi svých rodičů. Ve čtvrté fázi dochází s *pravděpodobností mutace* k „mutaci“, při níž jsou náhodně některé parametry chromozomu potomka lehce pozměněny. Získáme tak dva nové potomky. Následně opakujeme fáze tři a čtyři, dokud nezískáme n potomků, kteří nám vytvoří novou generaci. S novou generací se přesuneme do fáze dva a proces takto pokračuje do doby, než je dosaženo předem zadaného počtu generací.

³Pro vznik kombinace rodičovských chromozomů je nejdříve náhodně zvolen jeden z parametrů. Následně jeden potomek dostává od prvního rodiče sadu parametrů, od prvního po náhodně vybraný parametr, a zbytek pak od rodiče druhého. Druhý potomek je vytvořen opačně.

3 Simulátor SwarmLab

Simulátor SwarmLab¹ (Soria, Schiano a Floreano, 2020) poskytuje uživateli silný pracovní nástroj pro práci s drony. Dle uživatelských potřeb lze zvolit detailní simulaci jednoho dronu a jeho dynamiky, ale i libovolně velkého roje. Zároveň budou v této kapitole popsány vlastní úpravy tohoto simulátoru. Celý software je napsaný v prostředí MATLAB². Toto prostředí je pro simulátor vhodné z několika důvodů. MATLAB je skriptovací jazyk s vysokou úrovní abstrakce, umožňující uživateli programovat, i bez rozsáhlé zkušenosti s programováním. Dále prostředí obsahuje mnoho vestavěných nástrojových sad pro návrh, řízení, analýzu a vizualizaci studovaného systému, čímž ještě více snižuje programovací náročnost pro uživatele. Také poskytuje možnost automatického překladu kódu do programovacích jazyků C/C++, ulehčuje tak přesun kódu do reálného prostředí. Nakonec prostředí MATLAB umožňuje použití Objektově orientovaného programování³, na kterém je simulátor SwarmLab postaven. (Soria, Schiano a Floreano, 2020)

Základními stavebními kameny simulátoru jsou dvě třídy *Drone* a *Swarm*, které budou popsány v následujících sekcích.

3.1 Třída *Drone*

Třída *Drone* je jedním ze základních stavebních kamenů simulátoru pro následné simulování roje. Každá instance třídy *Drone* obsahuje proměnné třídy, jako:

- *uav_type*: typ UAV
- *pos_ned*: polohu v inerciálním rámci (vektor 3×1 ve formátu ned, tedy north, east, down, čili severní, východní a výšková souřadnice, která je orientovaná ve směru gravitačního vektoru)
- *vel_xyz*: rychlost (forma obdobná jako poloha)
- *attitude*: natočení dronu (v Eulerovo úhlech ϕ, θ, ψ)
- *rates*: úhlové rychlosti natočení dronu ($\dot{\phi}, \dot{\theta}, \dot{\psi}$)

dále pak proměnné navigačního modelu, např.:

¹Repozitář původního simulátoru SwarmLab: <https://github.com/lis-epfl/swarmlab>

²MATLAB, název složený z anglických slov *matrix laboratory*, tedy *maticová laboratoř*, je interaktivní programovací prostředí a skriptovací jazyk. Jak název napovídá, hlavním využitím toho prostředí je práce s maticemi. Dále umožňuje vykreslování funkcí a dat, implementaci algoritmů, vytváření uživatelských rozhraní a mnoho dalších funkcí.

³Objektově orientované programování je programovací paradigma, ve kterém je kód rozdělen do jednotlivých objektů, které obsahují atributy a metody. Hlavními principy jsou zapouzdření, abstrakce, dědičnost a polymorfismus.

- *nb_waypoints*: počet waypointů⁴
- *waypoints*: vektor waypointů (tedy matice $3 \times nb_waypoints$)

a nakonec proměnné algoritmu PSO:

- *best_score*: prozatímní nejlepší dosažené skóre dronu (viz. kapitola 4)
- *best_coord*: souřadnice bodu s nejlepším dosaženým skórem
- *pso_velocity*: vektor rychlosti používaný v algoritmu PSO

Simulátor nabízí dva typy UAV a to *fixed-wing* a *quadcopter*. Hlavním rozdílem, mezi těmito dvěma typy, je schopnost kvadrokoptéry stát ve vzduchu na místě a rotovat kolem vertikální osy (yaw). Může tak měnit směr letu na místě, popř. i létat v libovolném směru, bez nutnosti rotace kolem vertikální osy. Tuto schopnost letouny s pevnými křídly nemají, proto pro změnu směru letu musejí následovat příslušnou orbitu.

3.1.1 Navigační model

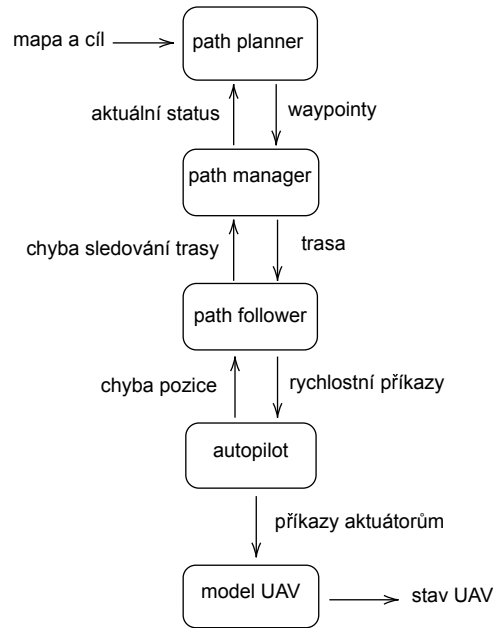
Pro ovládání pohybu dronu poskytuje simulátor navigační model. Ten se skládá z několika vrstev, které si mezi sebou předávají informace, takzvaný vodopádový model. Každá z vrstev modelu přijímá vstupní data z vrstvy nad ní a vytváří z obecnějšího úkolu (např. dopravení se z jednoho bodu do druhého po přímce) vstupní data pro vrstvu pod ní (např. směr, ve kterém by se měl dron vydat), která řeší úkol konkrétnější. Jednotlivé vrstvy modelu si tak sestupně předávají informace, podobně jako svažující se tok vodopádu. Navigační model simulátoru SwarmLab zobrazuje obrázek 3.1. Následující odstavce popisují jednotlivé vrstvy navigačního modelu, a to od vrstvy nejnižší.

První vrstvou je **autopilot**. Ten na vstupu přebírá příkazy, jakým směrem se má dron pohybovat (např. "leť rychlostí 2 m/s na sever"). Podle verze autopilota pak vygeneruje příkazy pro aktuátory dronu. Autopilot se liší podle toho, jestli používáme kvadrokoptéru nebo letadlo s pevnými křídly. Pro kvadrokoptéru simulátor nabízí čtyři různé verze autopilota:

- autopilot v natočení
- autopilot v rychlosti
- autopilot ve zrychlení
- autopilot v pozici

Jako výchozí je nastaven autopilot v rychlosti. Výstupem autopilota může být například příkaz „nastav příkaz motorům kvadrokoptéry na [0,2 0,2 1 1]“. Tento příkaz by byl u reálného dronu předán regulační zpětnovazební

⁴Waypoint, česky lze přeložit jako „trasový bod“, je bod nebo místo, jejichž množina, tvoří trasu nebo linii cesty. Dron se snaží dopravit k prvnímu waypointu, z jeho množiny waypointů, po jeho dosažení dron mění kurz a cestuje k následujícímu waypointu, pokud takový existuje, jinak zůstává na místě, tedy na posledním dosaženém waypointu.



OBRÁZEK 3.1: Navigační model využívající strukturu vodopádového modelu

smyčce⁵ rotorů s PID regulátorem⁶, toto chování je v MATLABU simulováno pomocí funkce `pid_loop`. PID regulátory jsou velmi často používané pro stabilizaci kvadrokoptér. Jejich hlavními výhodami jsou jednoduchá struktura, cenová dostupnost a snadná implementace požadované regulace.

Druhou vrstvou je **path follower**, do češtiny lze přeložit jako „sledovač cesty“. Tato vrstva přijímá na vstupu cestu, kterou je přímka z počátečního bodu v daném směru, a na výstupu generuje příkazy pro autopilota.

Následuje vrstva **path manager**, tedy „správce cesty“. Ten vytváří cestu z waypointů, které přebírá na vstupu.

Poslední vrstvou je **path planner**, česky „plánovač cesty“, který přidává do simulátoru možnost automatického generování waypointů pro vrstvu **path manager**, na základě znalosti mapy, aktuální polohy a požadovaného cíle. Za tímto účelem je použit algoritmus rapidly-exploring random tree (RRT). Tuto vrstvu, resp. algoritmus, nebudeme dále používat, protože v rámci zadané úlohy nepředpokládáme absolutní znalost prostředí, ve kterém se drony nacházejí. Tato informace je však pro tento algoritmus stěžejní. Předpokládáme pouze znalost nejbližšího okolí dronu, které snímá pomocí senzorů. Pro navádění dronů budeme používat algoritmy pro řízení pohybu roje, popsány v sekci 3.4, a algoritmus pro hledání unikajícího plynu, popsány v kapitole 4.

⁵Regulační zpětnovazební smyčka slouží k regulaci sledované veličiny systému. Na vstupu přijímá požadovanou hodnotu regulované veličiny a aktuální hodnotu regulované veličiny, kterou pomocí zpětné vazby předává na vstup regulátoru. Ten na jejich základě generuje požadované hodnoty vstupu regulovaného systému. Její hlavní výhodou oproti regulaci přímé, ve které není používána aktuální hodnota regulované veličiny, je schopnost reakce na vnější poruchy a nepřesné znalosti regulovaného systému.

⁶PID regulátor je v praxi často používaný typ regulátor, skládající se ze tří složek. Proporcionální složka je prostý zesilovač, tedy výstupní veličina je přímo úměrná vstupní veličině. Výstup Integrovní složky je přímo úměrný určitému integrálu ze vstupní veličiny od počátečního do aktuálního času. Nakonec složka Derivační, jejíž výstup je přímo úměrný derivaci vstupní veličiny.

3.2 Třída *Swarm*

Jak název třídy napovídá, třída *Swarm* představuje roj dronů. Instance této třídy obsahuje proměnné třídy, jako:

- *drones*: vektor obsahující instance třídy *Drone*, tedy drony, které jsou součástí roje
- *nb_drones*: počet dronů v roji
- *equivalent_drone*: ekvivalentní dron roje, jedná se o virtuální dron v těžišti roje, který je následně využíván např. v naváděcím modelu
- *swarming_algo*: typ algoritmu používaný pro řízení pohybu roje (viz. sekce 3.4)
- *collisions_history*: historie kolizí jak mezi dvěma agenty, tak mezi agentem a překážkou (viz. sekce 3.3)

dále pak proměnné pro navádění roje⁷:

- *as_swarm*: proměnná typu `boolean`⁸ která určuje, zda se roj má pohybovat jako celek
- *swarm_goal*: waypoint určující cíl roje, pokud cestuje jako celek
- *as_pso*: proměnná typu `boolean` určující, zda se roj řídí algoritmem PSO (viz. kapitola 4)

a nakonec proměnné algoritmu PSO:

- *best_score*: prozatimní nejlepší dosažené skóre roje (viz. kapitola 4)
- *best_coord*: souřadnice bodu s nejlepším dosaženým skórem

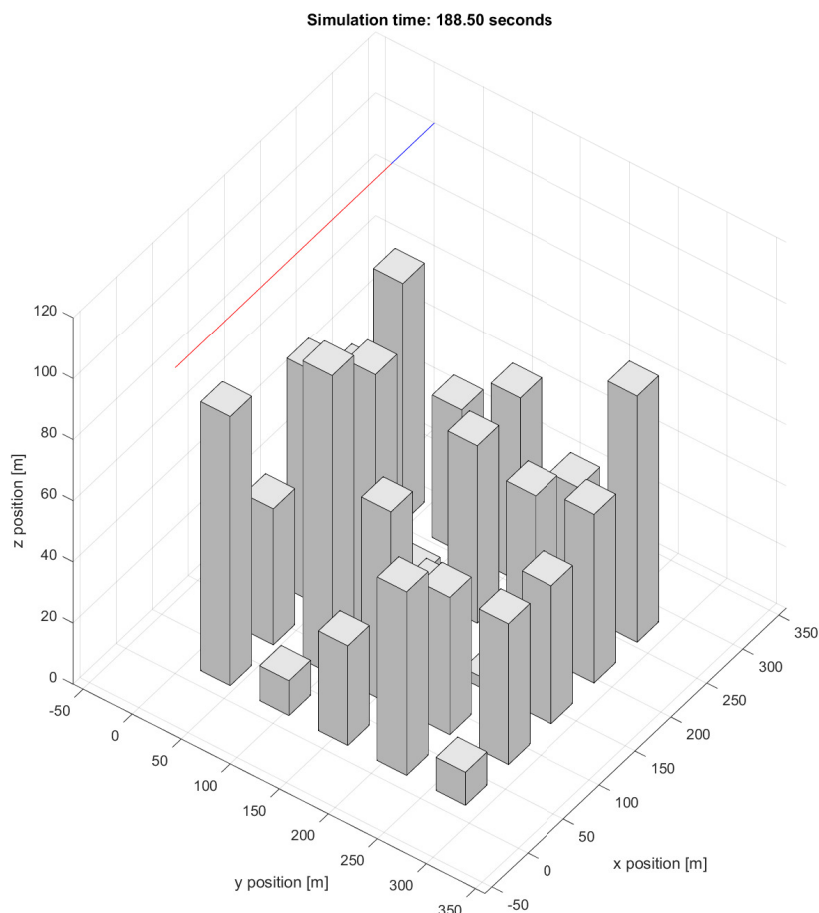
Hlavní částí této třídy jsou algoritmy řízení pohybu roje, které popisuje sekce 3.4. Před tím je ale potřeba se, pro kontext, podívat na prostředí, umožňující pohyb dronů, které bude popsáno v následující sekci.

3.3 Mapa

Další podstatnou částí simulátoru SwarmLab je prostředí, ve kterém se dron/drony pohybují. Pro tyto účely nabízí simulátor generátor mapy s požadovaným počtem objektů. Objekty, sloužící jako překážky, jsou válce, orientované tak, že stojí na své podstavě. Vizualizované jsou však jako hranoly (je tomu tak z důvodu jednodušší vizualizace hranolu než válce, naopak pro simulační účely je vhodnější válec, podrobněji dále v této sekci). Simulátor nabízí možnost vygenerovat mřížkovou strukturu ve tvaru čtverce (při pohledu shora), s libovolným počtem objektů. Zároveň byla do simulátoru přidána funkce generování náhodné mapy, tedy mapy se zadaným počtem objektů, které mají náhodnou velikost (průměr válce) a pozici. Je však ošetřeno, aby se objekty nepřekrývaly a byla mezi nimi minimální zadaná vzdálenost. Toto opatření zabraňuje vzniku velkých překážek, složených z několika jednotlivých objektů, které by mohly být pro drony nepřekonatelné (více viz

⁷Tyto proměnné, a jim příslušné metody roje, sloužící k funkčnosti algoritmu PSO, byly do simulátoru přidány

⁸`Boolean` je datový typ nabývající pouze dvou hodnot, a to `true`, nebo `false`

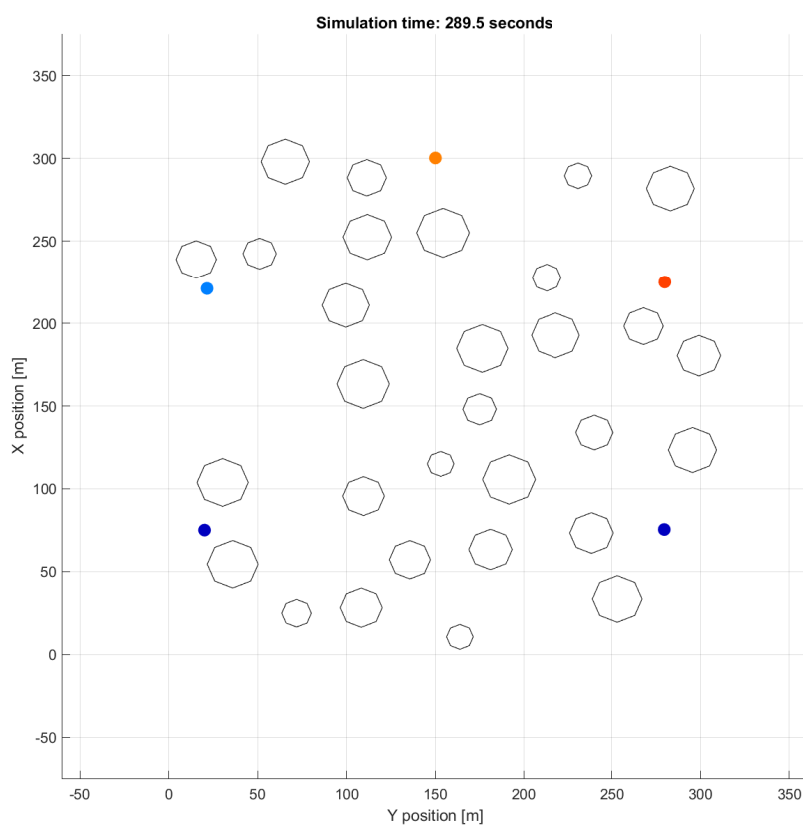


OBRÁZEK 3.2: Trojdimenzionální mapa s mřížkovou strukturou objektů

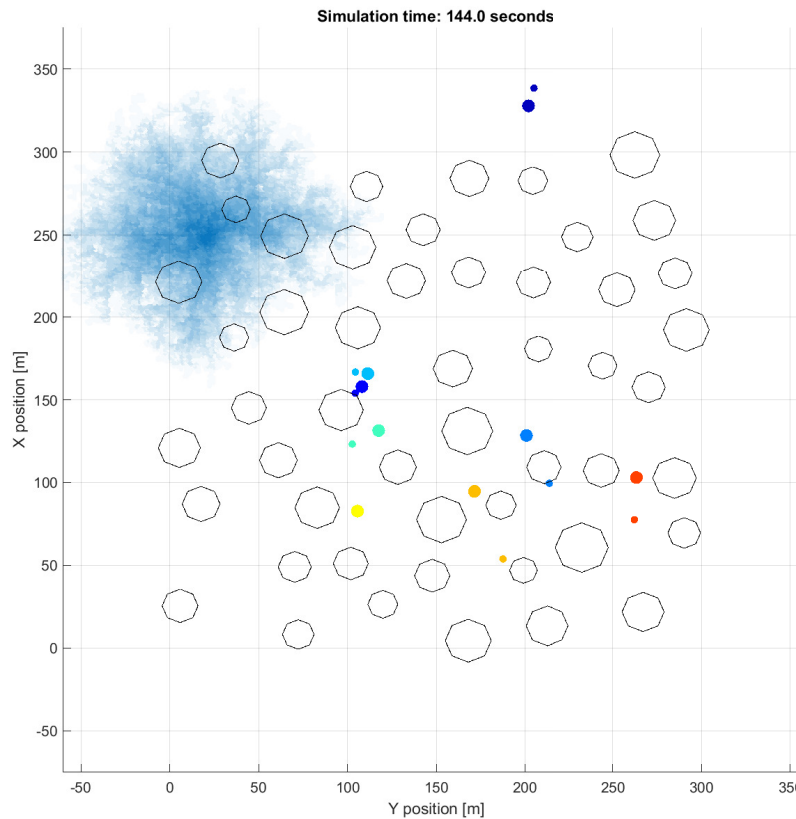
sekce 3.4). Funkci náhodného generování objektů můžeme aktivovat parametrem `RANDOM_BUILDINGS`, který se nachází na začátku hlavního souboru simulátoru `swarm_main` (viz sekce 3.6).

Mapa je trojdimenzionální, tudíž při simulaci jednoho dronu můžeme pracovat i s jeho výškou nad zemí. Příklad takovéto mapy se nachází na obrázku 3.2. Vidíme, že mapa byla vygenerována za použití mřížkové struktury a výška objektů byla vygenerována náhodně. Při simulaci roje nebudeme s výškou dronů nad zemí pracovat, budeme tedy používat pouze dvou-dimenzionální mapu, čehož je docíleno pohledem na mapu shora. Příklad takovéto mapy zobrazuje obrázek 3.3. Vidíme, že bylo použito náhodné generování mapy, barevné kolečka představují jednotlivé drony.

Simulátor dále zaznamenává historii všech kolizí, a to jak dronu a překážky, tak dvou dronů. Vyhodnocení, zda ke kolizi došlo, probíhá v jednotlivých algoritmech řízení roje (viz sekce 3.4). Pokud při simulaci klesne vzdálenost mezi dvěma drony pod dvojnásobek poloměru velikosti dronu (dron je aproximován kruhem), resp. pod součet poloměru dronu a překážky, dojde



OBRÁZEK 3.3: Dvou-dimenzionální mapa s náhodným generováním objektů



OBRÁZEK 3.4: Ukázka vizualizace waypointů (menší barevné kroužky) jednotlivých dronů a unikajícího plynu (modrý oblak, sytost modré barvy určuje koncentraci plynu)

ke kolizi. Ta je zaznamenána v historii kolizí, instance třídy *Swarm*, *collisions_history*⁹.

Do simulátoru byla dále přidána funkce zobrazení aktuálního waypointu jednotlivých dronů (obrázek 3.4), nebo pokud roj cestuje jako celek waypointu roje. Waypointy dronů jsou generovány ve fázi hledání lokace úniku plynu algoritmem PSO (kapitola 4) a jejich vizualizace může být užitečná pro ověření funkčnosti algoritmů simulátoru, popř. jejich ladění. Tuto funkci můžeme aktivovat parametrem `SWARM_VIEWER_DRAW_WAYPOINTS`, který se nachází na začátku hlavního souboru simulátoru `swarm_main`.

Nakonec byla do simulátoru přidána simulace unikajícího plynu. Lokaci centra úniku je možné pevně zvolit, nebo ji nechat náhodně vygenerovat. Šíření plynu pak modeluje pravděpodobnostní algoritmus, popsany na následujících

⁹Historie kolizí je vektor $t \times 3$, kde t je počet proběhlých iterací během simulace. V prvním sloupci se nachází počet kolizí mezi drony, v druhém počet kolizí mezi dronem a překážkou, a na třetí pozici minimální vzdálenost mezi dronem a překážkou, přičemž každý řádek odpovídá dané iteraci

řádcích. V centru úniku je inicializovaná maximální koncentrace plynu (hodnota 1). Následně se koncentrace generuje směrem od centra po zvětšujících se vrstvách. Její hodnota se odvíjí od nejbližšího okolí právě generovaného bodu. Z tohoto okolí je vybrána maximální hodnota koncentrace plynu a generovanému bodu je s danou pravděpodobností přiřazena koncentrace stejné hodnoty, nebo hodnoty zmenšené o daný krok (krok je vypočten tak, aby měl výsledný oblak plynu přibližně zadanou požadovanou velikost). Vznikne tak oblak plynu, který má globální maximum koncentrace v centru úniku a nemá žádné jiné lokální maxima. Tato vlastnost je důležitá pro funkčnost algoritmu PSO. Příklad takto vygenerovaného oblaku plynu můžeme vidět na obrázku 3.4.

3.4 Algoritmy řízení pohybu roje

Algoritmy řízení pohybu roje zajišťují (při správném nastavení parametrů) požadované chování roje. Umožňují přesun roje jako celku na zvolenou lokaci. Při tomto přesunu zabráňují kolizím agentů s ostatními agenty nebo s překážkami nacházejícími se v prostředí (antikolizní vlastnosti), ale zároveň udržují kompaktní tvar roje. Umožňují však také přesun na zvolené individuální lokace, pro každého agenta¹⁰. Při tomto přesunu již roj nedrží kompaktní tvar, jsou však zachované antikolizní vlastnosti.

Simulátor SwarmLab nabízí dva takové algoritmy, a to Olfati-Saberův a Vásárhelyiův algoritmus. Oba algoritmy jsou založené na decentralizovaném přístupu, tedy pohyb jednotlivých agentů je ovlivňován pouze lokálními informacemi, získanými od jeho sousedů. Důvodem této volby je, že decentralizovaný přístup dělá systém snadno škálovatelný a robustní vůči selhání jednotlivce (Soria, Schiano a Floreano, 2020). V následujících sekcích je popsáno fungování obou algoritmů.

3.4.1 Vásárhelyiův algoritmus

Vásárhelyiův algoritmus je založen na základě jednoduchého modelu hejna. Výsledný vektor požadované rychlosti v_i^d , které má agent i v dané iteraci dosáhnout, je výsledkem součtu čtyř složek, které jsou popsány v následujících sekcích.

Odpudivá a přitažlivá složka

Odpudivá složka zabráňuje kolizím mezi agenty, naopak přitažlivá složka udržuje kompaktní tvar roje¹¹. Pro jejich výpočet slouží jednoduchý pružinový model, tedy lineární rychlostní vztah, závislý na vzdálenosti od centra (Vásárhelyi et al., 2018) s přepínací hodnotou r_0^{rep} , určující, zda se budou agenti odpuzovat či přitahovat:

$$v_{ij}^{\text{rep}} = \begin{cases} p^{\text{rep}}(r_0^{\text{rep}} - r_{ij}) \frac{r_i - r_j}{r_{ij}} & \text{pokud } r_{ij} < r_0^{\text{rep}} \\ p^{\text{rep}}(r_{ij} - r_0^{\text{rep}}) \frac{r_j - r_i}{r_{ij}} & \text{pokud } r_{ij} \geq r_0^{\text{rep}} \end{cases} \quad (3.1)$$

¹⁰Tato funkce byla do simulátoru přidána.

¹¹Tato složka byla přidána do simulátoru, v původním Vásárhelyiovo algoritmu se nevyskytuje.

kde v_{ij}^{rep} je požadovaná rychlost agenta i , kterou způsobuje odpudivá/přitažlivá složka agenta j , p^{rep} laditelný koeficient této složky, r_i poloha agenta i a r_{ij} eukleidovská norma¹² $\|r_i - r_j\|$ rozdílu vektorů r_i a r_j .

Celková odpudivá a přitažlivá složka, působící na agenta i , je pak počítána jako součet všech těchto složek od jednotlivých agentů, tedy:

$$v_i^{\text{rep}} = \sum_{j \neq i} v_{ij}^{\text{rep}} \quad (3.2)$$

Pokud drony necestují jako roj, tedy nemají společný cíl nýbrž každý vlastní, přitažlivá složka není do výsledné rychlosti započítávána, neboť není žádoucí, aby se drony v této fázi shlukovaly.

Složka zarovnání rychlosti

Složka zarovnání rychlosti synchronizuje pohyb, aby bylo dosaženo kolektivního hejnového chování, ale také slouží jako tlumicí médium, které snižuje samobuzené oscilace, vznikající v důsledku zpožděné reakce, například na odpuzování (Vásárhelyi et al., 2018). Vektor rychlosti v_{ij}^{frict} , který tvoří tuto složku, závisí na rozdílu vektorů rychlosti blízkých agentů.

Nejdříve nadefinujeme požadovanou funkci hladkého poklesu rychlosti v prostoru $D(r, a, p)$:

$$D(r, a, p) = \begin{cases} 0 & \text{pokud } r \leq 0 \\ rp & \text{pokud } 0 < rp < \frac{a}{p} \\ \sqrt{2ar - \frac{a^2}{p^2}} & \text{jinak} \end{cases} \quad (3.3)$$

kde r je vzdálenost mezi agentem a očekávaným bodem zastavení, a je parametr preferovaného zrychlení a p je parametr určující hranici dvou fází zpomalování.

Úkolem složky zarovnání rychlosti je předejít tomu, aby rozdíl rychlosti dvou agentů byl větší, než jaký umožňuje zmiňovaná funkce hladkého poklesu rychlosti v prostoru $D(r, a, p)$. Tedy aby sloužila jako jakýsi plánovač pohybu v čase, narozdíl od ostatních složek, které jsou postavené na vztazích momentální síly (Vásárhelyi et al., 2018). Požadovanou rychlost v_{ij}^{frict} agenta i vzhledem k agentu j získáme podle vztahů:

$$v_{ij}^{\text{frictmax}} = \max(v^{\text{frict}}, D(r_{ij} - r_0^{\text{frict}}, a^{\text{frict}}, p^{\text{frict}})) \quad (3.4)$$

$$v_{ij}^{\text{frict}} = \begin{cases} C^{\text{frict}}(v_{ij} - v_{ij}^{\text{frictmax}}) \frac{v_i - v_j}{v_{ij}} & \text{pokud } v_{ij} > v_{ij}^{\text{frictmax}} \\ 0 & \text{jinak} \end{cases} \quad (3.5)$$

kde C^{frict} je parametr složky zarovnání rychlosti, v^{frict} je mez rychlostní volatility, která umožňuje určitý rozdíl rychlostí, nezávisle na vzdálenosti mezi agenty, r_0^{frict} je parametr relativní vzdálenosti bodu zastavení agenta i před agentem j , p^{frict} a a^{frict} specifické hodnoty již zmiňovaných parametrů (rovnice (3.3)) a $v_{ij} = \|v_i - v_j\|$ je eukleidovská norma rozdílu vektorů rychlostí agenta i a j .

Podobně jako u předchozí složky, je celková složka zarovnání rychlosti působící na agenta i spočítána jako součet všech těchto složek od jednotlivých

¹²Eukleidovská norma z vektoru v o velikosti n je nadefinována jako: $\|v\| = \sum_i^n \sqrt{|v_i|^2}$

agentů, tedy:

$$v_i^{\text{frict}} = \sum_{j \neq i} v_{ij}^{\text{frict}} \quad (3.6)$$

Poznamenejme, že horní index „frict“ („friction“, tedy tření) pochází z konceptu, že zarovnání rychlosti by mělo být lokální složkou tlumící rychlost, analogicky k viskóznímu tření (Vásárhelyi et al., 2018.)

Složka zajišťující vyhýbání se překážkám

Jak bylo popsáno dříve (sekce 3.3), překážky v simulovaném prostředí tvoří konvexní objekty (kruhy). Zabraňování kolizím s těmito objekty je dosaženo podobným principem, jakým je dosaženo zarovnání rychlosti popsané v předešlé sekci. Na pozicích překážek¹³ jsou generováni virtuální agenti s virtuální rychlostí v_i^{shill} , směřující směrem k agentovi i , pro kterého je složka vyhýbání se překážkám počítána. Pomocí takto vygenerovaných virtuálních agentů můžeme použít předchozí rovnice (3.4) až (3.6), s tím rozdílem, že nebudeme počítat zarovnání rychlosti reálného agenta s ostatními, ale zarovnání rychlosti reálného agenta s virtuálními agenty, kteří svojí rychlostí směřující z překážek nutí reálného agenta před překážkami brzdit a vyhýbat se jim. Pro tyto vztahy existují analogicky nastavitelné parametry, jako pro složku zarovnání rychlosti, místo horního indexu *frict* je použit index *shill* (lze přeložit jako návnada), tedy například v_i^{shill} .

Zároveň je žádoucí, aby agenti neopouštěli předem vymezenou simulační oblast. V původním Vásárhelyiovo algoritmu jsou na hranici této oblasti vygenerované umělé zdi. Udržení agentů v této oblasti je pak dosaženo podobným způsobem, jako vyhýbání se překážkám, tedy generováním virtuálních agentů. V této práci je však použit jednodušší způsob, nepředpokládáme, že je oblast ohraničena zdí (pokud by tomu tak bylo, mohla by být reprezentována souvislou řadou kruhových překážek), nýbrž pouze požadujeme, aby agenti neopouštěli simulační oblast. Toho je dosaženo tak, že požadované pozice, na které se mají agenti přesouvat (kapitola 4), jsou generovány pouze uvnitř simulační oblasti. Agent tedy může opustit simulační oblast, například pokud se vyhýbá překážce, ale po chvíli se do ní zase vrátí.

Složka zajišťující přibližování se k cíli

Poslední složka zajišťuje pohyb agentů směrem k požadovanému cíli. Generuje požadovanou rychlost v_i^{goal} agenta i , která je počítána podle:

$$v_i^{\text{goal}} = p^{\text{goal}} \frac{w_i - r_i}{\|w_i - r_i\|}, \quad (3.7)$$

kde w_i je aktuální waypoint agenta i , tedy požadovaný cíl a p^{goal} nastavitelný parametr této složky.

¹³Resp. na pozici uvnitř překážky, s nejmenší vzdáleností k agentovi i , ke kterému je složka zajišťující vyhýbání se překážkám počítána

Výsledná požadovaná rychlost

Výsledná, nezredukováaná, požadovaná rychlost \tilde{v}_i^d , které by měl agent i dosáhnout, je rovna součtu všech zmiňovaných složek:

$$\tilde{v}_i^d = v_i^{\text{rep}} + v_i^{\text{frict}} + v_i^{\text{shill}} + v_i^{\text{goal}} \quad (3.8)$$

Po této superpozici zavádíme mezní hodnotu v^{max} , která zachovává směr požadované rychlosti, ale snižuje její velikost, pokud je nad limitem (Vásárhelyi et al., 2018):

$$v_i^d = \frac{\tilde{v}_i^d}{\|\tilde{v}_i^d\|} \cdot \min\left(\|\tilde{v}_i^d\|, v^{\text{max}}\right) \quad (3.9)$$

Výsledkem je požadovaná rychlost v_i^d agenta i .

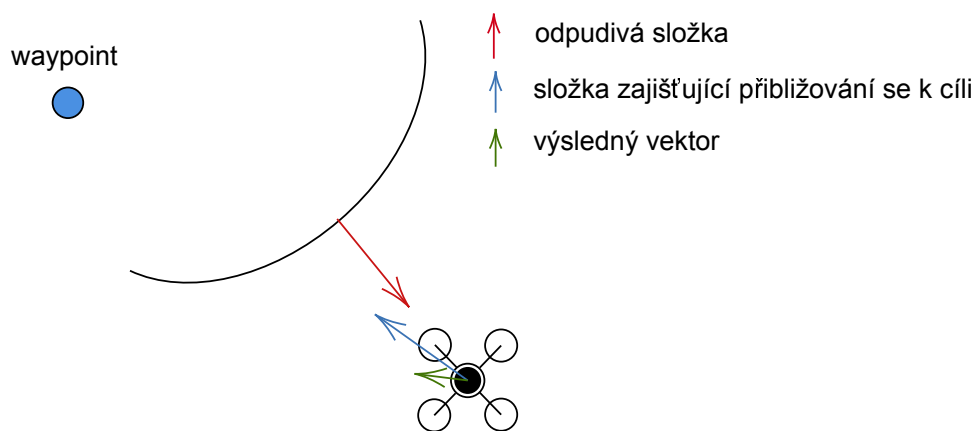
3.4.2 Olfati-Saberův algoritmus

Olfati-Saberův algoritmus je představen v článku *Flocking for multi-agent dynamic systems: algorithms and theory*, zveřejněném v roce 2006. Představuje zde tři decentralizované algoritmy řízení roje, které vedou k samoorganizujícímu se hejnovému chování. Tyto algoritmy fungují na bázi potenciálních polí a teorii grafů (Olfati-Saber, 2006). Jsou založené na konstrukci *kolektivního potenciálu*, který penalizuje odchylky agentů od požadovaného tvaru mřížky. Také zavádí *konsenzuální složku*, která umožňuje agentům dohodnout se na směru jejich pohybu a rychlosti (Soria, Schiano a Floreano, 2020). Třetí prezentovaný algoritmus, ve zmiňovaném článku, pak zavádí schopnost vyhýbaní se překážkám. Ta je založena na podobném principu, jako v předešlém algoritmu, tedy generování virtuálních agentů, na pozici překážek. Třetí jmenovaný algoritmus je v simulátoru implementován.

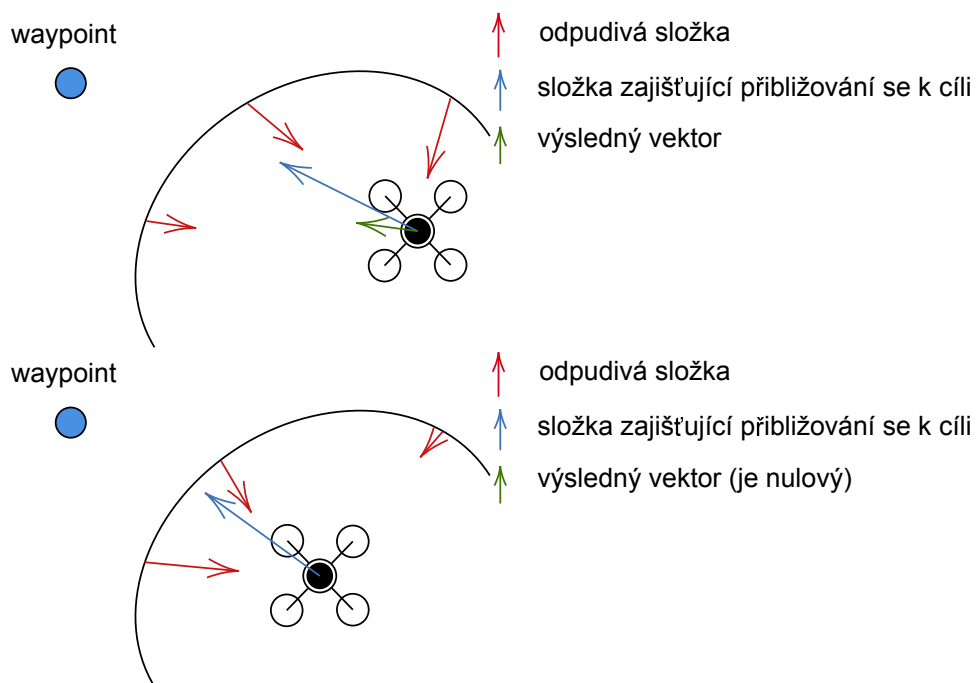
3.4.3 Problém vyhýbaní se konkávním překážkám

Jak bylo v předešlých sekcích zmíněno, oba algoritmy řízení pohybu roje implementují schopnost vyhýbaní se objektům podobným způsobem. Překážky, jimiž jsou kruhové objekty, nahrazují virtuálními agenty, kteří „odpuzují“ reálné agenty a tím předcházejí srážkám reálných agentů s překážkami. Tento způsob funguje pro překážky konvexního tvaru, resp. pro skupiny konvexních překážek tvořící dohromady konvexní objekt. Agent vždy působením zbylých sil, vystupujících v obou algoritmech, překážku/objekt obletí (obrázek 3.5). Problém nastává u překážek konkávního tvaru. Pro ně může nastat situace, kde zbylé síly nedokáží agenta z „konkávního údolí“ dostat a agent tak zůstane stát na místě, neschopen dosáhnout požadovaného cíle (obrázek 3.6). Tento problém lze těžko řešit modifikací zmíněných algoritmů, vyhýbaní se překážkám je zde řešeno takovým způsobem, který je principálně nemožný aplikovat pro konkávní objekty.

V této práci zmiňovaný problém neřešíme, generované překážky jsou konvexního tvaru a možnost vytvoření konkávní překážky, spojením několika překážek, je předcházeno minimální vzdáleností, kterou musí mít hranice dvou překážek, při generování prostředí (sekce 3.3).



OBRÁZEK 3.5: Náčrt případu obletění konvexní překážky



OBRÁZEK 3.6: Náčrt případu nemožnosti obletění konkávní překážky

3.5 Parametry simulátoru

Kromě dvou základních tříd simulátoru, popsaných v předchozích sekcích, jsou jeho další podstatnou částí soubory s parametry. Pomocí nich lze upravovat nastavení chodu simulace, generování prostředí, funkci algoritmů řízení pohybu roje a podobně. Většinu proměnných a parametrů zmiňovaných v předchozích sekcích můžeme najít a nastavit právě zde. Složku s parametry s názvem `parameters` najdeme v hlavní složce simulátoru. Následující tabulky poskytují přehled o parametrech a proměnných simulátoru, které byly zmíněny v textu nebo byly použity při simulacích.

Symbol/Název	Název v simulátoru	Popis
p^{goal}	<code>p_swarm.v_ref</code>	parametr složky zajišťující pohyb agentů směrem k cíli (Vásárhelyio algoritmus)
v^{max}	<code>p_swarm.max_v</code>	limitní celková rychlost (Vásárhelyio algoritmus)
počet dronů	<code>p_swarm.nb_agents</code>	počet dronů tvořící roj
waypoint roje	<code>p_swarm.goal_of_swarm</code>	úvodní cíl roje, kterého se snaží dosáhnout jako celek
rádus působení	<code>p_swarm.r</code>	maximální rádus okolo dronu, ve kterém může být ovlivněn ostatními ¹⁴
poloměr dronu	<code>p_swarm.r_coll</code>	rádus okolo dronu sloužící k detekci kolizí mezi dvěma drony
pozice roje	<code>p_swarm.P0</code>	počáteční pozice roje

TABULKA 3.1: Parametry roje (soubor `param_swarm.m`)

¹⁴Dron komunikuje, tedy je ovlivněn pouze ostatními drony, nacházejícími se v tomto rádiu. Tato skutečnost odpovídá realitě, kde podstata výhody decentralizovaného přístupu řízení skupiny spočívá právě v nepotřebě komunikace jednoho dronu se všemi ostatními. Díky tomu, jak v reálných podmínkách, tak v simulačním prostředí, je urychlena komunikace mezi drony, resp. teoreticky rychlost komunikace nezávisí na počtu dronů v roji.

Název	Název v simulátoru	Popis
velikost mapy	map.width	velikost strany čtverce tvořícího mapu
zobrazení plynu	map.ACTIVE_GASS	parametr určující vykreslení plynu
centrum plynu	map.gass_center	pozice centra unikajícího plynu, 2×2 vektor
poloměr plynu	map.gass_radius	rádus okolo centra unikajícího plynu, ve kterém ho jsou drony schopny detekovat
počet objektů	map.nb_blocks	počet vygenerovaných objektů, pokud jsou generovány pravidelně, je vygenerován čtverec se stranou rovnou zaokrouhlené odmocnině z požadovaného počtu
zastavenost	map.street_width_perc	při pravidelném generování objektů určuje procento nezastavené plochy ve čtverci s objekty
min. vzd. obj.	map.min_dst_between_objects	při náhodném generování objektů určuje minimální vzdálenost mezi pláštěmi objektů

TABULKA 3.2: Parametry mapy (soubor param_map.m)

Symbol/Název	Název v simulátoru	Popis
p^{rep}	p_swarm.p_rep	parametr odpudivé složky
r_0^{rep}	p_swarm.r0_rep	přepínací hodnota odpudivé složky mezi stavy přitahování/odpuzování
C^{frict}	p_swarm.C_fric	parametr složky zarovnání rychlosti
r_0^{frict}	p_swarm.r0_fric	parametr relativní vzdálenosti bodu zastavení agenta i před agentem j
p^{frict}	p_swarm.p_fric	parametr určující hranici dvou fází zpomalování
a^{frict}	p_swarm.a_fric	parametr preferovaného zrychlení
v^{frict}	p_swarm.v_fric	mez rychlostní volnosti, která umožňuje určitý rozdíl rychlostí nezávisle na vzdálenosti mezi agenty
r_0^{shill}	p_swarm.r0_shill	parametr relativní vzdálenosti bodu zastavení agenta i před virtuálním agentem j (složka vyhýbaní se překážkám)
p^{shill}	p_swarm.p_shill	parametr určující hranici dvou fází zpomalování
a^{shill}	p_swarm.a_shill	parametr preferovaného zrychlení
v^{shill}	p_swarm.v_shill	mez rychlostní volnosti, která umožňuje určitý rozdíl rychlostí nezávisle na vzdálenosti mezi agentem a virtuálním agentem

TABULKA 3.3: Parametry Vásárhelyiovo algoritmu řídicího pohyb roje (soubor param_vasarhelyi.m)

Název	Název v simulátoru	Popis
Perioda simulace	<code>p_sim.dt</code>	perioda, se kterou simulace běží
Perioda debugu	<code>p_sim.dt_plot</code>	perioda obnovování dat v diagnostickém okně
Perioda videa	<code>p_sim.dt_video</code>	perioda vizualizování simulace
Doba simulace	<code>p_sim.end_time</code>	čas, ve kterém je simulace ukončena

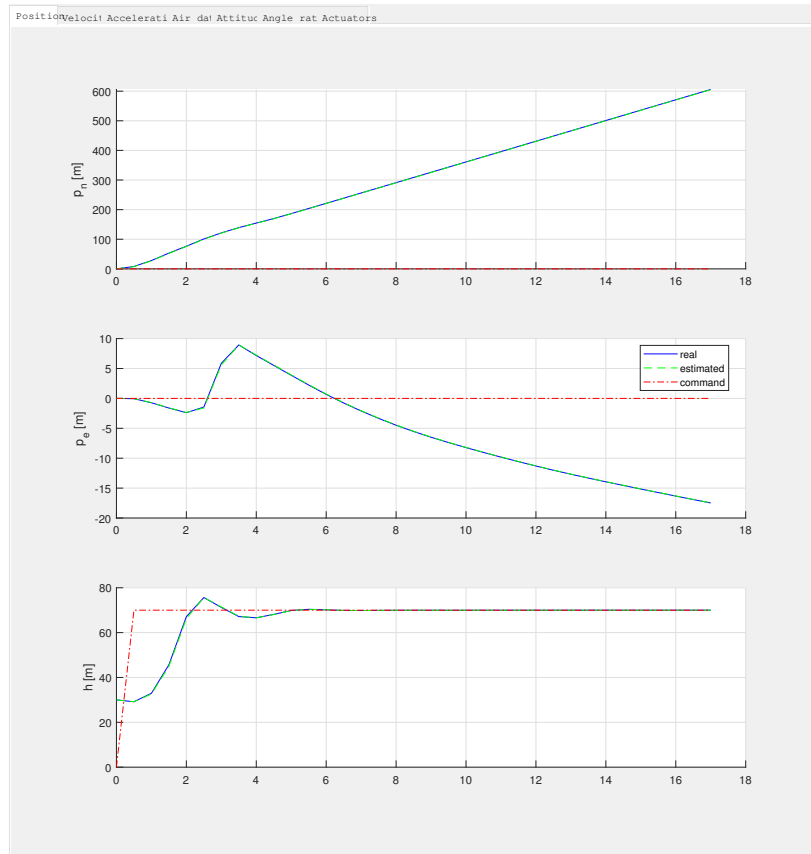
TABULKA 3.4: Parametry simulace (soubor `param_sim.m`)

3.6 Práce se simulátorem

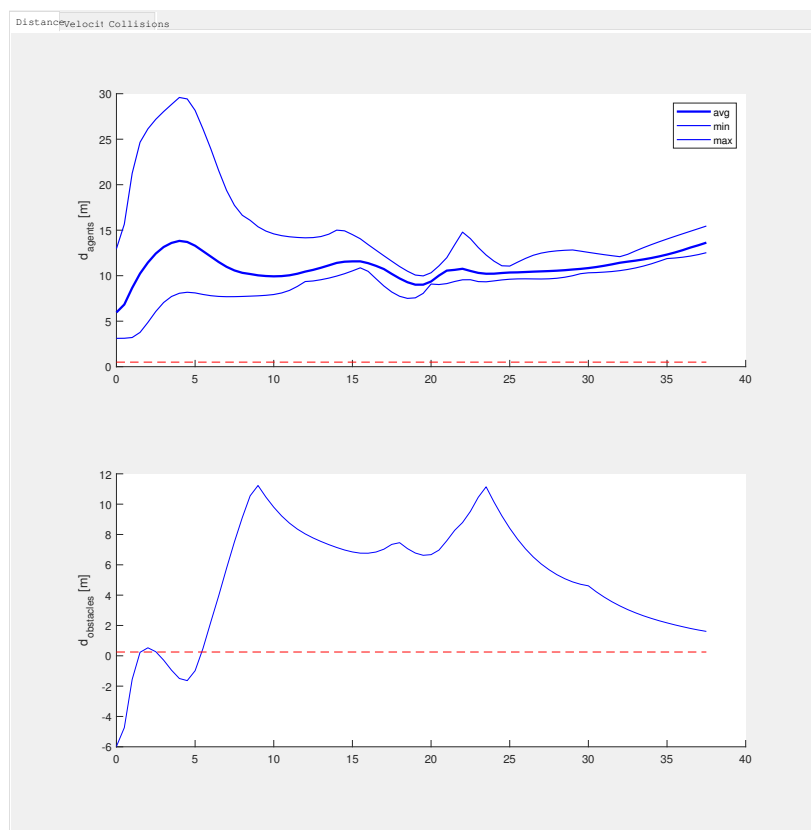
Pro spuštění simulace slouží složka `examples`, nacházející se v hlavní složce simulátoru. V té nalezneme připravené příklady prezentující funkce simulátoru. Nejdříve se podíváme na ty, které byly v simulátoru již zabudované. Ve složce `examples_drone` najdeme vzorové příklady funkcí simulátoru, týkající se simulace jednoho dronu. Jsou zde ukázány funkce jednotlivých vrstev navigačního modelu (sekce 3.1.1), například soubory `example_path_manager.m`, simulující fungování vrstvy **path_manager**. Průběh simulace můžeme sledovat pomocí vizualizace pohybu dronu prostředím, na výběr máme ale také možnost sledovat průběh některých proměnných třídy *Drone*, například pozici, rychlost, zrychlení, natočení dronu apod. (obrázek 3.7).

Pro simulaci roje použijeme soubor `swarm_main.m` ve složce `examples_swarm`, který je postavený na základě vzorových souborů pro práci s rojem. Tento soubor simuluje průběh vzorové úlohy lokalizace úniku plynu pomocí roje dronů, která je blíže popsána v kapitole 5. Na prvních řádcích souboru nalezneme nastavení různých možností simulace, například již zmiňované náhodné generování objektů v prostředí pod proměnnou `RANDOM_BUILDINGS`, ale i další, jako je zapnutí vizualizace prostředí pomocí proměnné `VIDEO` nebo volba řídicího algoritmu pohybu roje proměnnou `SWARM_ALGORITHM`. Stejně jako u třídy *Drone* i zde máme možnost kromě vizualizace roje sledovat průběh některých stavových veličin roje, například průměrnou, minimální a maximální vzdálenost mezi dvěma drony nebo průměrnou rychlost dronů (obrázek 3.8). Zobrazení tohoto diagnostického okna určuje proměnná `DEBUG`.

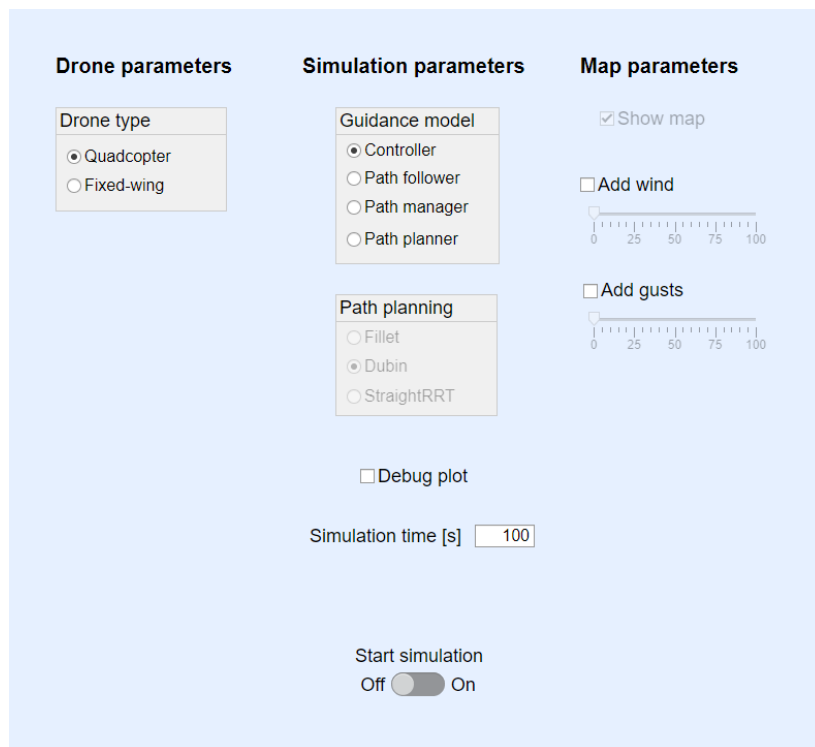
Druhou možností pro spuštění simulace je použití grafického uživatelského rozhraní (dále GUI). Pro simulaci dronu nabízí simulátor vestavené GUI s názvem souboru `GUI_drone.mlapp`, které můžeme vidět na obrázku 3.9. K simulaci roje pak nabízí GUI s názvem `GUI_swarm` (obrázek 3.10).



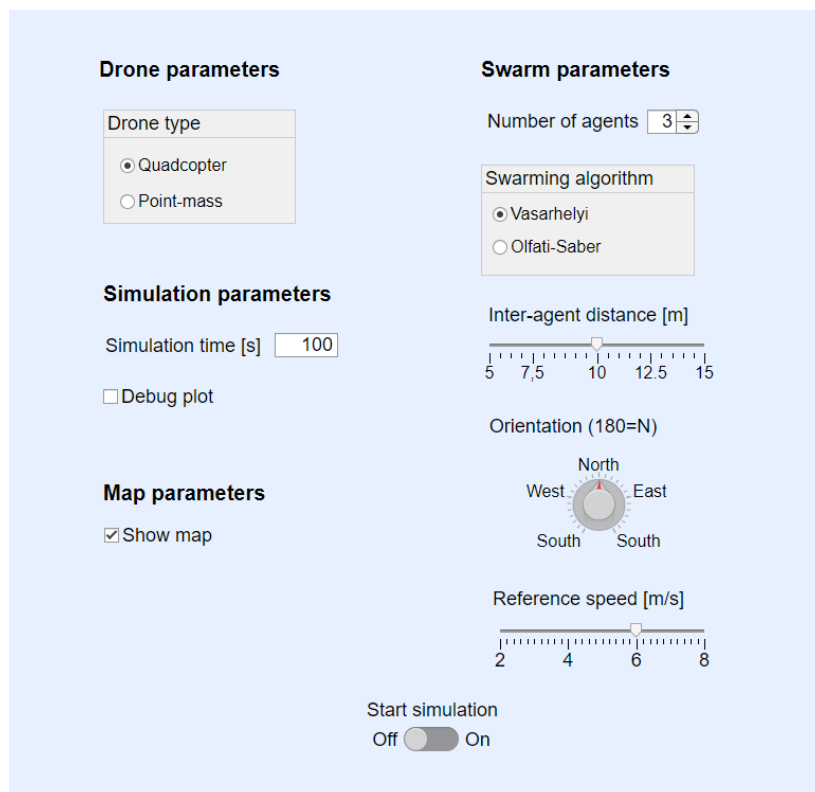
OBRÁZEK 3.7: Průběh stavových veličin dronu při simulaci



OBRÁZEK 3.8: Průběh stavových veličin roje při simulaci



OBRÁZEK 3.9: GUI dronu



OBRÁZEK 3.10: GUI roje

4 Particle swarm optimization

V sekci 3.4 byly zmíněny a popsány algoritmy řídící pohyb roje na základě souřadnic bodů, které mají jednotlivý agenti/celý roj následovat. Tyto informace přebírají zmiňované algoritmy na vstupu a na jejich základě generují příkazy pro pohyb agentů. Nyní potřebujeme algoritmus, který bude tyto body generovat za účelem plnění daného cíle, v našem případě lokalizaci unikajícího plynu. Použijeme proto algoritmus PSO.

particle swarm optimization (česky Optimalizace rojem částic, dále PSO) je stochastický algoritmus používaný pro optimalizační problémy, navržený Kennedym a Eberhartem v roce 1995. Tento algoritmus je možné použít v úlohách, ve kterých se snaží skupina částic/agentů najít specifický cíl, s předem neznámou lokací. Algoritmus simuluje chování ptačího hejna (Jatmiko, Sekiyama a Fukuda, 2007). Uvažujme následující situaci: ptačí hejno hledá potravu ve specifické oblasti. Pro algoritmus je důležité, aby se v oblasti nacházelo pouze jedno místo s potravou. Jaká je pak nejlepší strategie pro hledání potravy? Nejúčinnější strategií je sledovat ptáka, který je nejbližší k jídlu.

4.1 Stručná historie PSO

Agenti v počítačových modelech jsou decentralizované entity, které obecně nejsou schopny vnímat cíl na vyšší úrovni, ale přesto mohou modelovat složité systémy reálného světa (Sengupta, Basak a Peters, 2019). To je umožněno zadáním několika nízkoúrovňových cílů. Splněním těchto cílů mohou zdánlivě neinteligentních a neovlivňujících se agenti dosáhnout smysluplného kolektivního chování.

První použití tohoto principu lze pozorovat u Reevesova zavedení agentových systémů, v souvislosti s modelováním přírodních objektů, jako je oheň, mraky a voda, v počítačových animacích pro film v roce 1983. Následně Reynolds v roce 1987 stanovil agentům jednoduchá nízkoúrovňová pravidla, která vyvolala požadované kolektivní chování. Reynolds formuloval tři základní zásady již zmiňovaného flockingu, slukování se, kterými se agenti řídí: separace, zarovnání a soudržnost.

Zásada separace umožňuje agentům vzdalovat se od sebe, čím předchází přílišnému shlukování. Naopak zásada zarovnání vynucuje pohybu jednotlivých agentů směrem k průměrnému aktuálnímu směru pohybu všech agentů, zásada soudržnosti pak k průměrné pozici všech agentů. Inherentní náhodné chování agentů vede k chaotickému chování skupiny, naopak zmíněná jednoduchá nízkoúrovňová pravidla vedou k nařízenému chování skupiny.

V případě, kdy každý agent zná polohu všech ostatních agentů, vede na složitost $\mathcal{O}(n^2)$ ¹. Takováto náročnost je výpočetně velmi obtížná. Reynolds proto navrhl model s výměnou informací pouze se sousedními agenty v omezeném okolí, čímž se snížila složitost na $\mathcal{O}(n)$ a urychlila se tak implementace algoritmu.

PSO algoritmus byl následně uveden v roce 1995 Eberhartem a Kennedym jako rozšíření Reynoldsovo práce.

4.2 Algoritmus PSO

Neexistuje žádný standartní předpis PSO. Jeho jednotlivé verze se liší v implementaci rovnice (4.1), uvedeme tedy verzi předpisu algoritmu, která je použita v této práci. Popisují ji následující rovnice (4.1) a (4.2):

$$\begin{aligned} \mathbf{V}_i(t) &= \omega \mathbf{V}_i(t-1) + c_1 \text{rand}() (\mathbf{p}_i(t-1) - \mathbf{x}_i(t-1)) \\ &\quad + c_2 \text{rand}() (\mathbf{s}(t-1) - \mathbf{x}_i(t-1)) \end{aligned} \quad (4.1)$$

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{V}_i(t) \quad (4.2)$$

kde $\mathbf{x}_i(t)$ je poloha i -tého agenta v čase t (jak v reálném, tak v simulačním prostředí probíhá výpočet algoritmu v diskretním čase, dále používaný výraz $(t-1)$ tedy odpovídá časovému okamžiku $(t-\tau)$, kde τ je perioda, se kterou je algoritmus počítán), ω skalár nazývaný setrvačná váha, c_1 a c_2 skaláry ovlivňující chování agenta, $\text{rand}()$ je funkce vracující náhodné číslo v intervalu $(0,1)$ ², $\mathbf{p}_i(t-1)$ vektor uchovávající souřadnice bodu, ve kterém agent i zaznamenal maximum/minimum zadané ohodnocovací funkce³ v čase $(t-1)$ a $\mathbf{s}(t-1)$ vektor uchovávající souřadnice bodu, ve kterém zaznamenal maximum/minimum zadané ohodnocovací funkce celý roj, tedy maximum z maxima všech agentů, v čase $t-1$.

Jak rovnice (4.1) a (4.2) ukazují, požadovaná poloha agenta $\mathbf{x}_i(t)$, v čase t , se odvíjí od jeho původní polohy $\mathbf{x}_i(t-1)$ a vektoru $\mathbf{V}_i(t)$, který si lze představit jako vektor rychlosti, který se skládá ze tří složek.

První složkou je setrvačná složka ve tvaru: $\omega \mathbf{V}_i(t-1)$. Zajišťuje, aby se směr pohybu agentů příliš neměnil, v důsledku čehož se agenti pohybují plynuleji. Tato vlastnost je důležitá, jak v teoretické oblasti, kde zajišťuje lepší schopnost prozkoumávání okolí, tak v praktické oblasti, kde je plynulý pohyb agenta vhodnější než fragmentovaný pohyb, který má v jednotlivých časových okamžicích, ve kterých je algoritmus PSO počítán, výrazně jiný směr.

Druhou a třetí složkou, často nazývaných jako kognitivní a sociální faktor, je pohyb ve směru nalezeného maxima jednotlivého agenta/celého roje, ve

¹Tzv. velká \mathcal{O} notace vyjadřuje operační náročnost algoritmu. $\mathcal{O}(f(n))$ znamená, že náročnost algoritmu je menší než $A + B \cdot f(n)$, kde A a B jsou vhodně zvolené konstanty.

²Náhodné číslo generuje náhodná veličina s rovnoměrným rozdělením. To má ve všech bodech na daném intervalu konstantní hustotu pravděpodobnosti, to jest funkci, jejíž integraci na libovolném intervalu získáme pravděpodobnost, že hodnota náhodné veličiny bude náležet tomuto intervalu.

³V příkladové úloze, zmíněné v úvodu této kapitoly, to může být například vzdálenost od lokace s potravou. Zde by se tedy zjevně jednalo o hledání minima této funkce. V úloze řešené v této práci se jedná o koncentraci plynu (viz sekce 4.2.3), je zde tedy naopak hledáno maximum.

tvaru:

$$c_1 \text{rand}() (\mathbf{p}_i(t-1) - \mathbf{x}_i(t-1)) + c_2 \text{rand}() (\mathbf{s}(t-1) - \mathbf{x}_i(t-1))$$

Tyto složky jsou zjevně klíčové pro pohyb agentů směrem, kde se předpokládá lokace cíle. Volba parametrů c_1 a c_2 je jednou z možností, jak ladit funkci algoritmu PSO. Funkce `rand` slouží k diverzifikaci směrů jednotlivých agentů. Umožňuje tak lepší schopnost prozkoumávání prostoru. Jejím protipólem je již zmiňovaná setrvačná složka.

4.2.1 Omezení rychlosti

Správné nastavení mezí rychlosti V_{max} , tedy maximální možné velikosti vektoru rychlosti \mathbf{V} , je důležité pro chování roje. Při její absenci nebo nastavení na nevhodnou hodnotu, může chování roje divergovat. Vhodná volba maximální rychlosti V_{max} může sloužit také k ladění algoritmu PSO. Velká hodnota V_{max} vede ke globálnějším průzkumům okolí, zatímco malá hodnota znamená spíše místní, intenzivní průzkum (Sengupta, Basak a Peters, 2019).

Omezení je v této práci implementováno tak, že pokud eukleidovská norma, z vektoru rychlosti \mathbf{V}_{max} , překročí hodnotu V_{max} , je tento vektor znormován⁴ a následně vynásoben hodnotou V_{max} .

4.2.2 Setrvačná složka

Jak již bylo řečeno, setrvačná složka má předpis $\omega \mathbf{V}_i(t-1)$. Setrvačná váha ω slouží jako řídicí parametr pro rychlost roje. Umožňuje modulovat pohyb roje pomocí konstantní, lineárně časově proměnlivé nebo dokonce nelineárně časově proměnlivé závislosti (Sengupta, Basak a Peters, 2019). Setrvačná váha však nemůže zcela odstranit nutnost existence rychlostního omezení, popsaného v předešlé sekci. Je potřeba ji nastavit tak, abychom zajistili konvergentní chování roje a získali vhodný poměr, mezi prohledáváním nových oblastí a přibližováním se k domnělému cíli. Snížením hodnoty ω klesá schopnost roje prozkoumávat nové oblasti a umožňuje náhlejší změny ve směru pohybu jednotlivých agentů, z důvodu rostoucí role kognitivního a sociálního faktoru. Naopak s vyššími hodnotami ω získává roj větší schopnost prohledávání neprozkoumaných oblastí, a naopak ztrácí schopnost konvergovat k cíli. Jeví se tedy jako vhodné tuto hodnotu v čase měnit. Z počátku nastavit hodnotu ω na vyšší úroveň a následně ji v čase snižovat.

Jeden z možných způsobů, a zároveň způsob použitý v této práci, je časově lineární závislost $\omega(t)$. Tato závislost může být vyjádřena jako:

$$\omega(t) = (\omega_{max} - \omega_{min}) \frac{(t_{max} - t)}{t_{max}} + \omega_{min} \quad (4.3)$$

kde ω_{max} je maximální hodnota setrvační váhy, ω_{min} hodnota minimální a t_{max} maximální čas. Ten však v této úloze není známý. Simulace běží do té doby, dokud není nalezen zdroj úniku plynu. Je tedy zavedena podmínka $t = \min(t, t_{max})$, která zajišťuje, že hodnota $\omega(t)$ vždy náleží intervalu $\langle \omega_{min}, \omega_{max} \rangle$.

Do simulátoru SwarmLab je tato fáze doplněna metodou roje `pso_seeking`.

⁴Znormovaný vektor je vektor vydělen svojí vlastní normou.

4.2.3 Ohodnocovací funkce

Ohodnocovací funkce, která je součástí algoritmu PSO, se odvíjí od zadaného úkolu, který má roj řešit. Tím je, v našem případě, lokalizace unikajícího plynu. V reálných podmínkách by tedy drony byly vybaveny senzorem detekujícím unikající plyn, s očekávaným složením, a ohodnocovací funkce by se řídila daty z tohoto senzoru. V simulačním prostředí se tato funkce odvíjí od způsobu modelování unikajícího plynu. Ten se šíří prostředím způsobem popsáným v sekci 3.3. Ohodnocovací funkce přebírá vygenerovaný oblak plynu a jeho pozici v prostředí a přiřazuje jednotlivým dronům na základě jejich pozice hodnotu koncentrace plynu.

V simulátoru je tato funkce nadefinována metodou roje `compute_score`.

4.3 Prohledávací fáze

Algoritmus PSO je navrhnut tak, že je v celém prostoru známá hodnota dané ohodnocovací funkce. Tou je, v úloze řešené v této práci, míra koncentrace unikajícího plynu. Lze předpokládat, že tuto míru bude možné v reálných podmínkách měřit, pouze pokud překročí určitou mez. Znamená to tedy, že v části prohledávaného prostoru nebude možné určit hodnotu ohodnocovací funkce, resp. hodnota této funkce zde bude nulová. Pokud žádný z agentů nezaznamenal dosavad jinou hodnotu ohodnocovací funkce než nulovou, nachází se roj v tzv. prohledávací fázi. V této fázi je nutné implementovat algoritmus, který bude nahrazovat algoritmus PSO, do té doby, než jeden z agentů vstoupí do oblasti, kde míra ohodnocovací funkce překročí mezní hodnotu. Tuto fázi definují rovnice (4.4) a (4.5):

$$\mathbf{V}_i(t) = \omega_s \mathbf{V}_i(t-1) + c_s(2 \cdot \text{rand}() - 1) \quad (4.4)$$

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{V}_i(t) \quad (4.5)$$

Vidíme, že algoritmus je podobný jako algoritmus PSO, pouze kompenzuje absenci kognitivního a sociálního faktoru náhodnou složkou $c_s(2 \cdot \text{rand}() - 1)$. Ta generuje náhodný vektor ve čtverci se stranou rovnou parametru c_s . Setrvačná váha ω_s je v tomto případě konstantní, neboť je nežádoucí ji v čase měnit, protože cílem agentů v této fázi je prohledávat prostředí. Pokud některý z agentů zaznamená nenulovou hodnotu ohodnocovací funkce, začnou se všichni řídit algoritmem PSO.

Tato fáze je v simulátoru doplněna pomocí metody roje `pso_peeking`.

4.4 Parametry algoritmu PSO

Stejně jako u parametrů simulátoru SwarmLab, jsou nastavitelné parametry algoritmu PSO uloženy ve speciálním souboru. Nachází se ve složce `parameters` s názvem `param_pso.m` a do souvislosti s předcházejícím textem ho dává následující tabulka 4.1.

Symbol/Název	Název v simulátoru	Popis
ω	pso.we	setrvačná váha patřící fázi po nalezení první stopy plynu
ω_s	pso.ws	setrvačná váha patřící prohledávací fázi
c_1	pso.d	parametr složky mířící k dosavadnímu maximu dronu
c_2	pso.s	parametr složky mířící k dosavadnímu maximu roje
c_s	pso.r	parametr prohledávací fáze, strana čtverce uvnitř kterého jsou generovány waypointy
V_{\max}	pso.radius_seeking	maximální možná velikost vektoru $V_i(t)$ ve fázi po nalezení první stopy plynu
V_{\max_e}	pso.radius_exploring	maximální možná velikost vektoru $V_i(t)$ ve fázi prohledávání
Lineární ω	pso.linear_w	parametr zapínající lineární pokles váhy ω s časem
ω_{\max}	pso.w_max	maximální hodnota váhy ω
ω_{\min}	pso.w_min	minimální hodnota váhy ω
t_{\max}	pso.t_max	maximální čas, po který se váha ω mění

TABULKA 4.1: Parametry algoritmu PSO (soubor param_pso.m)

5 Testy

Tato kapitola obsahuje experimentální část práce. Budou laděny parametry řídicích algoritmů, popsaných v předešlých kapitolách, a následně bude ověřována jejich funkčnost.

5.1 Simulovaná úloha

Úlohou roje dronů, která je řešená v této práci, je lokalizace unikajícího plynu. Je tedy zapotřebí sestavit modelovou úlohu, jejíž řešení bude simulováno. Skládá se ze dvou fází, které popisují následující odstavce.

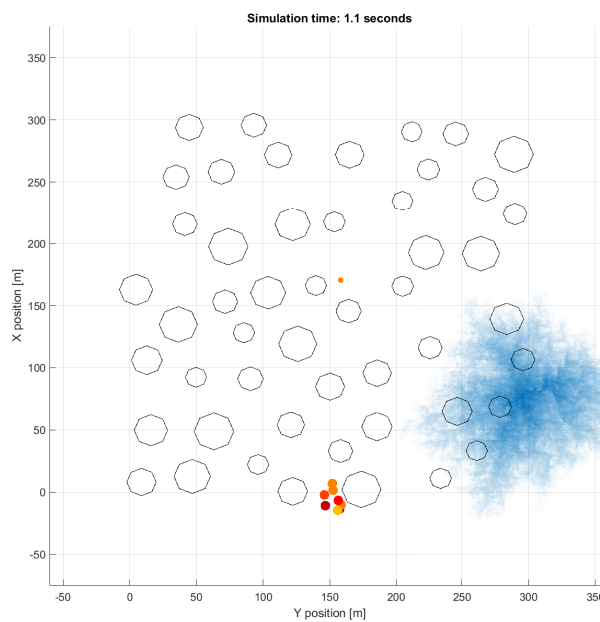
První fází je přesun roje. Testuje algoritmus řízení pohybu roje (sekce 3.4), tedy jeho schopnost přesunu jako celku na požadované místo. Přesun probíhá v prostředí s překážkami, je proto vyžadována eliminace kolizí mezi agenty a překážkami. Zároveň je požadován kompaktní tvar roje, nýbrž chceme aby se přesouval jako celek. Startovní pozice roje se nachází ve spodní části mapy s pomyslným prostorem s překážkami před sebou (obrázek 5.1). Jeho úkolem je dostat se do centra mapy. Je tak simulována situace, která by nastala v reálných podmínkách, kdy je potřeba dostat roj do objektu na předpokládané přibližné místo úniku plynu. Průběh této fáze zobrazuje obrázek 5.2. Druhou fází je samotná lokalizace unikajícího plynu. Algoritmus řízení pohybu roje zde slouží k přesunu dronů jako jednotlivců na požadované místa a ke generování waypointů je využit algoritmus PSO. Algoritmus PSO se skládá ze dvou částí, jak popisuje kapitola 4. První částí je hledání lokace s detekovatelnou koncentrací unikajícího plynu (obrázek 5.3), řídicí algoritmus této části popisuje sekce 4.3. Jakmile jeden z dronů detekuje unikající plyn, algoritmus přechází do druhé části (obrázek 5.4). Ta končí, a tím i simulační úloha, nalezením zdroje unikajícího plynu, tedy místa, s maximální koncentrací tohoto plynu (obrázek 5.5). Řídicí algoritmus druhé části je popsán v sekci 4.2.

5.2 Parametry simulací

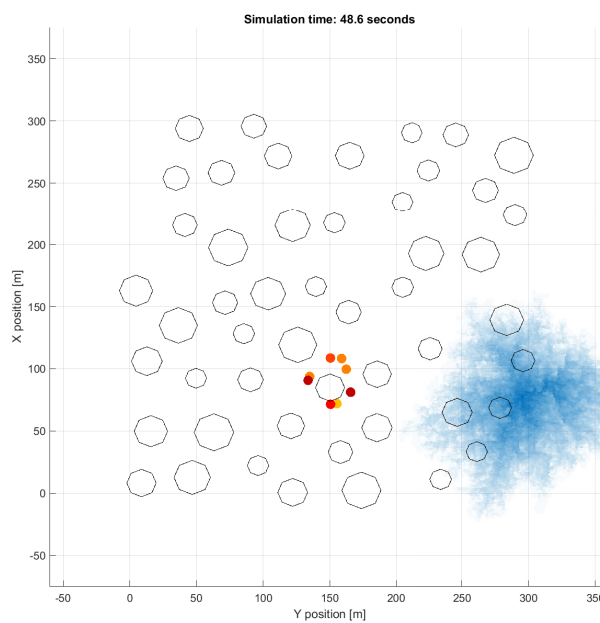
V této sekci je popsáno nastavení parametrů simulace, které nejsou dále trénovány, a jejich hodnota je tedy v následujících simulacích fixní.

Při představování třídy *Drone*, v sekci 3.1, byly zmíněny dva typy dronu, jež simulátor nabízí, a to *quadcopter* a *fixed-wing*. V této práci se věnujeme typu dronu kvadrokoptéra, dále proto budeme používat typ *quadcopter*, resp. typ *point mass*. Ten zanedbává dynamiku kvadrokoptéry a počítá pouze s dynamikou hmotného bodu. Toto zjednodušení značně snižuje výpočetní složitost simulace, která při simulování roje značně roste, aniž by nějak výrazně změnila výsledky experimentů.

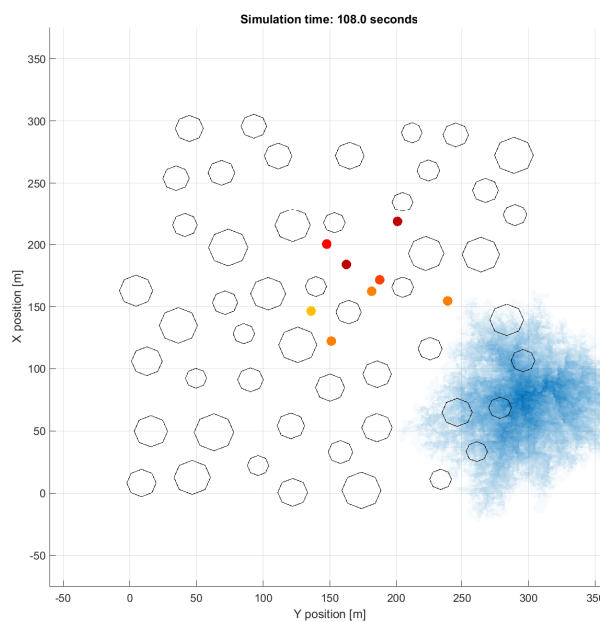
V následujících sekcích bude trénován Vásárhelyiův algoritmus, který slouží k řízení pohybu dronů (sekce 3.4.1).



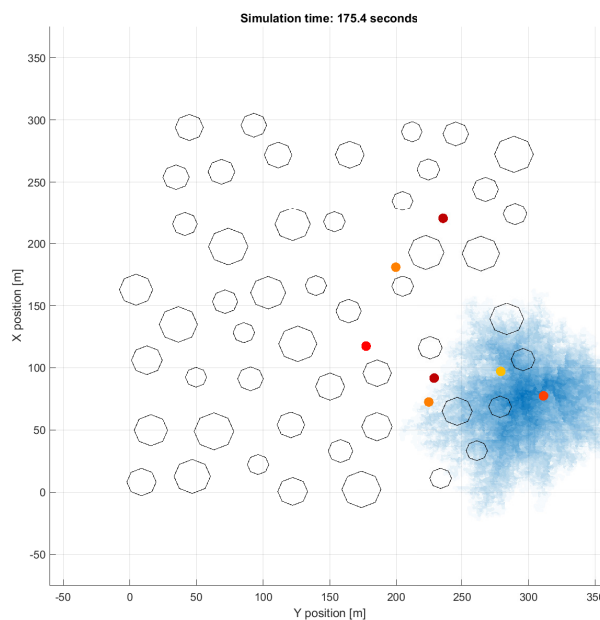
OBRÁZEK 5.1: Startovní pozice roje



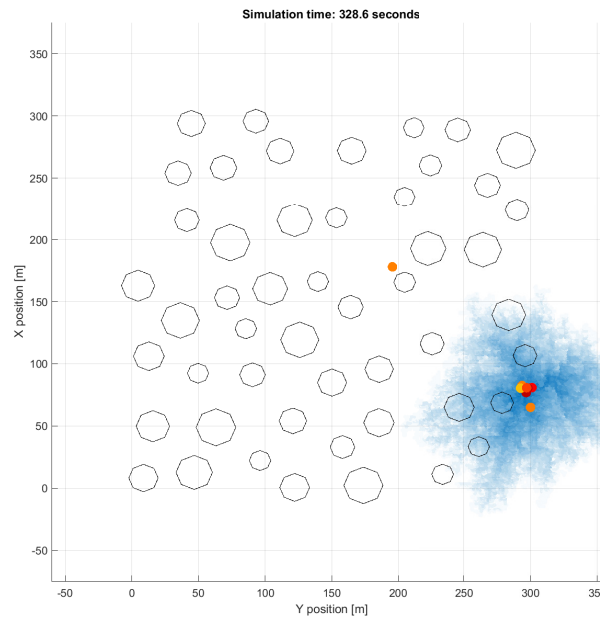
OBRÁZEK 5.2: Průběh první fáze simulační úlohy - přesun roje



OBRÁZEK 5.3: První část druhé fáze simulační úlohy - hledání lokace s detekovatelnou koncentrací unikajícího plynu



OBRÁZEK 5.4: Druhá část druhé fáze simulační úlohy - hledání zdroje unikajícího plynu



OBRÁZEK 5.5: Konec simulační úlohy - nalezení zdroje unikajícího plynu

Velikost mapy, tedy parametr `map.width`, je nastaven na hodnotu 300. Související parametr počtu objektů je zvolen na hodnotu 50. Je tak vytvořeno prostředí s velkým procentem zastavěné plochy, které omezuje pohyb dronů. Objekty jsou generovány náhodně, prostředí tak více odpovídá reálným podmínkám.

Roj se skládá z 8 dronů. Tento počet je zvolen s ohledem na výpočetní náročnost simulace (odvíjející se hlavně od tohoto parametru), která je klíčová pro použití genetického algoritmu, za účelem ladění parametrů. Počet dronů však nemůže být příliš nízký, kvůli zaručení funkčnosti algoritmu PSO.

Při ladění parametrů byl zvýšen poloměr dronu `p_swarm.r_coll` z hodnoty 0.5 na hodnotu 0.7. Díky vyšší hodnotě při trénování řídicí algoritmus lépe předchází kolizím, po snížení hodnoty zpět na původní, i v prostředích, pro která nebyl trénován.

Genetický algoritmus, popsáný v sekci 2.2.3, má dva nastavitelné parametry, jimiž jsou *pravděpodobnost křížení* a *pravděpodobnost mutace*. *Pravděpodobnost křížení* je nastavena na hodnotu 0.7, *pravděpodobnost mutace* je vyhodnocena podle předpisu $p_m = \frac{5}{p \cdot s}$, kde p je počet trénovaných parametrů a s je počet generovaných sad v jedné generaci. Průměrně tak dochází k mutaci jednoho parametru u každé páté sady nové generace. Genetický algoritmus a výpočet *fitness* funkce, pro jednotlivé trénovací fáze, najdeme ve složce `genetic_algo`.

5.3 Ladění parametrů

V této sekci budeme pomocí genetického algoritmu ladit parametry jednotlivých řídicích algoritmů. Počet sad a generací se bude v každé sekci lišit. Odvíjí se od počtu parametrů a náročnosti simulace. Například při vyšším počtu laděných parametrů je potřeba generovat více sad v rámci jedné generace pro dosažení dostatečné diverzifikace. Ladění probíhá na množině předem vygenerovaných map (mohutnost této množiny se liší dle trénovaného algoritmu), která je pro každou sadu parametrů stejná. Všechny sady parametrů tak mají při trénování stejné podmínky. Pro dosažení požadovaných kvalit chování dronů je klíčové správně zvolit *fitness* funkci. Do té je potřeba správně zvolit stavové proměnné simulace, které budou sledovány, a vzájemně vyvážit jejich vliv na hodnotu *fitness funkce*. Soubory, kterými můžeme spustit ladění parametrů, se nachází ve složce `examples_genetic_algo`.

5.3.1 Algoritmus řízení pohybu roje

Nejdříve naladíme parametry nižší vrstvy řízení pohybu dronu, tedy Vásárhelyiovo algoritmu. V simulované úloze se drony v první fázi pohybují jako celek, v druhé fázi jako jednotlivci. Tato chování se značně liší, proto naladíme sadu parametrů řídicího algoritmu pro každou fázi zvlášť. V simulacích úloze budeme následně mezi sadami přepínat.

Vásárhelyiovo algoritmus obsahuje velký počet laditelných parametrů, byl tedy zvolen vyšší počet sad v generaci, a to 30. Každá sada byla trénována na třech mapách. Simulace probíhá na jedné mapě po dobu 600 sekund.

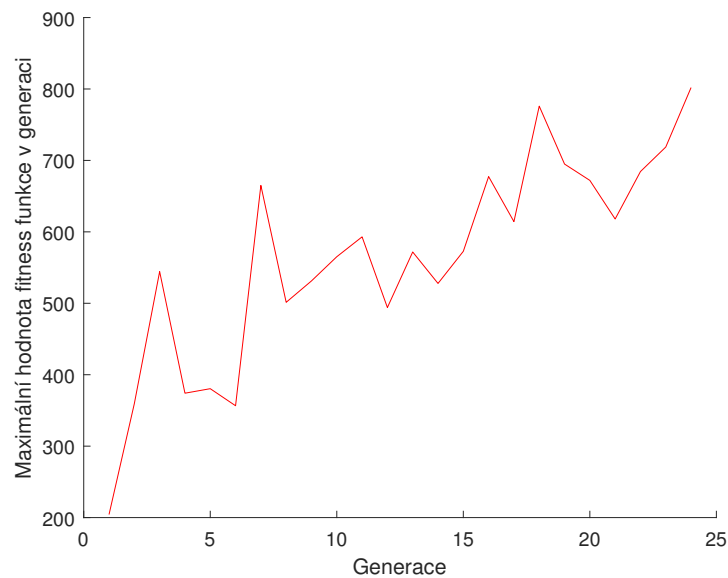
Pohyb dronů jako jednotlivců

Při pohybu dronů jako jednotlivců je z Vásárhelyiovo algoritmu vyloučena přitažlivá složka. Tato složka nemá laditelný parametr, resp. sdílí parametr se složkou odpudivou. Ladíme tedy všechny parametry tohoto algoritmu (tabulka 5.1). Drony jsou v testovací úloze, sloužící k určení *fitness* funkce, vypuštěny ze středu mapy. Každému z nich je vygenerovaný waypoint, kterého musí dosáhnout. Když tak učiní, je jim vygenerován waypoint nový. Waypointy nejsou generovány po celé mapě, ale pouze na menším prostoru, pro zhuštění koncentrace dronů na jednom místě za účelem lepšího natrénování parametrů. Takto drony „sbírají body“ do ukončení simulace. *Fitness* funkce má předpis:

$$f_{\text{dron}} = \frac{b}{1 + (\ln(c_o + 1)) \cdot (\ln(c_d + 1))}, \quad (5.1)$$

kde b je celkový počet dosažených waypointů, c_o je počet kolizí dronů s překážkami a c_d je počet kolizí dronů mezi sebou. Vidíme, že při nulovém počtu kolizí je hodnota *fitness* funkce rovna celkovému počtu dosažených waypointů. Pokud ke kolizím dojde, je tato hodnota snížena.

Naladěné parametry algoritmu zobrazuje tabulka 5.1 ve sloupci s názvem „jednotlivci“. Rozsah hodnot parametrů, ve kterém genetický algoritmus hledal jejich optimální nastavení, zobrazuje sloupec „rozsah“. Funkčnost genetického algoritmu si můžeme ověřit pomocí obrázku 5.6. Ten zobrazuje vývoj maxima *fitness* funkce v jednotlivých generacích. Vidíme, že hodnota funkce postupně stoupá, požadovaná kvalita trénovaného algoritmu se tedy s novými generacemi lepší. Podobným ukazatelem může být suma *fitness*



OBRÁZEK 5.6: Vývoj maxima hodnoty *fitness* funkce v jednotlivých generacích, při trénování pohybu dronů jako jednotlivců

funkcí v jednotlivých generacích (obrázek 5.7). Vývoj hodnoty jednoho z trénovaných parametrů zobrazuje obrázek 5.8.

Ladění parametrů této fáze spustíme souborem `train_drone.m`.

Pohyb dronů jako roje

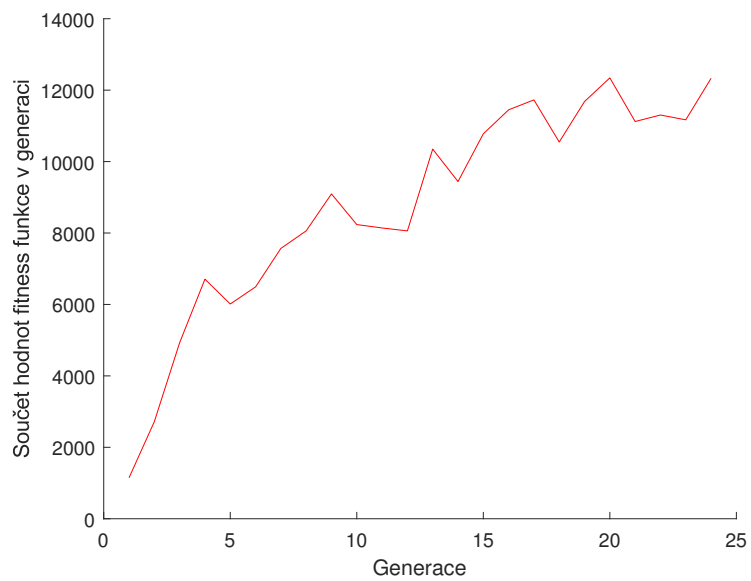
V této fázi je, narozdíl od předešlé fáze, započítávána přitažlivá složka řídicího algoritmu. Simulační úloha je obdobná předešlé fázi. Roj je vypuštěn ze středu mapy a má za úkol dosáhnout požadovaných waypointů. Do hodnoty *fitness* funkce se však promítá nová stavová proměnná, kterou je kompaktnost roje. Ta vyjadřuje zda se roj pohybuje prostředím jako celek. Počítána je jako průměrná vzdálenost dronů od pomyslného těžiště roje (průměr poloh dronů) během simulace. Výsledná hodnota *fitness* funkce sady i je

$$f_{roj_i} = f_{dron_i} \cdot \left(1 + \frac{\ln(k_{min})}{\ln(k_i)}\right), \quad (5.2)$$

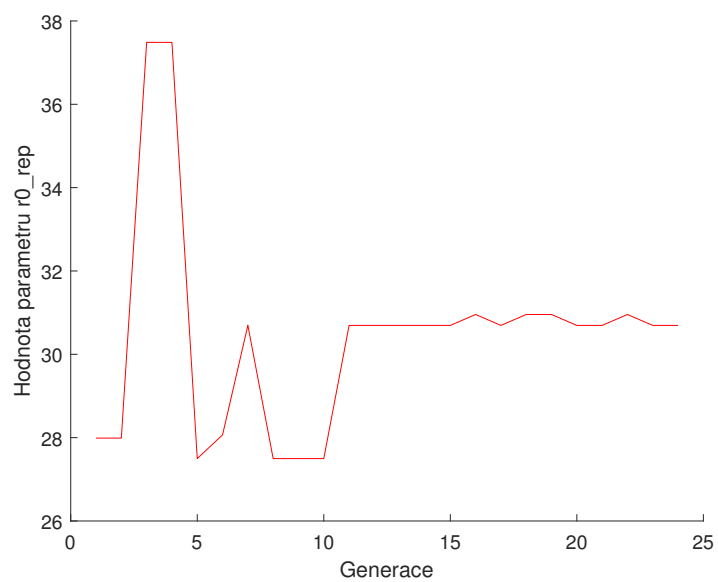
kde f_{dron_i} je hodnota části výsledné *fitness* funkce spočítané podle rovnice (5.1), k_{min} je minimální hodnota kompaktnosti roje ze všech sad jedné generace a k_i hodnota kompaktnosti roje sady i . Vidíme, že hodnota *fitness* funkce nezávisí pouze na stavových proměnných jedné sady, ale také na jejich porovnání s ostatními sadami.

Naladěné parametry algoritmu zobrazuje tabulka 5.1 ve sloupci s názvem „roj“. Vývoj maxima *fitness* funkce v jednotlivých generacích (obrázek 5.9) znovu naznačuje, že natrénování parametrů proběhlo úspěšně. Vývoj hodnoty jednoho z trénovaných parametrů zobrazuje obrázek 5.10.

Ladění parametrů této fáze spustíme souborem `train_swarm.m`.



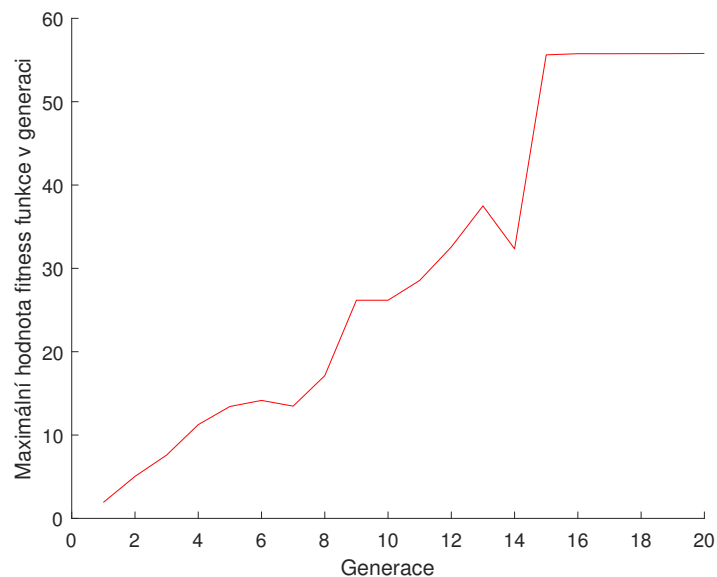
OBRÁZEK 5.7: Vývoj sumy *fitness* funkcí v jednotlivých generacích, při trénování pohybu dronů jako jednotlivců



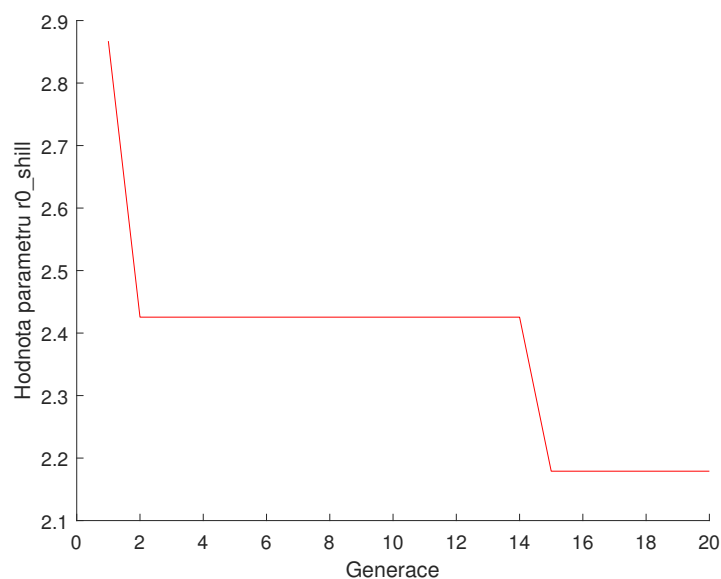
OBRÁZEK 5.8: Vývoj hodnoty parametru *r0_rep*, při trénování pohybu dronů jako jednotlivců

parametr	manuálně	rozsah	jednotlivci	roj
p_swarm.r0_rep	20	[1;50]	30,6936	46,4689
p_swarm.p_rep	0,1	[0;0,15]	0,1256	0,1179
p_swarm.r0_shill	1	[0;5]	0,8502	2,179
p_swarm.v_shill	13,6	[5;20]	8,9052	11,8292
p_swarm.p_shill	3,55	[0;8]	5,5358	4,4067
p_swarm.a_shill	5	[0;10]	7,1451	9,394
p_swarm.v_ref	6	[2;20]	4,6207	7,9394
p_swarm.r0_fric	85,3	[10;200]	119,7	119,1492
p_swarm.C_fric	0,05	[0;0,5]	0,1378	0,2323
p_swarm.v_fric	0,63	[0;3]	2,3962	0,0201
p_swarm.p_fric	3,2	[1;10]	6,2906	4,5322
p_swarm.a_fric	4,16	[1;10]	3,5514	9,9904

TABULKA 5.1: Manuálně a genetickým algoritmem natrénované parametry Vásárhelyiovo algoritmu



OBRÁZEK 5.9: Vývoj maxima hodnoty *fitness* funkce v jednotlivých generacích, při trénování pohybu dronů jako roje



OBRÁZEK 5.10: Vývoj hodnoty parametru `r0_shill`, při trénování pohybu dronů jako roje

5.3.2 Algoritmus PSO

Nyní naladíme parametry vyšší řídicí vrstvy, tedy algoritmu PSO. Tento algoritmus má dvě fáze, obě budeme tedy ladit zvlášť. U druhé fáze algoritmu můžeme použít konstantní hodnotu parametru ω , nebo hodnotu v čase lineárně měnit (viz sekce 4.2.2). Natrénujeme proto parametry druhé fáze pro obě možnosti.

Algoritmus PSO má méně laditelných parametrů. Proto byl počet sad v generaci snížen na 24 a počet generací na 10. Naopak první fáze vyžaduje více trénovacích map z důvodu dodržení funkčnosti i pro nová prostředí. Pro první fázi bylo použito 10 trénovacích map, ale pro druhou jen 5.

Trénovací úloha se shoduje s částí simulované úlohy. Při trénování první fáze algoritmu jsou drony vypuštěny ze středu mapy a jejich úkolem je lokalizovat první známku unikajícího plynu. Jakmile se to jednomu z nich podaří, simulace je zastavena a je zaznamenán aktuální čas simulace, který je následně použit jako *fitness* funkce sady. Trénování druhé fáze algoritmu probíhá obdobně. Nejdříve proběhne první fáze algoritmu (ta je navíc urychlena tím, že je zvětšen radius unikajícího plynu), přičemž jsou pro ni použity již natrénované parametry. Po nalezení stopy plynu je vynulován čas simulace a probíhá trénování fáze dvě, tedy lokace centra úniku plynu.

Konec simulace nastává po nalezení bodu úniku. Ten lze definovat jako místo, kde má koncentrace plynu hodnotu 1. Toto místo má velmi malou rozlohu, zpravidla jen několik bodů (souřadnic). Při konstantní hodnotě parametru ω se trénováním nezdařilo dosáhnout sady parametrů, která by tohoto bodu dosáhla v rozumném čase. Drony oscilují okolo přibližného bodu úniku, neschopny dosáhnout bodu s maximální koncentrací plynu. Kritérium pro zastavení simulace pro konstantní hodnotu ω bylo tedy sníženo na hodnotu koncentrace plynu o krok nižší (viz sekce 3.3).

Při použití lineárně časově proměnné hodnoty ω se již podařilo natrénovat

parametr	manuálně	rozsah	genetic. algo.
<code>pso.frequency</code>	5	[0,2;10]	3,134
<code>pso.we</code>	0,9	[0;3]	1,3792
<code>pso.r</code>	20	[0;50]	13,8951
<code>pso.radius_exploring</code>	60	[5;100]	95,4496

TABULKA 5.2: Manuálně a genetickým algoritmem natrénované parametry první fáze algoritmu PSO

parametr	manuálně	rozsah	genetic. algo.
<code>pso.ws</code>	0,5	[0;2]	1,4154
<code>pso.d</code>	0,8	[0;5]	3,7008
<code>pso.s</code>	2	[0;5]	3,6316
<code>pso.radius_seeking</code>	50	[5;100]	57,8644

TABULKA 5.3: Manuálně a genetickým algoritmem natrénované parametry druhé fáze algoritmu PSO, za použití konstantní hodnoty parametru ω

parametry algoritmu, který je schopný detekovat lokaci s maximální koncentrací plynu. Kritérium pro zastavení simulace tedy nebylo pro tuto variantu sníženo.

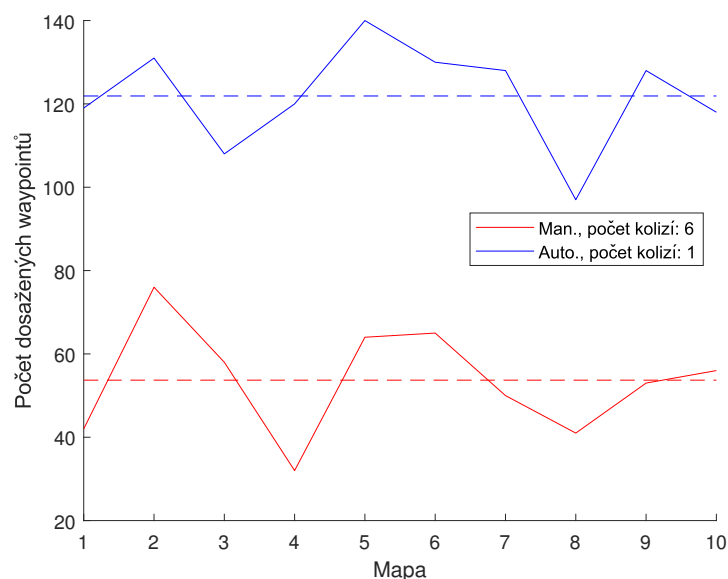
Naladěné parametry algoritmu zobrazují tabulky 5.2 až 5.4. Ladění parametrů první fáze algoritmu PSO spustíme souborem `train_pso_e.m`, druhé fáze s konstantním parametrem ω souborem `train_pso_s.m` a druhé fáze s lineárně časově proměnným parametrem ω souborem `train_pso_s_lin.m`.

5.4 Porovnání algoritmů

V této sekci porovnáme funkčnost řídicích algoritmů s manuálně a automaticky nastavenými parametry pomocí genetického algoritmu. Testovány budou na stejných úlohách, na jakých byly v předešlé sekci trénovány. Proběhnou však na jiné sadě map.

parametr	manuálně	rozsah	genetic. algo.
<code>pso.d</code>	0,8	[0;5]	1,3462
<code>pso.s</code>	2	[0;5]	1,459
<code>pso.radius_seeking</code>	50	[5;100]	40,6491
<code>pso.w_max</code>	3	[0,5;5]	1,5127
<code>pso.w_min</code>	0,5	[0;0,5]	0,2931
<code>pso.t_max</code>	100	[10;300]	92,2115

TABULKA 5.4: Manuálně a genetickým algoritmem natrénované parametry druhé fáze algoritmu PSO, za použití lineárně časově proměnné hodnoty parametru ω

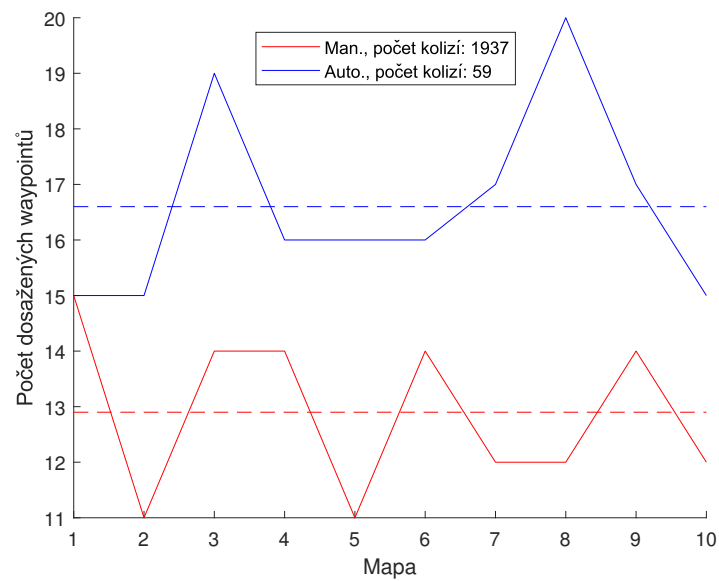


OBRÁZEK 5.11: Porovnání počtu dosažených waypointů drony, řízenými algoritmem s manuálně a automaticky nastavenými parametry

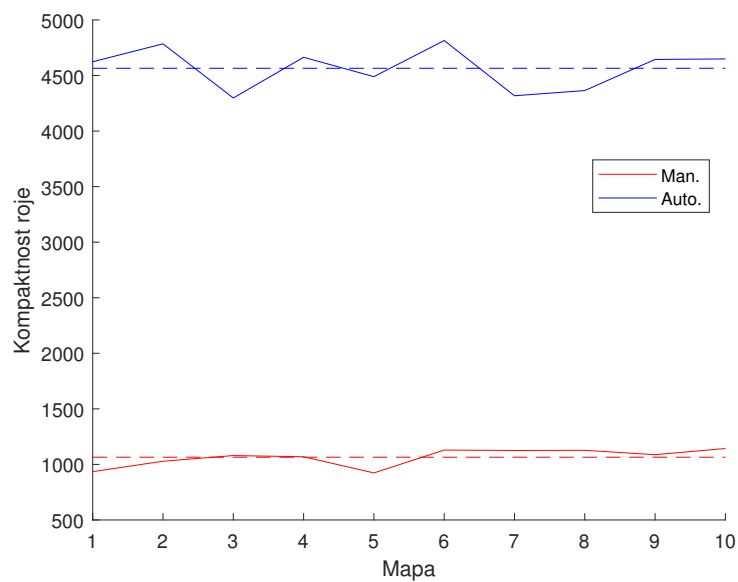
Nejdříve otestujeme pohyb dronů jako jednotlivců. Jak vidíme na obrázku 5.11, algoritmus s automaticky naladěnými parametry dosahuje lepších výsledků. Počet dosažených waypointů je vyšší a zároveň došlo k méně kolizím. Podobného výsledku bylo dosaženo při testování pohybu dronů jako roje. Vidíme, že roj, natrénovaný genetickým algoritmem, dosahuje obecně vyššího počtu dosažených waypointů a zároveň u něj dochází mnohonásobně méně ke kolizím (obrázek 5.12). Naproti tomu roj, řízený manuálně nastavenými parametry, cestuje kompaktněji (obrázek 5.13). Dále otestujeme jednotlivé fáze algoritmu PSO. Vyhodnocení první fáze vidíme na obrázku 5.14. Sady parametrů jsou zjevně podobně efektivní. Průměrně jsou zlehka lepší parametry naladěné automaticky. Podobného výsledku dosáhneme při testování samotné druhé fáze algoritmu PSO (obrázek 5.15) s konstantní hodnotou parametru ω . Vidíme, že algoritmus s automaticky naladěnými parametry potřeboval průměrně méně času k lokaci zdroje úniku plynu. Dobu trvání celého algoritmu PSO zobrazuje obrázek 5.16.

Nyní porovnáme použití konstantní a lineárně časově proměnné hodnoty parametru ω . Při tom využijeme automaticky naladěné parametry. Poznámemejme, že parametry algoritmu s lineárně časově proměnnou hodnotou ω , byly naladěny za účelem lokace maximální koncentrace unikajícího plynu. Zatímco parametry s konstantní hodnotou ω na o stupeň sníženou koncentraci. I přes to dosahují parametry s lineárně časově proměnnou hodnotou ω lepších výsledků, jak můžeme vidět na obrázku 5.17.

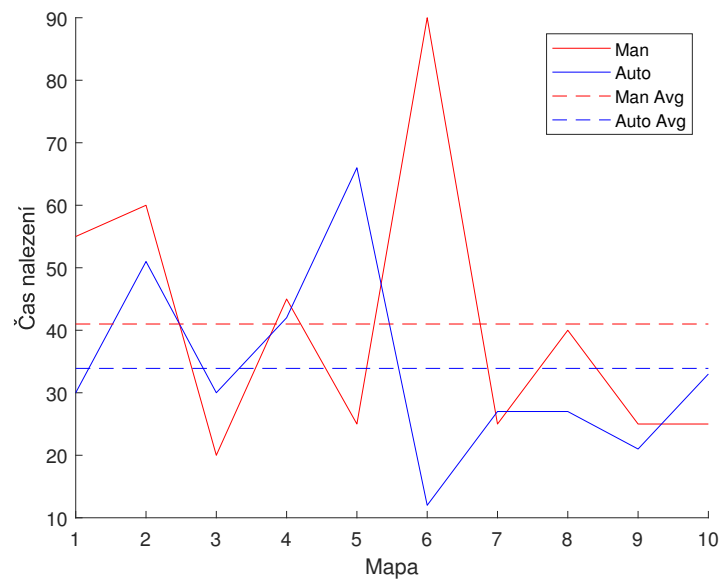
Nakonec otestujeme průběh celé simulační úlohy. Nejdříve porovnáme použití manuálně nastavených parametrů řídicích algoritmů s parametry naladěnými automaticky. Použita byla konstantní hodnota parametru ω . Obrázek 5.18 zobrazuje výsledek testu. Vidíme, že roj řízený algoritmy s automaticky naladěnými parametry dokončil simulační úlohu v kratším čase, vyjma jednoho případu. Průměrná doba, potřebná k dokončení úlohy, je téměř poloviční oproti výsledkům roje řízeného algoritmy s manuálně nastavenými



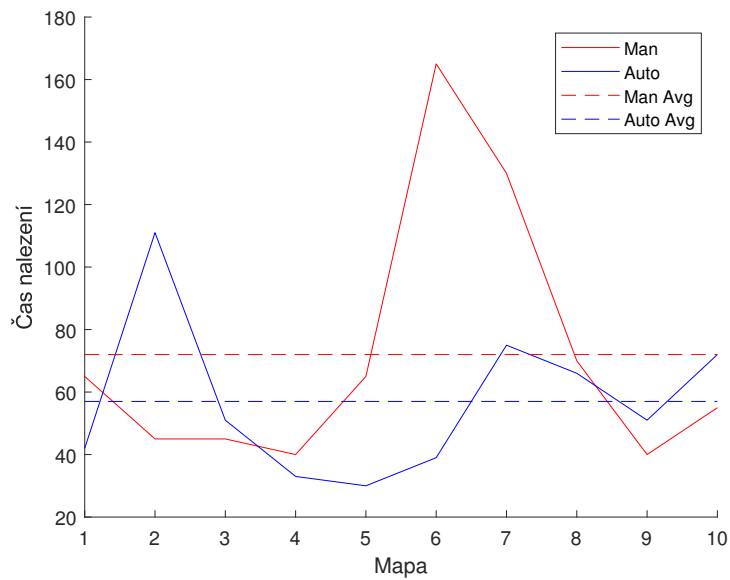
OBRÁZEK 5.12: Porovnání počtu dosažených waypointů rojem, řízeným algoritmem s manuálně a automaticky nastavenými parametry



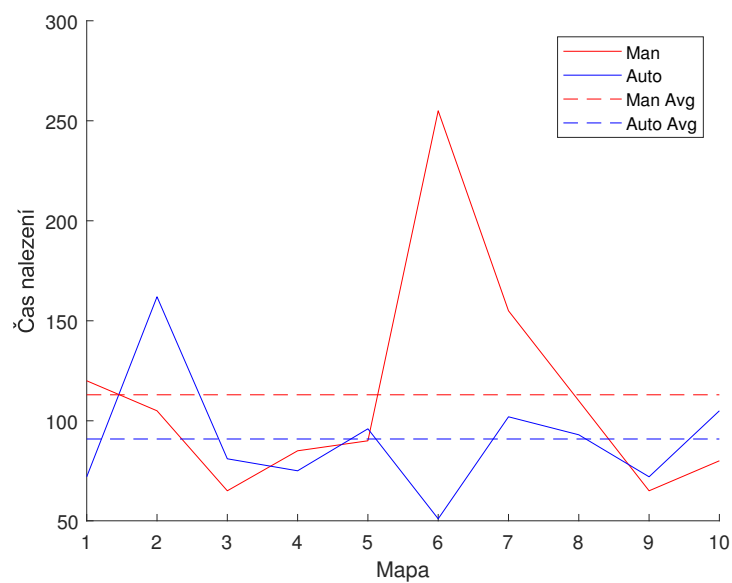
OBRÁZEK 5.13: Porovnání kompaktnosti roje, řízeným algoritmem s manuálně a automaticky nastavenými parametry



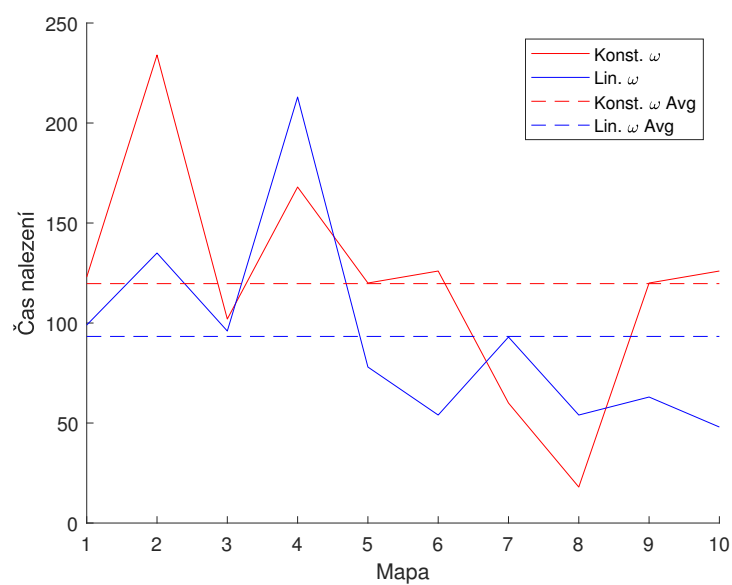
OBRÁZEK 5.14: Doba trvání první fáze algoritmu PSO, s manuálně a automaticky nastavenými parametry



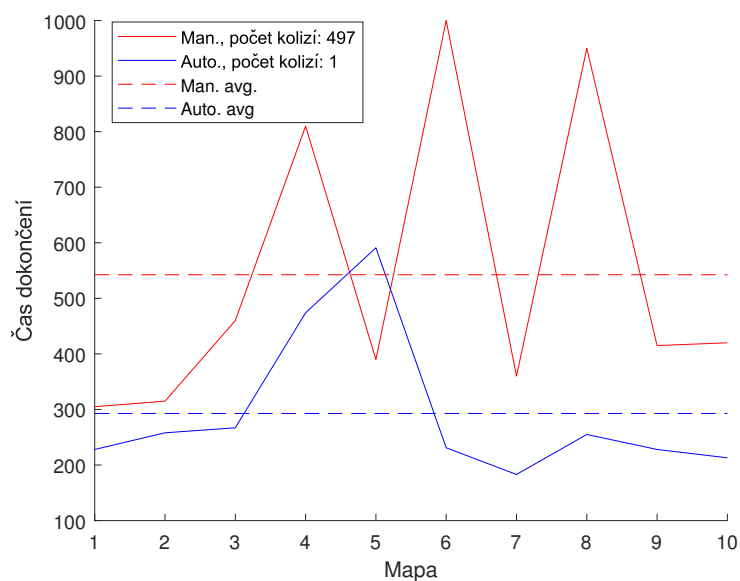
OBRÁZEK 5.15: Doba trvání druhé fáze algoritmu PSO (s konstantním parametrem ω), s manuálně a automaticky nastavenými parametry



OBRÁZEK 5.16: Doba trvání algoritmu PSO (s konstantním parametrem ω), s manuálně a automaticky nastavenými parametry



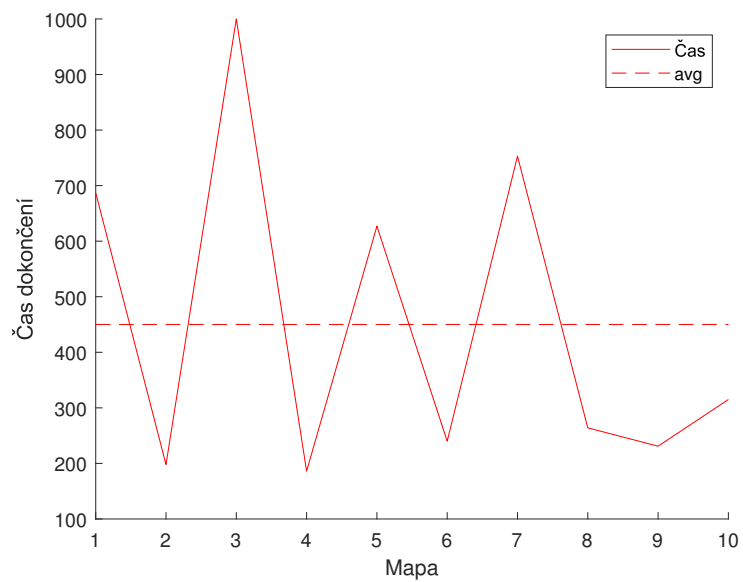
OBRÁZEK 5.17: Doba trvání druhé fáze algoritmu PSO, s konstantní a lineárně časově proměnnou hodnotou parametru ω



OBRÁZEK 5.18: Doba trvání celé simulační úlohy (s konstantním parametrem ω), s manuálně a automaticky nastavenými parametry

parametry. Zároveň došlo řádově k méně kolizím.

Na závěr byla otestována funkčnost řídicích algoritmů s automaticky nastavenými parametry a lineárně časově proměnnou hodnotou parametru ω , při lokalizaci koncentrace unikajícího plynu s hodnotou 1 (obrázek 5.19). Vidíme, že lokalizovat centrum úniku plynu se povedlo v 9 z 10 případů (limit trvání simulace byl nastaven na hodnotu 1000).



OBRÁZEK 5.19: Doba trvání celé simulační úlohy (s lineárně časově proměnnou hodnotou parametru ω), s automaticky nastavenými parametry, při lokaci koncentrace unikajícího plynu s hodnotou 1

6 Závěr

Tato práce se zabývala řízením roje dronů. Jejich úlohou byla lokalizace unikajícího plynu v prostředí s překážkami.

Nejdříve byl představen dron jako takový, byl popsán jeho pohyb prostoru, možná uplatnění a limitace. Postupně jsme se dopracovali k decentralizovaně řízené skupině dronů, tedy roji. Bylo popsáno řízení takové skupiny pomocí algoritmů řídících pohyb roje a algoritmu PSO. Simulace se uskutečnily v simulátoru SwarmLab, jehož funkce a manipulace s ním byly popsány. Proběhli testy efektivity řízení roje na dané simulační úloze. Pomocí genetického algoritmu byly naladěny parametry řídících algoritmů.

Výsledky simulací ukazují, že se povedlo vytvořit funkční strukturu řízení roje dronů, která úspěšně plní zadanou simulační úlohu. Z průběhů genetického algoritmu vidíme, že se zdařilo automaticky naladit parametry řídících algoritmů. Z následného testování a porovnávání s parametry manuálně nastavenými můžeme vyzorovat, že roj dosahuje lepších výsledků v případě řízení algoritmem s automaticky naladěnými parametry, a to ve všech případech.

Repozitář souborů simulátoru SwarmLab s přidanými funkcemi, popsány v této práci, je dostupný na platformě GitHub¹.

Obsah této práce poslouží zejména při aplikaci v reálném prostředí na roj dronů. Strukturu řízení lze použít, jak pro řešení zadané úlohy, tak pro řešení úloh s podobnou strukturou, za použití minimálních modifikací. Tato vlastnost plyne z podstaty chování a řízení roje.

Jedním z neřešených problémů, a tedy možným budoucím pokračováním práce, je pohyb v prostředí s konkrétními překážkami. Tento problém lze těžce řešit úpravou algoritmů řídících pohyb roje, možným řešením by byla modifikace algoritmu PSO. Dalším možným zdokonalením by mohla být propracovanější simulace unikajícího plynu nebo přesun roje do trojdimenzionálního prostoru.

¹<https://github.com/ondrej-benda/bp-swarmlab-swarm-control/tree/master>

Bibliografie

- [1] William Reeves T. „Particle Systems:A Technique for Modeling a Class of Fuzzy Objects“. In: 1 (1983).
- [2] Craig W. Reynolds. „Flocks, Herds and Schools-A Distributed Behavioral Model“. In: (1987), 25–34.
- [3] J. Kennedy a R. Eberhart. „Particle swarm optimization“. In: 4 (1995), 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.
- [4] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
- [5] Alcherio Martinoli a Kjerstin Easton. „Modeling swarm robotic systems“. In: *Experimental robotics VIII*. Springer, 2003, s. 297–306.
- [6] Henri Eisenbeiss et al. „A mini unmanned aerial vehicle (UAV): system overview and image acquisition“. In: *International Archives of Photogrammetry. Remote Sensing and Spatial Information Sciences* 36.5/W1 (2004), s. 1–7.
- [7] R. Olfati-Saber. *Flocking for multi-agent dynamic systems: algorithms and theory*. 2006. DOI: 10.1109/TAC.2005.864190.
- [8] Simon Garnier, Jacques Gautrais a Guy Theraulaz. „The biological principles of swarm intelligence“. In: *Computer* 1.4 (2007), s. 111–113. DOI: 10.1007/s11721-007-0004-y.
- [9] Michael G. Hinchey, Roy Sterritt a Chris Rouff. „Swarms and Swarm Intelligence“. In: *Computer* 40.4 (2007), s. 111–113. DOI: 10.1109/MC.2007.144.
- [10] Wisnu Jatmiko, Kosuke Sekiyama a Toshio Fukuda. „A pso-based mobile robot for odor source localization in dynamic advection-diffusion with obstacles environment: theory, simulation and measurement“. In: *IEEE Computational Intelligence Magazine* 2.2 (2007), s. 37–51.
- [11] Ed Obert. *Aerodynamic design of transport aircraft*. IOS press, 2009.
- [12] Teppo Luukkonen. „Modelling and control of quadcopter“. In: *Independent research project in applied mathematics, Espoo* 22 (2011), s. 22.
- [13] Shaer Blog. *Fixed Wing Drone For Surveillance*. 2012. URL: <https://shaear-hd.blogspot.com/2012/02/fixed-wing-drone-for-surveillance.html>.
- [14] Martin Saska, Jan Vakula a Libor Přeučil. „Swarms of micro aerial vehicles stabilized under a visual relative localization“. In: (2014), s. 3570–3575. DOI: 10.1109/ICRA.2014.6907374.
- [15] Nabil Nafia et al. „Robust Full Tracking Control Design of Disturbed Quadrotor UAVs with Unknown Dynamics“. In: *Aerospace* 5.4 (2018). ISSN: 2226-4310. DOI: 10.3390/aerospace5040115. URL: <https://www.mdpi.com/2226-4310/5/4/115>.
- [16] Gábor Vásárhelyi et al. „Optimized flocking of autonomous drones in confined environments“. In: *Science Robotics* 3.20 (2018), eaat3536. DOI: 10.1126/scirobotics.aat3536. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.aat3536>.

-
- [17] Saptarshi Sengupta, Sanchita Basak a Richard Alan Peters. „Particle Swarm Optimization: A Survey of Historical and Recent Developments with Hybridization Perspectives“. In: *Machine Learning and Knowledge Extraction* 1.1 (2019), s. 157–191.
- [18] Scott Camazine et al. *Self-Organization in Biological Systems*. Princeton University Press, 2020. ISBN: 9780691212920. DOI: doi:10.1515/9780691212920. URL: <https://doi.org/10.1515/9780691212920>.
- [19] Mario Coppola et al. „A Survey on Swarming With Micro Air Vehicles: Fundamental Challenges and Constraints“. In: *Frontiers in Robotics and AI* 7 (2020). ISSN: 2296-9144. DOI: 10.3389/frobt.2020.00018. URL: <https://www.frontiersin.org/article/10.3389/frobt.2020.00018>.
- [20] Enrica Soria, Fabrizio Schiano a Dario Floreano. „SwarmLab: A MATLAB drone swarm simulator“. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, s. 8005–8011.
- [21] Guinness World Records. „3,281 drones break dazzling record for most airborne simultaneously“. In: (2021). URL: <https://www.guinnessworldrecords.com/news/commercial/2021/5/3281-drones-break-dazzling-record-for-most-airborne-simultaneously-655062>.
- [22] Miquel Kegeleirs, Giorgio Grisetti a Mauro Birattari. „Swarm SLAM: Challenges and Perspectives“. In: *Frontiers in Robotics and AI* 8 (2021). ISSN: 2296-9144. DOI: 10.3389/frobt.2021.618268. URL: <https://www.frontiersin.org/article/10.3389/frobt.2021.618268>.
- [23] Wikipedia. *Quadcopter*. 2022. URL: <https://en.wikipedia.org/wiki/Quadcopter>.