FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Navigace letky dronů v prostředí s překážkami
Student:	Erich Beneš
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této práce je realizace multiagentního hledání cest (MAPF) ve 3D prostoru na dronech Crazyflie. Speciálně se práce bude zaměřovat na navigaci dronů v prostředí s překážkami. Předpokládá se využití existujících algoritmů pro MAPF a jejich implementací.

Úkoly pro uchazeče jsou následující:

1. Nastudovat literaturu k problému multi-agentního hledání cest, zejména řešící algoritmy a seznámit se s programovým vybavením dronů Crazyflie a jejich ovládáním

2. Zvolit a implementovat vhodný postup transformace cest nalezených algoritmem pro MAPF na navigační sekvenci pro dron

3. Výsledný systém pro navigaci dronů otestovat v relevantních scénářích s překážkami

[1]. Pavel Surynek. Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories. Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), pp. 1916-1922.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 17. února 2022 v Praze.

Bakalářská práce

NAVIGACE LETKY DRONŮ V PROSTŘEDÍ S PŘEKÁŽKAMI

Erich Beneš

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Prof. RNDr. Pavel Surynek, Ph.D.
16. února 2023

Obsah

Seznam obrázků

Seznam tabulek

Chtěl bych touto cestou poděkovat své přítelkyni a rodině za to, že mi jsou dlouhodobě oporou. Také bych chtěl poděkovat vedoucímu této práce Prof. RNDr. Pavlu Surynkovi, Ph.D., za cenné rady a podporu při vypracování této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 16. února 2023 .....

Abstrakt

Tato práce se zaměřuje na tvorbu systému pro převod existujících letových plánů na letovou sekvenci dronů za pomoci algoritmů multiagentního hledání cest. V úvodu práce je definováno multiagentní hledání cest a dva algoritmy pro řešení této problematiky, CBS a SMT-CBS^{R}. V práci je také představeno prostředí pro ovládání dronů Crazyflie 2.1, speciálně pak systém Loco Positioning. Výstupem práce je pak balíček *crazyflie_controller*, který umožňuje jednoduchou komunikaci s drony Crazyflie 2.1, převod plánů na letové sekvence, vyhodnocení jejich kvality a vizualizaci letů. Tento balíček je následně otestován v prostředí s překážkami.

Klíčová slova Multiagentní systémy, drony, MAPF, Crazyflie 2.1, Systém Loco Positioning

Abstract

The focus of this thesis is mainly on the creation of a system for the transformation of existing flight plans found using multi-agent pathfinding algorithms to flight sequences for drones. The definition of multi-agent pathfinding and algorithms CBS and SMT-CBS^{R} are introduced in the first chapter. The second chapter explains how Crazyflie 2.1 drone control environment works, with emphasis on Loco Positioning system. Then package *crazyflie_controller* is presented. It offers a simple way of communicating with drones Crazyflie 2.1, along with a way of transforming flight plans into flight sequences, evaluation and visualization of the results. The package is then tested in an environment containing obstacles.

Keywords Multi-agent systems, drones, MAPF, Craziflie 2.1, Loco Positioning System

Kapitola 1
Úvod

Dnes již ustálené slovo „drone“ v anglickém jazyce původně označovalo včelího trubce nebo monotónní stálý zvuk. Jako název pro bezpilotní letouny se podle Merriam-Websterova slovníku začalo používat už v době po druhé světové válce. Charakteristický zvuk vydávaný při letu dronů společně s jednoduchým posláním včelích trubců patrně zapříčinil zpopularizování tohoto označení [1].

Drony se v současné době těší čím dál větší oblibě v mnoha aplikacích. Primárně se drony nasazují v situacích, kdy je k dispozici mnoho volného prostoru. Příkladem může být průzkum terénu, monitorování staveništ, natáčení filmových záběrů či pro doručování menších objektů. Všechny tyto aplikace ale spojuje dostatek prostoru a tedy primárně venkovní užití.

V této práci se budeme věnovat navigaci dronů ve výrazně uzavřenějším prostředí, specificky prostředí s překážkami. Autonomní let dronů v uzavřeném prostoru vytváří řadu problémů, které je nutno řešit. Hlavním z nich je naplánování vhodných tras tak, abychom zamezili případné srážce dvou dronů. Tím se zabývá multiagentní hledání cest. Touto problematikou se budeme zabývat v první kapitole této práce. Uvedeme si zde její definici a také několik příkladů množných řešení.

Následně představíme prostředí pro ovládání dronů Crazyflie 2.1, které budeme využívat v rámci této práce. Společně s nimi si navíc představíme lokalizační systém a další pomocné nástroje, které budou použity.

V další části uvedeme systém, který jsme vytvořili pro komunikaci s drony Crazyflie 2.1 a principy převodu cest nalezených v prostoru s překážkami na letové sekvence, které budou vykonatelné těmito drony.

Na závěr pak provedeme několik experimentů, které otestují vlastnosti námi vytvořeného systému pro navigaci dronů a provádění definovaných letových sekvencí.

Kapitola 2
Kooperativní MAPF

Multiagentní hledání cest nebo také MAPF (z anglického „Multi-agent pathfinding“) je téma, které se velmi úzce váže k mnoha odvětvím současné lidské činnosti. Velké využití má například v herním průmyslu, ale je také nedílnou součástí témata automatizace. Další vyžití nalezne například u navigace robotů v automatizovaných skladech nebo v našem případě při tvorbě letového plánu pro letku dronů v uzavřeném prostředí s překážkami.

Tento typ MAPF se nazývá kooperativní a lze popsat jako problém, ve kterém se několik agentů současně snaží nalézt bezkolizní cestu ke svým cílovým destinacím, přičemž každá z nich má informace o cestách ostatních agentů [2]. Kontrastem ke kooperativnímu MAPF může být varianta nekooperativní, kdy agenti informace o cestách nesdílí. [3].

2.1 Jednoagentní hledání cest

Abychom si mohli lépe popsat MAPF, je vhodné nejprve představit jednoagentní hledání cest. V této problematice se snažíme o nalezení optimální (což v našem případě znamená nejkratší, ale v závislosti na definici problému se může tento význam měnit) cesty ze startovní pozice agenta do cíle. Je mnoho způsobů, jak definovat prostor, ve kterém se snažíme cestu najít nicméně v tomto případě se omezíme na prohledávání neorientovaného grafu. Tuto problematiku můžeme definovat následujícím způsobem:

▶ **Definice 2.1.** *Nechť $G = (V, E)$ je neorientovaný graf, kde jako $v \in V$ označujeme jednotlivé vrcholy grafu a jako $e \in E$, kde $v_1, v_2 \in V$, a $e = \{v_1, v_2\}$ označujeme hrany tohoto grafu. Dále definujeme počáteční vrchol $\alpha_0 \in V$ a koncový vrchol $\alpha_+ \in V$. Pak jako jednoagentní hledání cest označujeme problém nalezení nejkratší možné cesty z vrcholu α_0 do vrcholu α_+.*

Za řešení takto definovaného hledání považujeme nejkratší možnou souvislou sekvenci vrcholů $\pi = \{\alpha_0, \alpha_1, \ldots, \alpha_+\}$, která splňuje podmínku $(\forall \alpha_i, \alpha_{i+1} \in \pi) : \{\alpha_i, \alpha_{i+1}\} \in E$ a pro všechny ostatní cesty π_j pro které existuje souvislá posloupnost $\pi_j = \{\alpha_0, \ldots, \alpha_+\}$ platí, že $|\pi| \leq |\pi_j|$. Pokud pak zjistíme, že neexistuje žádná taková cesta π, pak daný problém nemá řešení.

Tyto algoritmy můžeme dále kategorizovat podle toho, zda máme dostupnou informaci o pozici cílového vrcholu α_+ v grafu nebo ne. V případě, že touto informací nedisponujeme, musíme použít algoritmy, které „náhodně“ prochází graf z počátečního vrcholu α_0, dokud nenarazí na koncový vrchol. Příkladem takového algoritmu je BFS (z anglického „Breadth-first search“), který postupně prohledává nejbližší dostupné a dosud nenavštívené vrcholy, dokud nenajde vrchol α_+ nebo neprojde všechny dostupné vrcholy. Tento typ prohledávání nazýváme neinformované.

Pokud ovšem máme informaci o pozici koncového vrcholu, můžeme využít prohledávání informované.

2.1.1 A*

Jako příklad informovaného algoritmu pro řešení jednoagentního plánování cest si v této práci uvedeme algoritmus A*. Ten se řadí mezi algoritmy heuristické, což znamená, že využívá pro rychlejší nalezení optimální cesty heuristický odhad.

Pro představu uvádíme pseudokód:

Algoritmus 1 A*[4]

```
1:  prev ← fill(0)
2:  dist ← fill(0)
3:  open ← (0, α0)
4:  while open not empty do
5:      x ← remove_min(open)
6:      if x = α+ then
7:          return reverse(prev, x)
8:      for ∀y ∈ neighbors(x) \ closed do
9:          d ← dist[x] + 1
10:         if y ∉ open ∨ dist[y] > d then
11:             dist[y] ← d′
12:             prev[y] ← x
13:             add_to_open(d + h(y), y)
14:     closed ← closed ∪ {x}
```

Na počátku inicializujeme prostředky, které budeme v tomto algoritmu používat. Jako první vytvoříme pole *dist* a *prev*, obě o velikosti $|V|$, která vyplníme nulovými hodnotami. Do $dist[i]$ budeme v budoucnu ukládat hodnoty reprezentující vzdálenost vrcholu i od počátku a do $prev[i]$ pak vrchol ze kterého jsme do i vstoupili. také inicializujeme prázdnou množinu *closed*, do které budeme vkládat již navštívené vrcholy grafu. Poté inicializujeme prioritní frontu *open*. Do této fronty budeme v průběhu algoritmu vkládat páry (k, v). V tomto páru k označuje délku cesty (která bude zmíněna později) a daný vrchol v. Algoritmus tedy iniciujeme tím, že do této fronty vložíme $(0, \alpha_0)$. Dokud se pak tato fronta nevyprázdní nebo nebude nalezeno řešení, bude algoritmus postupně odebírat páry z této fronty seřazené od nejmenší vzdálenosti od počátku po tu nejvyšší.

Poté co byl odebrán pár $x = (k, v)$ z fronty, kontroluje se, zda není vrchol v koncovým vrcholem α_+. Pokud to je koncový vrchol, pak algoritmus našel řešení $\pi = reverse((prev[i] = (i-1), prev[i-1] = (i-2), \ldots, prev[i-n] = 0))$.

V případě, že se o řešení nejedná algoritmus nalezne všechny sousední vrcholy $y \in V$ vrcholu v, které nejsou obsaženy v množině *closed*. Pro každý vrchol y, který není v *open* pak upraví hodnoty $dist[y] = dist[x] + 1$ a $prev[y] = x$. Poté přidá do fronty *open* dvojici $(d + h(y), y)$. Nakonec pak uzavře vrchol x a pokračuje dalším cyklem.

Ústředním prvkem algoritmu je tedy prioritní fronta *open*, do které vkládáme v průběhu prohledávání vrcholy grafu, kterým přiřazujeme nějakou váhu $d + h(y)$, kterou nazýváme heuristickou funkcí. Za její pomoci pak můžeme ovlivnit, v jakém pořadí budeme procházet vrcholy, které byly otevřeny algoritmem. To označujeme jako heuristický odhad. V případě, že by byla tato heuristická funkce ideální, odhadl by algoritmus pro každý vrchol váhu, která je rovna délce nejkratší možné cesty procházející daným vrcholem grafu. Takovou funkci značíme $\widehat{f}$ a pro každý vrchol grafu $n \in V$ odhaduje vzdálenost:

$$\widehat{f}(n) = \widehat{g}(n) + \widehat{h}(n) \tag{2.1}$$

kde $\widehat{g}(n)$ je délka optimální cesty z α_0 do vrcholu n a $\widehat{h}(n)$ je délka optimální cesty z vrcholu n do vrcholu α_+. Ideální heuristika je ale typicky nedosažitelná a tedy se musíme spokojit

s heuristikou přípustnou, která otevírá více vrcholů, ale zaručuje že výsledná nalezená cesta je optimální v případě že pro:

$$f(n) = g(n) + h(n) \tag{2.2}$$

platí, že $g(n)$ je rovno nejkratší nalezené vzdálenosti z α_0 do n a $h(n)$ splňuje podmínku přípustnosti:

$$\forall v \in V : h(v) \leq \widehat{h}(v) \tag{2.3}$$

[5, 6]

2.2 Multiagentní hledání cest v diskrétním prostoru

Nyní, po zadefinování jednoagentního hledání cest můžeme tuto definici rozšířit na hledání cest pro více agentů současně. V problematice MAPF v diskrétním prostoru se snažíme současně o nalezení cesty pro každého z množiny agentů, kteří se nachází v různých vrcholech neorientovaného grafu do jejich cílových vrcholů. Formálně MAPF definujeme jako čtveřici:

▶ **Definice 2.2.** $\Sigma = (G, A, \alpha_0, \alpha_+)$, *kde*

- $G = (V, E)$ *je neorientovaný graf, kde jako* $v \in V$ *označujeme jednotlivé vrcholy grafu a jako* $e \in E$, *kde* $v_1, v_2 \in V$, *a* $e = \{v_1, v_2\}$ *označujeme hrany tohoto grafu*
- $A = \{a_1, a_2, \ldots, a_k\}$ *je množina agentů, kde* $|A| < |V|$ *a zároveň platí, že v každém vrcholu* V *se nachází maximálně jeden agent*
- α_0 *je množina počátečních vrcholů taková, že pro každého agenta* $a_i \in A$ *existuje právě jeden unikátní vrchol* $\alpha_0(a_i) \in V$
- α_+ *je množina koncových vrcholů taková, že pro každého agenta* $a_i \in A$ *existuje právě jeden unikátní vrchol* $\alpha_+(a_i) \in V$.

Jako řešení pak označujeme množinu jednoagentních plánů $\pi = \{\pi(a_1), \ldots, \pi(a_k)\}$, kde $\pi(a_i)$ je sekvence akcí pro jednoho agenta $a_i \in A$. Nově také potřebujeme definovat i čas rozdělený na množinu diskrétních časových kroků $T = \{t_1, \ldots, t_n\}$ kde $t_n \in \mathbb{N}$. Každý z agentů může v jednom časovém kroku provést akci „čekání“, kdy setrvá ve vrcholu $v \in V$ grafu G nebo akci „pohyb“, kdy se agent přesune z vrcholu v do sousedního vrcholu v_n. Pro tyto vrcholy pak musí platit, že $\{v, v_n\} \in E$. [7]

2.2.1 Konflikty

Konflikty jsou další zásadní součásti MAPF problematiky. Jelikož jsme do prostoru hledání uvedli další agenty, musíme si lépe definovat, jaké situace jsou přípustné a jaké už tolerovat nemůžeme. Tyto zakázané stavy nazýváme konflikty. Prozatím se omezíme na takzvaný vrcholový konflikt. Ten nastává v situaci, kdy se v plánu $\pi(a_i)$ a $\pi(a_j)$ a časovém kroku $t \in T$ nachází oba agenti ve stejném vrcholu $v \in V$. Takovou situaci značíme čtveřicí (a_i, a_j, v, t). [8]

2.2.2 Cenové funkce

Abychom mohli hodnotit jednotlivá nalezená řešení MAPF problémů, je nutné definovat si kritéria takového hodnocení. Tato kritéria se obvykle nazývají cenové funkce. Mezi často používané cenové funkce se řadí „Součet cen“, zkráceně SoC (z anglického „Sum-of-costs“), která hodnotí nalezené řešení podle celkového počtu kroků vykonaných všemi agenty při provádění letového plánu. Další možnou volbou pro cenovou funkci je délka nejdelších z těchto cest – „Makespan“

(budeme označovat „M“). Formálně můžeme tyto funkce definovat takto:

$$SoC(\pi) = \sum_{1 \leq i \leq k} |\pi_i| \tag{2.4}$$

$$M(\pi) = \max_{1 \leq i \leq k} |\pi_i| \tag{2.5}$$

Cenové funkce jsou zásadní pro nalezení optimálního řešení, jelikož ve většině situací existuje více možných řešení daného MAPF problému. Pokud neignorujeme konflikty, je nalezení takového řešení dokonce NP-úplný problém [9]. Z tohoto důvodu také existují algoritmy, které se spokojí s nalezením libovolného řešení, které nemusí být optimální. Těmto algoritmům se říká suboptimální.

2.2.3 CBS

Algoritmus CBS (zkratka anglického názvu „Conflict-Based Search“) je optimální algoritmus používaný pro řešení MAPF problémů. Tento algoritmus je rozdělen do dvou úrovní – High-level a Low-level.

2.2.3.1 Low-level algoritmus

Low-level algoritmus má za úkol nacházení jednoagentních cest. Pro tento problém lze použít libovolný algoritmus, který zaručuje nalezení optimální cesty. Například lehce použít upravenou variantu algoritmu A*. Ta neprovádí prohledávání pouze v grafu G, ale bere v potaz i množinu konfliktů, které jsou jí předány. Je vhodné, aby tento algoritmus byl co nejefektivnější, jelikož je opakovaně volán z High-level funkce.

2.2.3.2 High-level algoritmus

Tento algoritmus implementuje strom CT, který v každém svém vrcholu N udržuje následující informace:

- $N.constraints$ je množina strukturovaných trojic ve formátu ($a \in A$, $v \in V$, $t \in T$). Jedná se o množinu omezení, která v kořenovém vrcholu začíná jako prázdná a s detekcí konfliktů se postupně rozšiřuje o omezení, která těmto konfliktům předchází. Každý potomek dědí všechna omezení definovaná rodičem a rozšiřuje je o omezení další.
- $N.solutions$ je množina k cest, kde $k = |A|$. Obsahuje tedy pro každého z agentů $a_i \in A$ cestu nalezenou pomocí Low-level vyhledávání.
- $N.cost$ je hodnota výstupu zvolené cenové funkce současného řešení, která je vždy nižší nebo rovna hodnotě ceny v rodiči.

V pseudokódu 2 můžeme vidět, že algoritmus nejprve inicializuje kořenový vrchol K stromu CT, který na počátku obsahuje prázdnou množinu omezení. Následně pro každého agenta nalezne cestu za pomoci Low-level algoritmu. Získané cesty uloží do proměnné $K.solution$, určí cenu tohoto řešení a uloží ji do $K.cost$. Tento vrchol je pak vložen do fronty $Open$.

Algoritmus vždy odebere vrchol P s nejnižší hodnotou cenové funkce $P.cost$ z fronty $Open$ a validuje jej. Cílem validace je ověřit, že řešení neobsahuje žádný konflikt, tedy například že neexistuje konflikt (a_i, a_j, v, t), kde $a_i, a_j \in A$ a zároveň $a_i \neq a_j$. Pokud v průběhu validace není nalezen žádný konflikt, označíme řešení v tomto vrcholu CT za konečné a algoritmus vrátí set cest $P.solution$. V případě že algoritmus na konflikt narazí, řeší vždy pouze první objevený. V této chvíli vytvoří algoritmus pro každého agenta a_i, který je součástí daného konfliktu nového potomka CT. Každý z těchto potomků pak obsahuje kromě omezení rodiče i nové omezení (a_i, v, t). Potom iniciuje hledání nové cesty splňující tato omezení Low-level algoritmem.

Algoritmus 2 High Level CBS [10]

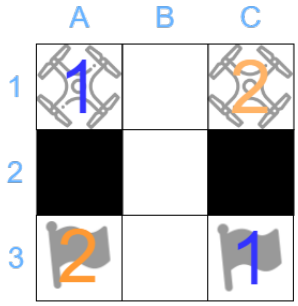
1: $K.constraints = \emptyset$
2: $K.solution = LowLevel()$
3: $K.cost = CostFunction(K.solution)$
4: Vlož K do $Open$
5: **while** $Open$ not empty **do**
6: $P \leftarrow$ lowest cost member from $Open$
7: Validate(P)
8: **if** P has no conflict **then**
9: **return** $P.solution$
10: $C \leftarrow$ first_conflict(a_i, a_j, v, t) z P
11: **for** agent a_i v C **do**
12: $A \leftarrow$ new_child(P)
13: $A.constraints \leftarrow P.constraints + (a_i, v, t)$
14: $A.solution \leftarrow P.solution$
15: update $A.solution$ with $LowLevel(a_i)$
16: $A.cost = CostFunction(A.solution)$
17: **if** $A.cost < \infty$ **then**
18: Insert A into $Open$

Nakonec se algoritmus pokusí určit cenu nového řešení. V případě že takové řešení existuje, přidá tento vrchol do fronty. Pokud se algoritmus dostane na konec fronty, pak řešení neexistuje.

2.2.3.3 Příklad

Jako příklad fungování algoritmu CBS vezměme situaci na obrázku 2.1. V této situaci máme dva agenty označené jako 1 a 2. Vrcholy prohledávaného grafu G jsou označeny světlými čtverci. Tyto vrcholy označujeme kombinací příslušného písmena a čísla, tedy například agent 1 se nachází ve vrcholu $A1$. Hrany jsou pak definované mezi každými dvěma vertikálně nebo horizontálně sousedícími vrcholy. Počáteční vrchol je označen ikonou dronu s číslem agenta a cíl je označen vlaječkou. V tomto příkladu budeme používat cenovou funkci M (2.5).

Obrázek 2.1 Ukázkový MAPF problém



Algoritmus v první fázi nalezne optimální cestu pro každého z agentů a určí jejich cenu. V tomto případě dostaneme následující cesty:

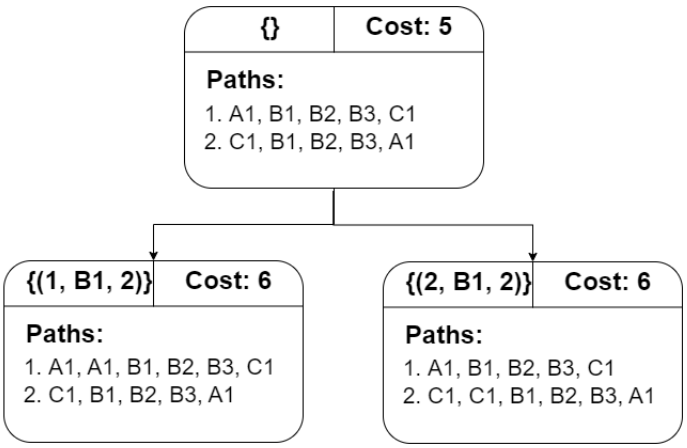
- $\pi_1 = (A1, B1, B2, B3, C3)$
- $\pi_2 = (C1, B1, B2, B3, A3)$

Následuje validace cest. Algoritmus detekuje konflikt v prvním kroku, kdy se oba agenti současně pokusí přesunout na pozici $B1$. Rozšíří tedy CT o dva potomky 2.2. Každý z těchto potomků definuje nové omezení zamezující jednomu z agentů v tomto kroku navštívit vrchol $B1$. Pro každého z těchto potomků pak nalezne nové řešení a určí jeho cenu. Poté pokračuje v prohledávání CT v potomkovi s nejnižší cenou. Jelikož mají obě řešení stejnou cenu, závisí na konkrétní implementaci, který z vrcholů bude zvolen. Nechť je to v našem případě vrchol s omezením $(1, B1, 2)$:

- $\pi_1 = (A1, B1, B1, B2, B3, C3)$
- $\pi_2 = (C1, B1, B2, B3, A3)$

V tomto potomkovi se již nenachází konflikt v plánech jednotlivých agentů, proto algoritmus označí cesty v tomto vrcholu jako validní řešení a ukončí se.

Obrázek 2.2 CT pro příklad 2.2.3.3



2.3 Multiagentní plánování ve spojitém čase a prostoru

V předchozí kapitole jsme se při definici MAPF omezovali pouze na diskrétní čas a diskrétní prostor a předpokládali jsme, že agenti se instantně pohybují z vrcholu grafu do vrcholů sousedních v jednom časovém kroku. V realitě ovšem musíme brát v potaz jak rychlostní limitace jednotlivých agentů, tak reálnou vzdálenost vrcholů, kterou musí agent urazit. Dále musíme vzít v úvahu realistickou velikost agentů. To přináší řadu dalších komplikací, jelikož se již nemůžeme omezovat na vrcholové konflikty, ale musíme řešit konflikty prostorové.

Definice multiagentního plánování v kontinuálním čase a prostoru MAPF$^{\mathrm{R}}$ se opírá o předchozí definici MAPF a rozšiřuje ji o další složky, kterými jsou pozice vrcholů v prostoru, rychlost agentů a prostor který agenti zabírají. Tyto hodnoty jsou definovány parametrem ρ. MAPF$^{\mathrm{R}}$ tedy definujeme jako pětici:

▶ **Definice 2.3.** $\Sigma^R = (G, A, \alpha_0, \alpha_+, \rho)$, *kde*

- $G = (V, E)$ *je neorientovaný graf, kde jako* $v \in V$ *označujeme jednotlivé vrcholy grafu a jako* $e \in E$, *kde* $v_1, v_2 \in V$, *a* $e = \{v_1, v_2\}$ *označujeme hrany tohoto grafu*
- $A = \{a_1, a_2, \ldots, a_k\}$ *je množina agentů, kde* $|A| < |V|$ *a zároveň platí, že v každém vrcholu* G *se nachází maximálně jeden agent*

- α_0 *je množina počátečních vrcholů taková, že pro každého agenta* $a_i \in A$ *existuje právě jeden unikátní vrchol* $\alpha_0(a_i) \in V$
- α_+ *je množina koncových vrcholů taková, že pro každého agenta* $a_i \in A$ *existuje právě jeden unikátní vrchol* $\alpha_+(a_i) \in V$
- $\rho.x(v), \rho.y(v), \rho.z(v)$ *pro* $v \in V$*, reprezentují pozici jednotlivých vrcholů v prostoru*
- $\rho.v(a)$ *pro každého agenta* $a \in A$*, reprezentuje konstantní rychlost, kterou se agent pohybuje*
- $\rho.d(a)$ *pro každého agenta* $a \in A$*, reprezentuje poloměr koule, ve které se agent nachází*

Díky zasazení jednotlivých vrcholů do reálného prostoru můžeme nyní definovat vzdálenost mezi vrcholy a tedy i délku případných hran mezi nimi (pokud jsou v grafu G definovány). Pro libovolné $v_1, v_2 \in V$ tedy můžeme určit vzdálenost následovně:

$$||v_1, v_2|| = \sqrt{(\rho.x(v_1) - \rho.x(v_2))^2 + (\rho.y(v_1) - \rho.y(v_2))^2 + (\rho.z(v_1) - \rho.z(v_2))^2} \tag{2.6}$$

Další významnou změnou je definice rychlosti agentů $\rho.v$, což nám redefinuje jednu z akcí kterou jsme představili v MAPF. Akce „pohyb“ nyní není prováděna instantně, ale agent se přesunuje po hraně v časovém intervalu $[t_i, t_{i+1})$. Z tohoto důvodu již nelze používat diskrétní časové kroky, ale musíme pracovat s časem kontinuálním, tedy $t \in \mathbb{R}^+$.

Řešením MAPF$^{\mathrm{R}}$ stále označujeme množinu jednoagentních plánů $\pi = \{\pi(a_1), \ldots, \pi(a_k)\}$. Nicméně nyní budeme definovat plán pro jednoho agenta $a_i \in A$ jako sekvenci trojic definující dva vrcholy mezi kterými se agent přesouval a časový interval, ve kterém prováděl agent akci „přesun“ – $(\alpha_j(a_i), \alpha_k(a_i), [t_0(a_i), t_1(a_i)))$ nebo akci „čekání“ – $(\alpha_j(a_i), \alpha_j(a_i), [t_0(a_i), t_1(a_i)))$. Akci „pohyb“ lze navíc provést jen pokud v grafu G existuje hrana mezi vrcholy $\alpha_j(a_i)$ a $\alpha_k(a_i)$ v daném jednoagentním plánu, tedy $\{\alpha_j(a_i), \alpha_k(a_i)\} \in E$. Plán pro agenta a_i bude tedy vypadat takto:

$$\begin{aligned}\pi(a_i) = [&(\alpha_0(a_i), \alpha_1(a_i), [t_0(a_i), t_1(a_i)); (\alpha_1(a_i), \alpha_2(a_i), [t_1(a_i), t_2(a_i)); \\ &\ldots; (\alpha_{m(a_i)-1}(a_i), \alpha_{m(a_i)}(a_i), [t_{m(a_i)-1}(a_i), t_{m(a_i)}(a_i))]\end{aligned} \tag{2.7}$$

kde jako $m(a_i)$ označujeme délku plánu $\pi(a_i)$. [11]

2.3.1 Konflikty

U MAPF$^{\mathrm{R}}$ je definice konfliktu složitější, než tomu bylo u MAPF, jelikož agenti nabírají reálné rozměry a pohybují se v reálném prostoru a čase. Z tohoto důvodu se již nemůžeme omezovat na jednoduché „vrcholové“ konflikty, ale musíme řešit konflikty prostorové. K takovému konfliktu dochází, pokud se agenti a a b přiblíží na vzdálenost $d < \rho.d(a) + \rho.d(\mathrm{b})$. Takto definovaný konflikt je nicméně příliš výpočetně náročný, proto se stejně jako v [11] omezíme na volnější definici:

▶ **Definice 2.4.** *Kolize mezi dvěma plány* $\pi(a) = [\alpha_i(a), \alpha_{i+1}(a), [t_i(a), t_{i+1}(a))]_{i=0}^{m(a)-1}$ *a* $\pi(b) = [\alpha_i(b), \alpha_{i+1}(b), [t_i(b), t_{i+1}(b))]_{i=0}^{m(b)-1}$ *pro* $a \neq b$ *nastává, pokud:*

- $\exists i \in \{0, 1, \ldots, m(a) - 1\}$ *a* $\exists j \in \{0, 1, \ldots, m(j) - 1\}$ *takové že:*
  - $dist([\rho.x(\alpha_i(a)), \rho.y(\alpha_i(a)), \rho.z(\alpha_i(a)); \rho.x(\alpha_{i+1}(a)), \rho.y(\alpha_{i+1}(a)), \rho.z(\alpha_{i+1}(a))];$ $[\rho.x(\alpha_j(b)), \rho.y(\alpha_j(b)), \rho.z(\alpha_j(b)); \rho.x(\alpha_{j+1}(b)), \rho.y(\alpha_{j+1}(b)), \rho.z(\alpha_{j+1}(b))])$ $< \rho.d(a) + \rho.d(b)$
  - $[t_i(a), t_{i+1}(a)) \cap [t_j(b), t_{j+1}(b)) \neq \emptyset$

Tedy konflikt nastane, pokud se dva agenti a a b současně nacházejí na dvou vrcholech nebo hranách, které jsou od sebe vzdálené méně než $\rho.d(a) + \rho.d(b)$. Jako vzdálenost $dist(A = [a_1, b_1, c_1; a_2, b_2, c_2]; B = [a'_1, b'_1, c'_1; a'_2, b'_2, c'_2])$ pak označujeme minimální vzdálenost mezi body nacházejícími se na úsečkách A a B. Pokud nastane situace, kdy je délka jedné z těchto úseček nulová, definujeme tuto vzdálenost jako vzdálenost úsečky a bodu, případně dvou bodů, pokud jsou obě délky nulové. Tato definice konfliktu omezuje i případy kdy by ke konfliktu reálně nedošlo, nicméně je výrazně jednodušší na výpočet.

2.3.2 Cenové funkce

Cenové funkce u MAPF$^{\mathrm{R}}$ nejsou příliš rozdílné od cenových funkcí klasického MAPF. Stále nás zajímají délky nalezených cest, nicméně nyní již nemůžeme ceny redukovat na sumy pohybů, jelikož musíme brát v úvahu různé vzdálenosti mezi vrcholy a různé rychlosti jednotlivých agentů. Cenové funkce pro kontinuální MAPF nazveme SoC^R a μ a definujeme je následovně:

$$SoC^R(\pi) = \sum_{1 \leq i \leq k} |T(\pi(a_i))| \tag{2.8}$$

$$\mu(\pi) = \max_{1 \leq i \leq k} |T(\pi(a_i))| \tag{2.9}$$

Kde $k = |A|$ a jako $|T(\pi(a_i))|$ pak v rovnicích označujeme délku časového intervalu, který agent a_i potřeboval k provedení celé navigační sekvence.

2.3.3 SMT-CBS$^{\mathrm{R}}$

CBS$^{\mathrm{R}}$[12] je rozšířením dříve představeného algoritmu CBS pro MAPF$^{\mathrm{R}}$. High-level algoritmus funguje stejně jako klasický CBS. Nicméně v této části práce si představíme jeho upravenou variantu SMT-CBS$^{\mathrm{R}}$, která na rozdíl od CBS$^{\mathrm{R}}$ využívá princip „Statisfiability modulo theory“ (zkráceně SMT). Tato varianta je představena v práci [13] a je schopna nalézt pro cenovou funkci μ řešení ve výrazně lepším čase než CBS$^{\mathrm{R}}$.

Hlavním principem tohoto algoritmu je převod problému nalezení splnitelného plánu π na teorii T. Tato teorie je tvořena logickými formulemi v konjunktivní formě, u kterých lze rozhodnout o splnitelnosti (ta se také označuje jako SAT, což zkratka anglického „Satisfiability“).

Jako atomické prvky formulí využívá SMT-CBS$^{\mathrm{R}}$ proměnou $\chi_v^t(a_i)$ popisující „čekání“ ve vrcholu v v časovém kroku t a $\varepsilon_{u,v}^t(a_i)$ představující „pohyb“ agenta a_i po hraně $\{u, v\} \in E$ v čase $t \in \mathbb{R}+$. Tyto proměnné nejsou definovány pro každý stav t, ale pouze pro zásadní situace, jako například v případě konfliktu, v začátku provádění akce „pohyb“ nebo při ukončení akce „čekání“. Generování těchto proměnných popisuje algoritmus 3.

Ten generuje množinu rozhodovacích proměnných VAR tak, že pro každého z agentů postupně prochází vrcholy, kterých je agent schopen dosáhnout v čase $t < \mu_{max}$ a vrcholy s nimi sousedící. Toto prohledávání je iniciováno ze startovního vrcholu agenta $(\alpha_0(a))$. V každém dosaženém vrcholu $v \in V$ jsou přidány do fronty vrcholů určených k prohledání všechny sousední vrcholy $u \in V$ takové, že platí $\{u, v\} \in E$. Zároveň uloží informaci o tomto potenciálním přechodu do množiny VAR. Poté, pokud by tento přechod způsobil konflikt, uloží do této množiny zároveň informaci o čekání. Takto vygenerovaná množina proměnných je pak využívána v Low-level cyklu algoritmu.

2.3.3.1 High-level algoritmus

Hlavním rozdílem oproti CBS$^{\mathrm{R}}$ algoritmu je relativně jednoduchý High-level algoritmus, který nemá sklony k rozsáhlému větvení (tento problém lze částečně vidět i na klasickém CBS). Algoritmus neimplementuje CT jako tomu je v CBS$^{\mathrm{R}}$, ale pouze inicializuje množinu konfliktů

Algoritmus 3 Generace rozhodovacích proměnných pro SMT-CBS$^{\mathrm{R}}$

```
1:  function GENERATE-DECISIONS(Σ^R = (G, A, α_0, α_+, ρ), conflicts, μ_max)
2:      VAR ← ∅
3:      for each a ∈ A do
4:          OPEN ← ∅
5:          INSERT(α_0(a), 0) into OPEN
6:          VAR ← VAR ∪ {χ^{t_0}_{α_0(a)}(a)}
7:          while OPEN ≠ ∅ do
8:              (u, t) ← MIN_t(OPEN)
9:              REMOVE-MIN_t(OPEN)
10:             if t ≤ μ_max then
11:                 for each v such that {u, v} ∈ E do
12:                     Δt ← DIST(u, v)/ρ.v(a)
13:                     INSERT(v, t + Δt) into OPEN
14:                     VAR ← VAR ∪ {ε^t_{u,v}(a), χ^{t+Δt}_v(a)}
15:                 for each v such that {u, v} ∈ E do
16:                     for each (a, {u, v}, [t_0, t_+)) ∈ conflicts do
17:                         if t_+ > t then
18:                             INSERT(u, t_+) into OPEN
19:                             VAR ← VAR ∪{χ^{t_+}_u(a)}
20:     return VAR
```

conflicts a cenovou funkci μ. Ta je definována stejně jako v 2.9, ale je odvozována z plánu π, který je tvořen cestami ignorujícími případné kolize. Takto inicializovaná hodnota je poté postupně inkrementována po každém volání Low-level algoritmu, dokud není nalezeno validní řešení.

Algoritmus 4 High-level Algoritmus pro SMT-CBS$^{\mathrm{R}}$

```
1: function SMT-CBS^R(Σ^R = (G, A, α_0, α_+, ρ))
2:     conflicts ← ∅
3:     π ← {π^+(a_i)}
4:     μ ← max_{i=1}^{k} μ(π(a_i))
5:     while TRUE do
6:         (π, conflicts, μ_next) ← SMT-CBS-FIXED^R(Σ^R, conflicts, μ)
7:         if π ≠ UNSAT then
8:             return π
9:         μ ← μ_next
```

2.3.3.2 Low-level algoritmus

V SMT-CBS$^{\mathrm{R}}$ se prohledávání provádí primárně v Low-level algoritmu. Ten z High-level funkce dostává pouze informace o řešeném problému Σ^R, množinu známých konfliktů *conflicts* a maximální cenovou funkci μ do které má být prohledáván graf.

V prvním kroku je zavolán algoritmus GENERATE-DECISIONS, který vygeneruje množinu proměnných VAR. Poté za pomoci funkce ENCODE-BASIC formuluje teorii $\mathcal{F}(\mu)$. Podrobnější popis postupu formulace této teorie lze najít v práci [14].

V dalším kroku CONSULT-SAT-SOLVER ověří splnitelnost této teorie $\mathcal{F}(\mu)$ v čase μ. Pokud v tomto intervalu teorie splnitelná není, je algoritmus ukončen a High-level algoritmu je předána

množina konfliktů a μ_{next}, které je navýšeno tak, aby algoritmus GENERATE-DECISIONS dosáhl při generaci dalšího vrcholu.

V případě že teorie splnitelná je, je převedena zpět na plán π a funkce VALIDATE-PLAN ověří, zda se v tomto plánu vyskytují kolize. Pokud tomu tak není, pak funkce vrací High-level algoritmu validní plán, který je splnitelný v čase definovaném μ.

V opačném případě proběhne zpracování kolizí. Pro každý pár v kolizi je vytvořena nová disjunktní podmínka, která je přidána do teorie $\mathcal{F}(\rho)$ a do množiny konfliktů jsou přidány dva nové konflikty. Poté se s těmito novými konflikty provede generace proměnných za použití algoritmu GENERATE-DECISIONS, upraví se teorie $\mathcal{F}(\mu)$ za pomoci funkce AUGUMENT-BASIC a cyklus se vrací k vyhodnocení SAT řešičem.

Algoritmus 5 Low-level Algoritmus pro SMT-CBS$^{\mathrm{R}}$

```
1:  function SMT-CBS^R-FIXED(Σ^R, conflicts, μ)
2:      VAR ← GENERATE-DECISIONS(Σ^R, conflicts, μ)
3:      F(μ) ← ENCODE-BASIC(VAR, Σ^R, μ)
4:      while TRUE do
5:          assignment ← CONSULT-SAT-SOLVER(F(μ))
6:          if assignment ≠ UNSAT then
7:              π ← EXTRACT-SOLUTION(assingment)
8:              collisions ← VALIDATE-PLANS(π)
9:              if collisions = ∅ then
10:                 return (π, UNDEF, conflicts)
11:             for each(a_i, {u, v}, [t_0, t_+)) × (a_j, {u', v'}, [t'_0, t'_+)) ∈ collisions do
12:                 Y ← (u = v) ? χ^{t_0}_u(a_i) : ε^{t_0}_{u,v}(a_i)
13:                 Z ← (u' = v') ? χ^{t'_0}_{u'}(a_j) : ε^{t'_0}_{u',v'}(a_j)
14:                 F(μ) ← F(μ) ∪ {¬Y ∨ ¬Z}
15:                 [τ_0, τ_+) ← [t_0, t_+) ∪ [t'_0, t'_+)
16:                 conflicts ← conflicts ∪ {(a_i, {u, v}, [τ_0, τ_+)), (a_j, {u', v'}, [τ_0, τ_+))}
17:             VAR ← GENERATE-DECISIONS(Σ^R, conflicts, μ)
18:             F(μ) ← AUGMENT-BASIC(VAR, F(μ), Σ^R, μ)
19:         else
20:             μ_next ← min{t | χ^t_u(a_i) ∈ VAR ∨ t > μ}
21:             return (UNSAT, conflicts, μ_next)
```

2.3.3.3 Příklad

Pro ukázku fungování SMT-CBS$^{\mathrm{R}}$ použijeme stejnou situaci, jako u příkladu v CBS. Ta lze vidět na obrázku 2.1. Pro jednoduchost řekněme, že jednotlivé čtverce označují vrcholy grafu G, které se nacházejí ve středu daného čtverce a mezi sousedními světlými poli existují hrany o délce 1. Například tedy $A1, B1 \in E$. Každý z agentů a_i je reprezentován dronem označeným číslem k, kde $i = k$. Dále má každý z agentů velikost $\rho.d(a_i) = 1$ a konstantní rychlost $\rho.v(a_i)$.

Nejprve je spuštěn High-level algoritmus, který stanoví počáteční hodnotu cenové funkce μ vytvořením plánu skládajícího se z nejkratších možných cest, kdy nebereme ohled na konflikty. Tento plán můžeme vyjádřit jako sekvenci vrcholů a časů $[v \in V(t)]$:

- $\pi_1 = (A1(0), B1(1), B2(2), B3(3), C3(4))$
- $\pi_2 = (C1(0), B1(1), B2(2), B3(3), A3(4))$

Poté je spuštěn Low-level algoritmus, který iniciuje generování proměnných za pomoci algoritmu GENERATE-DECISIONS. Ten vygeneruje proměnou pro každý přechod $\varepsilon^t_{u,v}(a_i)$ a čekání

$\chi_v^t(a_i)$, které je dosažitelné v čase $t \leq \mu$. Poté je množina VAR společně se Σ^R a μ převedena na $\mathcal{F}(\mu)$. Ta je poté ohodnocena pomocí SAT řešiče.

Zde nacházíme množství kolizí, které se poté propagují do množiny $conflicts$. Jako příklad zajímavých konfliktů, které jsou vygenerovány můžeme uvést na tyto dva:

- $\neg\varepsilon^1_{A1,B1}(1) \vee \neg\varepsilon^1_{C1,B1}(2)$
- $\neg\chi^1_{B1}(1) \vee \neg\chi^1_{B1}(2)$

Ty vyjadřují kolizi v čase $t = 1$ ve vrcholu $B1$ a hranový konflikt při přechodu do tohoto vrcholu dvou agentů zároveň. Tyto konflikty jsou poté propagovány do $\mathcal{F}(\rho)$ a $conflicts$. Tento cyklus je vyhodnocován dokud SAT řešič nepotvrdí, že tento problém není řešitelný v čase definovaném cenovou funkcí μ (Tento fakt je jednoduše viditelný, nicméně pro algoritmus netriviální).

Poté je cenová funkce inkrementována tak, aby byla v čase definovaném μ dosažitelná další proměnná $\chi_v^{t(\mu)}(a)$ libovolným agentem a. S tímto novým časovým omezením je znovu spuštěna Low-level funkce, která znovu provádí předchozí úkony. V další iteraci musí být $\mu = 5$, což je zaručeno naší definicí problému. Poté je znovu prováděn cyklus generování proměnných, nalezení splnitelné formule, detekce konfliktů a rozšiřování teorie, dokud algoritmus nedorazí na splnitelnou teorii $\mathcal{F}(\rho)$, která je převedena na plán neobsahující žádné kolize. Ten bude mít po převedení na letový plán v závislosti na implementaci algoritmu a SAT řešiče jednu ze dvou forem. Buď:

- $\pi_1 = (A1(0), B1(1), B1(2), B2(3), B3(4), C3(5))$
- $\pi_2 = (C1(0), B1(1), B2(2), B3(3), A3(4))$

a nebo:

- $\pi_1 = (A1(0), B1(1), B2(2), B3(3), C3(4))$
- $\pi_2 = (C1(0), B1(1), B1(2), B2(3), B3(4), A3(5))$

.

Tento plán je pak po validaci předán zpět High-level algoritmu, který navrátí informaci o nalezení plánu splnitelného v čase $t(\rho)$.

Kapitola 3
Crazyflie prostředí

V této práci budeme používat prostředí pro kontrolu a navigaci dronů švédské společnosti Bitcraze. Ta se specializuje na vývoj hardwarové a softwarové podpory jejich dronů Crazyflie 2.1.

3.1 Dron Crazyflie 2.1

Dron Crazyflie 2.1 je programovatelná létající platforma s vlastním vývojovým prostředím, spravovaným společností Bitcraze. Váží 27 g a je vybavena nízkolatenčním rádiem nRF24LU1 s dlouhým dosahem schopným komunikovat s Crazyradio PA vysílačem na vzdálenost vyšší než jeden kilometr. Tato platforma podporuje velké množství přídavných desek, které rozšiřují její funkcionalitu o lokalizační prvky, led display nebo další senzory podporující letové vlastnosti [15]. Pro naše účely budeme používat rozšiřující desku Loco positioning deck, která umožňuje lokalizaci dronu v omezeném prostředí představenou v sekci 3.3.

Obrázek 3.1 Dron Crazyflie 2.1

3.2 Crazyradio PA

Crazyradio PA je vysokodosahový rádiový USB dongle, postavený na čipu nRF24LU1+ od firmy Nordic Semiconductors, který používá pro komunikaci s Crazyflie 2.1 platformou "Enhanced ShockBurst™" paketový protokol. Tento protokol umožňuje vysílat 32-bajtové pakety dronům

a přijímat 1-bajtový potvrzující packet a 32-bajtovou odpověď. Crazyradio PA dokáže také komunikovat s ostatními Crazyradio PA dongly [15].

Obrázek 3.2 Crazyradio PA

3.3 Systém Loco Positioning

Loco Positioning systém určuje absolutní polohu dronů v 3D prostoru za pomoci Ultra-Wideband rádiového signálu. Základem systému jsou uzly (Anchors), které jsou rozmístěné v prostoru a fungují jako referenční body. Ty vysílají krátké vysokofrekvenční rádiové zprávy značkám (Tag). Podle těchto zpráv jsme pak schopni spočítat absolutní polohu těchto značek [15].

3.3.1 Uzly a desky

Uzel Loco Positioning lze využít buď jako uzel nebo jako značku v Loco Positioning systému. V režimu uzlu tedy vysílá zprávy a v režimu značky je schopný zprávy přijímat a vypočítat svoji polohu v systému.

Deska Loco Positioning rozšiřuje funkcionalitu dronu Crazyflie 2.1 a umožňuje mu fungovat jako jako značka v systému Loco Positioning. Tato deska přijímá zprávy od jednotlivých uzlů v systému a z těchto zpráv je schopna dopočítat svoji absolutní polohu v systému. Drony vybavené touto deskou jsou schopny automaticky detekovat současný mód systému a automaticky se přepnou do správného módu [15].

3.3.2 Sestavení systému

Pro co nejefektivnější sestavení Loco Positioning systému je zapotřebí konzistentní nastavení uzlů. Toho můžeme dosáhnou například za použití Loco Positioning System Tools [15]. Po připojení jednotlivých uzlů přes micro USB můžeme v tomto programu aktualizovat firmware a nastavit uzly do požadovaného módu. Tyto uzly by pak měly být podle využívaného módu rozestavěny do referenčního rozmístění na obrázku 3.3 nebo obrázku 3.4 tak, aby byly uzly alespoň patnáct centimetrů od zdí a země. Při použití těchto rozmístění dosahuje systém optimální přesnosti.

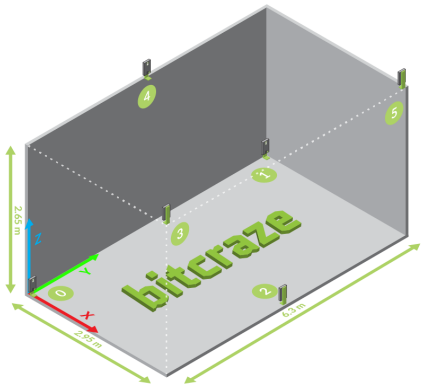
V případě použití jiného rozmístění je doporučeno zajistit, aby byly všechny uzly vzdáleny od ostatních alespoň dva metry a v různé výšce. Důvodem je zamezení jevu *Rozmělnění přesnosti* [16].

Při použití tohoto systému je doporučeno spouštět drony ve specifické poloze v souřadnicovém systému zobrazeném na obrázku 3.5, jelikož systém nedokáže určit orientaci dronu po spuštění, což může negativně ovlivnit jeho letové vlastnosti.

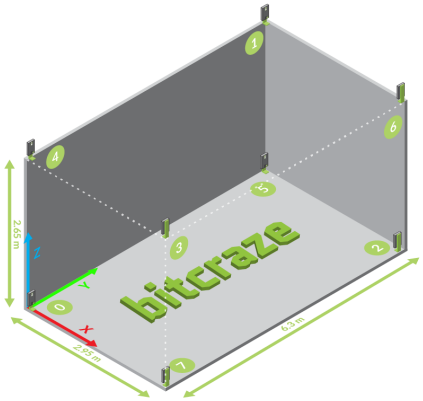
3.3.3 Módy systému

Celý systém se skládá z několika Loco Positioning uzlů (počet je závislý na použitém módu) a Loco Positioning desek, které mezi sebou komunikují za pomoci Loco Positioning Protokolu (LPP). Tento protokol umožňuje vysílat vytyčovací zprávy mezi jednotlivými uzly nebo mezi

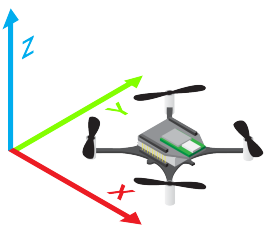
Obrázek 3.3 Referenční rozmístění uzlů pro TWR mód



Obrázek 3.4 Referenční rozmístění uzlů pro TDoA nebo TWR mód



Obrázek 3.5 Doporučené postavení dronu při spuštění



uzly a Loco Positioning deskou a také umožňuje vysílat konfigurační zprávy, které se používají pro nastavení uzlů a desek [15].

K detekci pozice v systému je využíván UWB signál. Tento způsob ovšem může detekovat pozici v systému špatně z mnoha různých důvodů. Nejčastějším důvodem jsou odrazy signálu v případech, kdy není dron přímo viditelný nebo pokud se signál dostává k dronu z více směrů. Pro zmírnění dopadů těchto chyb na výsledné měření polohy je v systému využíván takzvaný „Rozšířený Kalmanův filtr“ (EKF). Tento filtr bere v potaz měření interních senzorů dronu Crazyflie 2.1 a externí měření pozice jednotlivých uzlů Loco Positioning systému. Přesnost interních měření může být zlepšena přidáním přídavné desky Flow, která nám při vypracování této práce nebyla k dispozici.

3.3.3.1 TWR mód

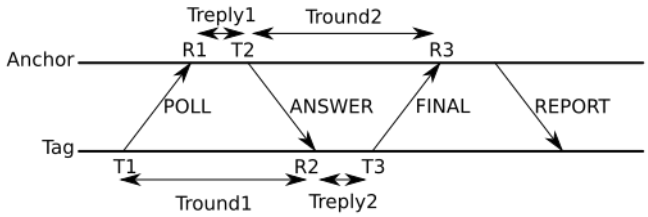
V módu TWR („Two Way Ranging“) si značka vyměňuje zprávy s jednotlivými uzly v daném pořadí. Schéma této komunikace je vidět na obrázku 3.6. Značka nejdříve vyšle zprávu se specifickým identifikátorem této komunikace a je obsažena ve všech dalších zprávách. Tato zpráva je ve schématu ilustrována jako POLL. Následně odešle uzel odpověď (ANSWER), ve které odešle značce svoje souřadnice v systému. Značka poté odpoví zprávou obsahující pouze identifikátor komunikace (FINAL), na kterou pak reaguje uzel zprávou REPORT. Tato zpráva obsahuje tři časové informace - $R1$, $T2$, $R3$ a zároveň pomocné systémové informace o stavu uzlu. Za pomoci těchto informací dokáže značka dopočítat délku časových úseků $Treply1$, $Treply2$, $Tround1$ a $Tround2$. Z těchto informací pak může dopočítat ToF („Time of Flight“), aplikací formule:

$$ToF = \frac{Tround1 \times Tround2 - Treply1 \times Treply2}{Tround1 + Tround2 + Treply1 + Treply2} \tag{3.1}$$

Za pomoci ToF pak dokáže značka spočítat svoji vzdálenost od uzlu. Pro lokalizaci v trojrozměrném prostoru je zapotřebí alespoň čtyř uzlů, nicméně pro optimální funkci systému je doporučováno použití alespoň šesti, abychom zajistili redundanci pro případ chyby a vyšší přesnost lokalizace.

Tento mód určuje polohu značek s nejlepší přesností z možných módů a funguje i v případě, že značka opustí oblast vytyčenou těmito uzly. Jeho hlavní nevýhodou je nemožnost lokalizace více značek zároveň, tedy i přes vysokou přesnost je nevhodný pro navigaci více dronů současně[15].

Obrázek 3.6 Schéma komunikace TWR protokolu



3.3.3.2 TDoA

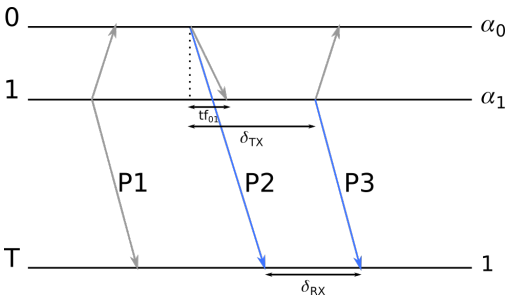
TDoA (z anglického „Time Difference of Arrival“) je způsob lokalizace založený na komunikaci uzlů mezi sebou za pomoci TWR módu. Značky pouze pasivně naslouchají této komunikaci a za pomoci časových rozdílů mezi příchody zpráv od jednotlivých uzlů jsou schopny dopočítat svoji polohu.

Jelikož jednotlivé uzly mezi sebou komunikují v TWR módu, značka které této komunikaci naslouchá je schopna odposlechnout informace mezi uzlem 0 a uzlem 1 (tf_{01} a δ_{TX}) 3.7. Dále má

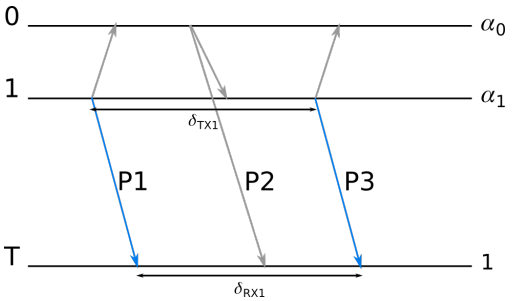
k dispozici informaci o čase, ve kterém se k ní zprávy vyslané uzlem 0 a 1 dostaly, ze kterých je schopna dopočítat δ_{RX}. Jelikož hodiny v každém ze zařízení pracují nezávisle na sobě, je nutné provést korekci těchto časových údajů. Toho značka docílí za pomoci časových rozdílů mezi vysláním a přijetím dvou po sobě jdoucích zpráv z jednoho uzlu, v tomto případě uzlu 1, δ_{TX1} a δ_{RX1}. 3.8 Výsledný TDoA pak vypočítá následovně:

$$TDoA = \delta_{RX} - (\delta_{RX1}/\delta_{TX1} \times \delta_{TX}) \tag{3.2}$$

Obrázek 3.7 Vzdálenost mezi zprávami uzlů 1 a 2



Obrázek 3.8 Korekce vzdálenosti mezi odesláním a přijetím zprávy



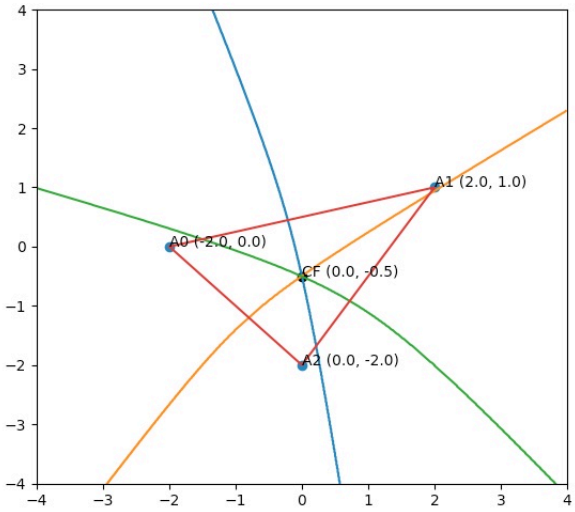
Za pomoci výsledných TDoA mezi alespoň čtyřmi uzly uzlů je značka schopna dopočítat svoji polohu v prostoru. Příklad lokalizace za pomoci TDoA v dvourozměrném prostoru je vidět na obrázku 3.9.

3.3.3.3 TDoA2 a TDoA3 módy

Loco Positioning systém podporuje dva různé módy využívající TDoA princip. TDoA2 mód používá plánovanou komunikaci, která je inicializována hlavním uzlem v systému. Tento uzel se stará o synchronizaci komunikace a obsahuje 8 časových oken, ve kterých spolu komunikují vždy dva uzly tak, aby v každém cyklu každý komunikoval alespoň jednou. Z tohoto důvodu tento mód podporuje maximálně osm uzlů současně a všechny musí být v dosahu hlavního uzlu, aby byl schopen tuto komunikaci korigovat. Tento přístup zajišťuje, že v systému nedochází ke kolizím komunikací mezi jednotlivými uzly, což zjednodušuje proces naslouchání komunikaci pro značky v systému. Nevýhodou tohoto módu je omezená plocha pokrytí systému a riziko selhání celého systému v případě selhání hlavního uzlu.

Mód TDoA3 odstraňuje některé limitace módu TDoA2 na úkor pravidelnosti komunikace a výsledné přesnosti. V tomto módu se uzly nesynchronizují vzhledem k hlavnímu uzlu, ale každý funguje samostatně, díky čemuž systém může fungovat dále i v případě selhání některého z uzlů. Tento přístup navíc odstraňuje limitaci na na maximální počet uzlů v systému. Může tak pokrývat teoreticky libovolně velký prostor. Značky v tomto módu musí být schopny řešit

Obrázek 3.9 Příklad lokalizace za pomoci TDoA principu. Barevné čáry reprezentují konstantní hodnoty TDoA mezi jednotlivými uzly. Modrá mezi A0 a A1, žlutá mezi A0 a A2 a zelená mezi A1 a A2



případné kolize v komunikaci. Tyto kolize obvykle řeší zahozením jedné nebo obou kolidujících informací, což ovšem negativně ovlivňuje přesnost lokalizace, jelikož pak musejí čekat na další relevantní informace potenciálně delší dobu než tomu je u módu TDoA2 [15].

3.4 Cflib

Cflib je knihovna vyvíjená firmou Bitcraze za účelem vytvoření uživatelského rozhraní vhodného pro komunikaci s drony Crazyflie 2.1. Tato knihovna je plně open-source a dává uživateli přístup k široké škále funkcionalit, které zprostředkovávají reálnou komunikaci s drony a většinou související navigační technologie [15]. Za pomoci této knihovny budeme komunikovat s drony v námi implementované aplikaci *crazyflie_controller*.

Pro ovládání dronů jsou v této knihovně implementovány dva balíčky. Jedním je balíček *Crazyflie*, který umožňuje jednoduchou komunikaci a navigaci dronu a druhým je *Swarm*. Ten staví na funkcionalitách balíčku *Crazyflie* a rozšiřuje je pro jednodušší komunikaci s letkou dronů a exekuci synchronních pohybů. Pro účely naší práce ovšem implementace tohoto balíčku obsahuje zásadní omezení, jelikož neumožňuje jednoduchou kontrolu nad libovolnými podmnožinami synchronizovaných dronů. Z tohoto důvodu se budeme v této práci opírat o balíček *Crazyflie*.

3.5 RoboAgeLab

Pro testování a provádění experimentů nám bylo v průběhu práce přístupné zázemí v RoboAgeLab na Fakultě informačních technologií ČVUT. Ta laboratoř je vybavena technikou a prostory umožňujícími práci s drony. K pokusům je zde dostupných dvacet dronů Crazyflie 2.1, každý z nich označený unikátním identifikátorem (uri). dva rádiové ovladače Crazyradio PA a voliéra umožňující let dronů v prostoru $1.8 \times 1.8 \times 2$m, ohraničená osmi Loco Positioning uzly. Je vhodné zmínit, že rozměry voliéry jsou menší než doporučené (3.3.3), což může způsobovat nepřesnost lokalizace. Výsledky pozdějších experimentů podporují tuto teorii.

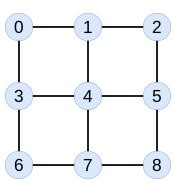
Kapitola 4

Transformace nalezených cest na navigační sekvence

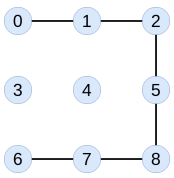
V této kapitole si představíme princip převodu cesty nalezené MAPF algoritmem na navigační sekvenci pro drony Crazyflie 2.1. Jelikož implementace MAPF algoritmů je mimo rozsah této práce, budeme používat již existující balíček algoritmů *boOX* [17], který je aktivně vyvíjen v rámci výzkumu MAPF problematiky na FIT ČVUT. Ten obsahuje implementaci mnoha algoritmů použitelných jako vstup pro náš navigační systém, primárně řešič utilizující MAPF^{R} algoritmus SMT-CBS^{R}, který jsme představili dříve v této práci. Originálním účelem balíčku *boOX* je použití na problémy definované pouze v dvourozměrném prostoru pro kruhové agenty, kteří jsou v implementaci označováni jako „kruhoboti“. Tento fakt silně omezuje množinu nám dostupných řešení, nicméně postačí pro ukázku funkcionality transformace navigační sekvence pro drony. Námi představený systém samozřejmě podporuje i navigační sekvence definované ve trojrozměrném prostoru.

4.1 Implementace

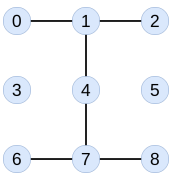
Abychom mohli použít výše zmíněný SMT-CBS^{R} řešič, musíme si nejprve definovat prostor, ve kterém se budeme pokoušet o nalezení navigačních sekvencí pro drony. Tímto prostorem bude voliéra v RoboAgeLab. Tu musíme definovat jako graf $G = (V, E)$. Dále v tomto grafu potřebujeme definovat pro každý vrchol $v \in V$ souřadnice $v.x$, $v.y$ (a volitelně i $v.z$) v reálném prostoru a hrany mezi těmito vrcholy. Pro účely demonstrace fungování grafu jsme vytvořili tři takovéto grafy na kterých provedeme hledání cest za pomoci balíčku *boOX* které můžeme vidět na 4.1.



(a) Voliéra bez překážek



(b) Voliéra s nedostupnými vrcholy 3 a 4



(c) Voliéra s nedostupnými vrcholy 3 a 5

Obrázek 4.1 Grafy reprezentující voliéru pro účely použití MAPF^{R} balíčku *boOX*

Tyto grafy se skládají z devíti vrcholů očíslovaných nula až devět a hran, které jsou znázorněny čamarami spojujícími tyto vrcholy. Grafy 4.1b a 4.1c obsahují vrcholy, které se zbytkem grafu nespojují žádné hrany. Tím je signalizováno, že ve vrcholech se nachází překážka a nelze skrze tento vrchol naplánovat cestu dronu.

Grafy jsou reprezentovány textovou formou v následujícím formátu:

```
Locations: n
v0: x, y
...
vn: x, y
Vertices: n
Edges: m
{vi, vj}
...
```

Kde n označuje počet vrcholů v daném grafu, $vi : x, y$ určuje pozici vrcholu vi a m určuje počet hran, které se v grafu nachází, specifikované poté ve formátu $\{vi, vj\}$ což reprezentuje existenci hrany mezi vrcholy vi a vj. Tento soubor poté využívá *boOX* MAPF$^{\mathrm{R}}$ řešič společně se souborem *.kruR*, ve kterém jsou definovány vlastnosti agentů – zde je specifikována jejich rychlost a poloměr prostoru který zabírají, vrcholy ve kterých se agenti na počátku nachází a vrcholy cílové.

Z těchto informací je *boOX* schopen nalézt bezkonfliktní letové plány pro jednotlivé agenty a poté je uložit v následujícím formátu:

```
Kruhobot schedules [
    schedule [1]: {
        v1 --> v2 [t0, t1]
        v2 --> v3 [t1, t2]
        ...
    }
    schedule [2]: {
        ...
    }
    ...
]
```

Ten specifikuje jednotlivé plánované trasy (*schedules*) pro libovolné množství agentů. Každá z těchto tras je pak definovaná množinou přechodů mezi vrcholy ($v1 - - > v2$) a časovým intervalem, který tento přechod trvá ($[t0, t1]$). V případě, že $v1$ a $v2$ označují stejný vrchol, jedná se o akci „čekání“.

4.1.1 crazyflie_controller

4.1.1.1 commander

Knihovna *cflib* sice umožňuje kontakt s drony a jejich navigaci, nicméně její použití stále vyžaduje hlubší pochopení dílčích procesů, které jsou nutné provést pro jednoduché ovládání. Cílem balíčku *crazyflie_controller* je umožnit uživateli ovládat drony přímočařejším způsobem a implementuje další pomocné funkce k vyhodnocování výsledků daných letových sekvencí.

crazyflie_controller zjednodušuje komunikaci s drony za pomoci modulu *commander*, který zpracovává dílčí úkony nutné k navigaci dronů a redukuje je na práci s třídou *Commander*. Tato třída při inicializaci navazuje spojení s drony a provádí další úkony, které jsou s navigací spojené. Při inicializaci používá následující parametry:

- *uris* - Vstupem tohoto parametru je množina unikátních identifikátorů dronů, se kterými chceme navázat spojení a inicializovat je.
- *enable_logging* - Tento parametr určuje, zda plánujeme ukládat informace o provedených letech do .csv souborů (pokud tento parametr nespecifikujeme, pak je nastaven na hodnotu $False$, abychom zamezili tvorbě nevyžádaných souborů.)

Při vytvoření třídy proběhne inicializace ovladačů nutných k fungování dronů a za pomoci modulu *SyncCrazyflie* z knihovny *cflib* je navázáno spojení s drony. Pokud je parametr *enable_logging* nastaven na hodnotu $True$, pak je po inicializaci také vytvořeno vlákno, které inicializuje moduly *LogConfig* a *SyncLogger* a připraví soubor, do kterého bude po spuštění zaznamenáván let každého z dronů. Poté je třída připravena na přijímání úkonů, a můžeme s ní interagovat za pomoci příkazů, které jsou implementovány jako veřejné metody této třídy.

U všech metod které přijímají jako vstupní parametr URI je nutné, aby bylo specifikováno jedno z URI dronů, které byly inicializovány při tvorbě třídy *Commander*, jinak metoda nebude mít žádný efekt nebo selže. Specificky metody, které se starají o přesun v prostoru (*take_off*, *land* a *fly_to*) používají ke komunikaci s dronem *high_level_commander*, který za pomoci systému Loco Positioning umožňuje navigaci v absolutních souřadnicích ve vztahu k systému.

Jedná se o následující metody:

- $take\_off(height, *uris)$

Metoda při zavolání vytvoří vlákno pro každé ze specifikovaných URI. Pomocí něho poté vyšle každé vlákno zprávu danému dronu, aby vzlétl. Výška do které drony vzlétnou je definována parametrem *height* v metrech. Pokud se dron již nachází ve vzduchu, není tento příkaz vyslán.

- $land(*uris)$

Tato metoda funguje obdobně jako metoda *take_off*. Také vytvoří vlákno pro každé ze specifikovaných URI. Ta vyšlou zprávu jednotlivým dronům s příkazem pro přistání. Pokud je dron na zemi, příkaz nebude vyslán.

- $fly\_to(time\_to\_execute, position\_dict)$

Po zavolání tato metoda přijímá parametr *time_to_execute*, který udává, kolik času (v sekundách) by měl každý z definovaných přesunů zabrat. Tyto úseky jsou pak specifikovány jako slovník obsahující páry $uri : (x, y, z)$, které obsahují URI dronu a souřadnice x, y, z v prostoru, do kterých by se dron měl přesunout. Pro každé z URI je potom vytvořeno vlákno, které se stará o zasílání příkazů danému dronu s informací o pozici, které musí dosáhnout a času, který zbývá pro dosažení těchto souřadnic. Dron poté setrvává v tomto prostoru, dokud nedostane další příkaz. Toho je dosaženo tak, že vlákno pravidelně vysílá dronu příkaz, aby setrval v daných souřadnicích. Pokud by dron nedostával tyto příkazy, hrozilo by, že by se postupem času vychýlil z pozice vlivem externích jevů.

- $start\_logs()$
- $end\_logs()$

start_logs při zavolání vyšle informaci všem existujícím logovacím vláknům, aby začala zaznamenávat informace o letu. Zavoláním metody *end_logs* toto logování přerušíme.

Třída *Commander* za pomoci těchto metod zjednodušuje kontrolu nad drony a lze ji využít v libovolných externích aplikacích pro ovládání dronů Crazyflie 2.1 s minimálním vhledem do knihovny *cflib*.

Modul *commander* navíc implementuje další funkcionalitu, která umožňuje převod plánu vygenerovaného řešičem SMT-CBS^{R} na navigační sekvenci pro drony za pomoci třídy *Commander*. Případně se dá použít pro libovolný plán, který je implementován ve stejném formátu jako je výstup tohoto řešiče. Tato funkcionalita je přístupná v podobě funkce:

- $execute\_plan(plan\_filepath, map\_filepath, uris, log)$

Ta postupně provádí tři elementární kroky. V prvním kroku provede zpracování souboru s plánem identifikovaným parametrem $plan\_filepath$ a souboru s grafem mapujícím vrcholy grafu do prostoru $map\_filepath$. Tyto dva soubory převede na slovník, který mapuje jednotlivé plány k agentům zadaným v listu $uris$. Navíc také zaznamená pozice, ze kterých mají tito agenti vzlétnout při provádění plánu. Pokud v těchto plánech není specifikovaný třetí rozměr (tedy výška dané pozice), doplní v tomto kroku funkce výchozí hodnotu letové výšky.

Ve druhém kroku pak slovník s plánem převede funkce na sekvenci letových instrukcí, která je reprezentována listem definujícím posloupnost pokynů, které je nutno vykonat v následujícím kroku.

Poté je inicializována třída *Commander*, která naváže spojení s drony a poté předá ve formátu textového výstupu uživateli instrukce, jak drony rozestavit v prostoru před startem. Po stisknutí libovolné klávesy provede funkce vzlet se všemi drony a začne postupně provádět příkazy uvedené v sekvenci instrukcí. Po dokončení této sekvence všechny drony přistanou.

Tato funkce má další parametr *log*, který je ve výchozím nastavení vypnutý, nicméně po zapnutí umožňuje logování pozice dronů při provádění letu.

4.1.1.2 plotter

Modul *plotter* je součástí balíčku *crazyflie_controller* a používá se k grafickému zobrazení souborů vytvořených drony při logování letů. Je také schopen načítat data, která jsou přidána do souboru v pravidelných intervalech a tedy mapovat let dronu v reálném čase. Modul je primárně postaven na vizualizační knihovně Matplotlib [18], které se používá pro tvorbu grafů, specificky pak na animačním submodulu *animation*.

plotter používá k zobrazování třídu *Plotter*. Ta je ovládána pomocí metod:

- $add\_obstacle(position, size)$

Metoda $add\_obstacle(position, size)$ přidává překážky do výsledného zobrazení. Tyto překážky mají formu hranolu. Tento hranol je určen vrcholem *position*, který je definován trojicí (x, y, z) určující jeho pozici v prostoru a velikostí *size*, která udává jeho rozměry ve směru jednotlivých os za pomoci trojice (x_s, y_s, z_s).

- $plot\_from\_csv(refresh\_rate, *args)$

Tato metoda umožňuje vizualizovat libovolné množství souborů, reprezentovaných jako trojice $(color, cf\_name, file)$. Jednotlivé složky postupně akceptují: označení barvy, označení dronu, které bude pro tento soubor použito v legendě zobrazení a cestu k souboru, ze kterého se mají data načítat. Metoda používá ještě jeden parametr – $refresh\_rate$. Tento parametr definuje jak často (v ms) budou aktualizovány data, které jsou načítány ze souborů.

Za pomoci těchto jednoduchých metod budeme tvořit vizualizace provedených letových sekvencí v následujících sekcích.

4.1.2 eval

eval je posledním z hlavních modulů balíčku *crazyflie_controller*. Tento modul obsahuje množství pomocných funkcí, sloužících k vyhodnocování jednotlivých experimentů, které provádíme v poslední části této práce. Implementovány jsou zde následující funkce:

- $create\_reference\_file(f\_input, f\_output, segments)$

Funkce $create\_reference\_file(f\_input, f\_output, segments)$ slouží k tvorbě referenčních souborů obsahujících předpokládanou cestu, kterou lze v experimentech používat k vyhodnocení kvality letové trajektorie jednotlivých dronů. Tato funkce v parametru f_input přijímá cestu k souboru, který obsahuje jednoduchou sekvenci pozic, přes které se má dron v dané letové sekvenci přesouvat. Každé dvě pozice potom rozšíří na množinu ekvidistantních segmentů, které jsou definované parametrem $segments$. To nám poté při vyhodnocování kvality letu umožňuje jednoduché porovnání pozice dronu v určitém časovém segmentu s pozicí předpokládanou. Takto rozšířená sekvence je poté uložena do souboru f_output.

- $idle\_localization\_stats(csv\_path)$
- $idle\_localization\_list(csv\_list)$

První z těchto funkcí je určena k vyhodnocování přesnosti lokalizace systému Loco Positioning. Očekává na vstupu v parametru csv_path soubor s daty naměřenými dronem setrvávající v jedné pozici po určitou dobu. Výstupem této funkce je pak několik ukazatelů. Střední bod všech nahlášených pozic, list odchylek všech měření od tohoto bodu, průměrná odchylka měření, střední kvadratická chyba měření a maximální naměřená odchylka.

Druhá funkce vyhodnocuje více měření za pomoci $idle\_localization\_stats(csv\_path)$. Tato funkce je používána k vyhodnocení výsledků více měření zároveň. Na výstup vrací průměr odchylek naměřených v rámci všech měření, maximální průměrnou chybu naměřenou v rámci jednoho měření, kvadratickou chybu všech naměřených hodnot a maximální kvadratickou chybu naměřenou v rámci jednoho měření.

- $evaluate\_flight(expected\_path, actual\_path)$
- $evaluate\_experiment(experiment\_dir, reference)$

Tyto funkce slouží k vyhodnocení kvality letových sekvencí dronu. Funkce $evaluate\_flight(expected\_path, actual\_path)$ přijímá na vstupu soubor $expected\_path$ obsahující log mapující let dronu. Parametr $actual\_path$ obsahuje soubor definující očekávanou trasu letu. Tyto dvě trasy pak funkce srovná a na výstupu vrací množinu absolutních odchylek každého segmentu trasy od očekávané pozice, průměrnou odchylku a kvadratickou odchylku.

Funkce $evaluate\_experiment(experiment\_dir, reference)$ pak jako parametr $experiment\_dir$ přijímá složku obsahující .csv soubory s logy, které mají být vyhodnoceny oproti referenčnímu souboru, který je předán v parametru $reference$. Tato funkce poté provede vyhodnocení všech těchto logů a vypíše průměrnou chybu ve všech segmentech všech měření, lety s nejlepší a nejhorší průměrnou odchylkou a lety s nejlepší a nejhorší kvadratickou odchylkou.

4.2 Příklad provedení letové sekvence

Pro příklad použití námi vytvořeného balíčku si uvedeme provedení dvou různých navigačních sekvencí.

První z nich bude na grafu 4.1b který je v textové formě reprezentován následovně:

```
Locations: 9
0: 0.400, 0.400
1: 0.400, 1.000
2: 0.400, 1.600
3: 1.000, 0.400
4: 1.000, 1.000
5: 1.000, 1.600
6: 1.600, 0.400
```

```
7: 1.600, 1.000
8: 1.600, 1.600
Vertices: 9
Edges: 6
{0,1}
{1,2}
{2,5}
{5,8}
{8,7}
{7,6}
```

Na tomto grafu se pokusíme nalézt vhodný letový plán pro dva drony z vrcholu 0 do 7 a z vrcholu 1 do 8. K tomu použijeme SMT-CBS^{R} řešič z balíčku *boOX*. Ten nalezl následující cestu:

```
Kruhobot schedules [
    schedule [1]: {
        0 --> 1 [0.000, 3.000]
        1 --> 2 [3.000, 6.000]
        2 --> 5 [6.000, 9.000]
        5 --> 5 [9.000, 10.414]
        5 --> 8 [10.414, 13.414]
        8 --> 7 [13.414, 16.414]
    }
    schedule [2]: {
        1 --> 2 [0.000, 3.000]
        2 --> 5 [3.000, 6.000]
        5 --> 8 [6.000, 9.000]
        8 --> 7 [9.000, 12.000]
        7 --> 6 [12.000, 15.000]
        6 --> 6 [15.000, 16.414]
    }
]
```

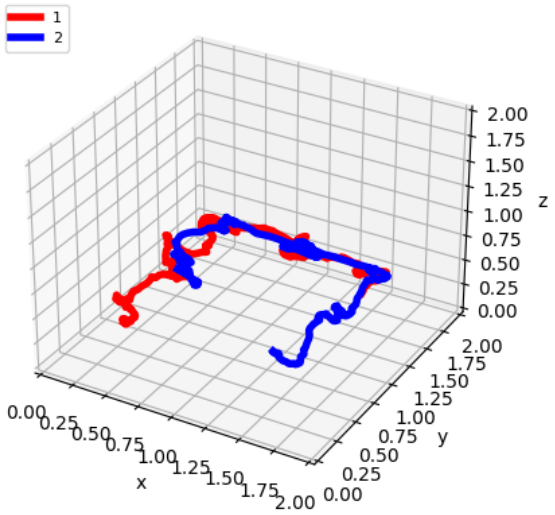
Tyto soubory nám stačí pro provedení letové sekvence. Za pomoci funkce $execute\_plan(plan\_filepath, map\_filepath, uris, log)$, které v parametru $plan\_filepath$ předáme cestu k souboru s nalezeným plánem a v parametru $map\_filepath$ reprezentaci prostoru ve formě grafu. Dále musíme funkci předat množinu URI dronů, se kterými má navázat spojení a použít je k provedení letové sekvence. Pro účely vyhodnocení také nastavíme hodnotu parametru *log* na $True$.

Tato funkce následně převede oba soubory na letovou sekvenci a vyzve nás, abychom drony rozmístili do prostoru v následujícím rozmístění:

```
<uri_dronu1>: 0.400, 0.400
<uri_dronu2>: 0.400, 1.000
```

Po rozmístění dronů stiskem libovolného tlačítka odstartujeme provádění sekvence za pomoci modulu *commander*. Funkce poté zajistí vzlet dronů a provedení letové sekvence. Přitom (jelikož jsme povolili logování pozice dronů) zaznamenává funkce pozici dronů do .csv souborů. Tento let pak můžeme prostřednictvím modulu *plotter* vizualizovat (obrázek 4.2).

Obrázek 4.2 Ukázka provedení letové sekvence



Kapitola 5
Experimenty

Pro vyhodnocování v experimentální část této práce jsme implementovali moduly *plotter* a *eval*, které jsou také součástí balíčku *crazyflie_controller*. Tyto moduly slouží k mapování letu jednotlivých dronů, případných překážek v nich a k získání prezentovaných metrik.

Ohodnocení kvality námi vytvořeného navigačního systému pro drony je ovšem poněkud složité, jelikož nemáme relevantní výzkum se kterým výsledky systému porovnat. Nicméně můžeme vytvořit metodiku pro budoucí srovnání v následujících experimentech.

5.1 Test detekce systému Loco Positioning

Prvním experimentem bude zjištění přesnosti zaměření dronů v námi použité voliéře. Účelem tohoto experimentu je ověřit, zda a případně jak je závislá kvalita lokalizace na volbě konkrétního dronu a na pozici v rámci systému Loco Positioning v laboratoři a jakou můžeme očekávat přesnost v dalších experimentech.

Těchto zjištění se pokusíme docílit rozložením dronů do jednoduché čtvercové formace 5.1 a spuštěním logování pozice dronů v rámci systému na deset sekund. Toto měření opakujeme čtyřikrát a poté posuneme pozici jednotlivých dronů ve směru hodinových ručiček. Takto protočíme drony také čtyřikrát, budeme tedy pracovat s celkově šestnácti měřeními od každého dronu.

Naměřené hodnoty jsme shrnuli v tabulkách 5.1 a 5.2.

Tabulka 5.1 Naměřené chyby lokalizace u jednotlivých dronů

Dron	AVG [m]	Max. AVG [m]	MSE [m^2]	Max. MSE [m^2]
EA	0.0231	0.0331	0.0007	0.0013
EC	0.0370	0.1518	0.0046	0.0341
EE	0.0525	0.2905	0.0099	0.1085
F1	0.0364	0.1079	0.0033	0.0172

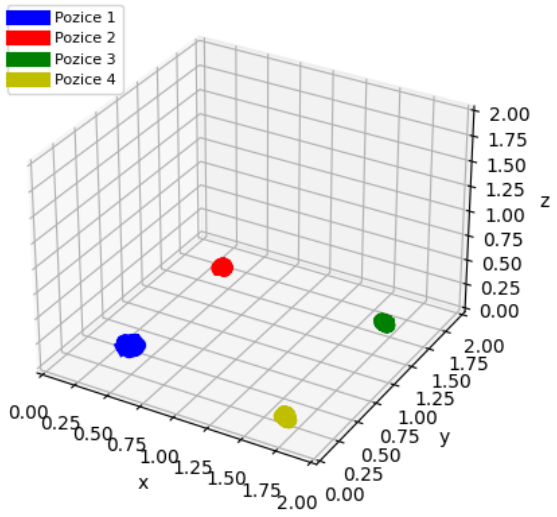
Při vyhodnocení experimentu jsme bohužel neměli přístup k přesné informaci o poloze dronu. Z toho důvodu jsme se museli uchýlit k vyhodnocení odchylky od bodu který jsme vypočítali jako střed všech bodů v daném měření. Jednotlivé hodnoty v tabulkách jsou určeny následovně:

$$AVG = \sum_{i=1}^{n} \sqrt{(x_s - x_i)^2 + (y_s - y_i)^2 + (z_s - z_i)^2}/n \tag{5.1}$$

Tabulka 5.2 Naměřené chyby lokalizace u jednotlivých pozic

Pozice	AVG [m]	Max. AVG [m]	MSE [m^2]	Max. MSE [m^2]
1	0.0910	0.2905	0.0171	0.1085
2	0.0235	0.0331	0.0007	0.0013
4	0.0191	0.0278	0.0005	0.0010
3	0.0153	0.0198	0.0003	0.0005

Obrázek 5.1 Pozice dronů v testu přesnosti lokalizace



$$MSE = \sum_{i=1}^{n} \sqrt{(x_s - x_i)^2 + (y_s - y_i)^2 + (z_s - z_i)^2}^2 / n \tag{5.2}$$

kde n je množina všech detekovaných bodů ve všech měřeních. Tedy AVG a MSE určují hodnoty v závislosti na všech naměřených pozicích v rámci všech měření, zatímco $Max.\ AVG$ a $Max.\ MSE$ označují nejhorší dosažené výsledky v rámci jednoho měření.

Z těchto výsledků lze usoudit, že volba dronu může ovlivnit daný experiment, jelikož přesnost detekce různých dronů se výrazně liší. Abychom se vyhnuli vlivu tohoto zjištění, budeme v následujících experimentech používat vždy pouze srovnání výsledků specifických dronů.

Dále z naměřených hodnot vyplývá, že pozice dronu také výrazně ovlivňuje přesnost měření. Tento jev může vyplývat z nevhodné kalibrace celého systému. Provedli jsme proto rekalibraci, která spočívá v přeměření pozice jednotlivých uzlů systému Loco Positioning relativně k počátku souřadnicového systému.

Po rekalibraci jsme dosáhli hodnot, které je možné vidět v tabulkách 5.3 a 5.4. Z těchto měření můžeme usuzovat, že přesnost lokalizace v pozici 1 se výrazně zlepšila na úkor přesnosti v ostatních pozicích. Snížené hodnoty maximálního MSE u jednotlivých dronů navíc indikují, že lokalizace nyní reportuje pozici dronu výrazně blíže reálné poloze. Přesnost lokalizace je nyní navíc daleko předvídatelnější než před rekalibrací, jelikož kvalita lokalizace je nyní méně závislá na pozici, ve které se dron nachází.

Tabulka 5.3 Naměřené chyby lokalizace u jednotlivých dronů po rekalibraci

Dron	AVG [m]	Max. AVG [m]	MSE [m^2]	Max. MSE [m^2]
EA	0.0411	0.0733	0.0024	0.0066
EC	0.0308	0.0482	0.0013	0.0033
EE	0.0356	0.0771	0.0018	0.0069
F1	0.0368	0.0564	0.0018	0.0038

Tabulka 5.4 Naměřené chyby lokalizace u jednotlivých pozic po rekalibraci

Pozice	AVG [m]	Max. AVG [m]	MSE [m^2]	Max. MSE [m^2]
1	0.0501	0.0771	0.0032	0.0069
2	0.0391	0.0675	0.0021	0.0058
4	0.0204	0.0278	0.0005	0.0010
3	0.0346	0.0551	0.0015	0.0038

5.2 Vliv překážky na detekci pozice dronu

V rámci druhého experimentu se pokusíme zjistit vliv přítomnosti překážek na detekci polohy dronu, případně dronů za klidného stavu. Jako překážku v tomto testu použijeme molitanovou desku, která bude umístěna do středu voliéry. Tato deska má rozměr $1 \times 1 \times 0.1$m. Při vyhodnocování experimentů budeme využívat stejné metriky, jaké byly použity v předchozím experimentu, jelikož se také jedná o stacionární experiment, a tedy můžeme vztahovat polohu dronu pouze samu k sobě. Měření budou provedena také stejnou metodikou, abychom mohli výsledky jednoduše porovnat. Výsledky experimentu lze vidět v tabulkách 5.5 a 5.6.

Tabulka 5.5 Naměřené chyby lokalizace u jednotlivých dronů při přítomnosti překážky

Dron	AVG [m]	Max. AVG [m]	MSE [m^2]	Max. MSE [m^2]
EA	0.0425	0.0687	0.0025	0.0060
EC	0.0323	0.0524	0.0014	0.0036
EE	0.0328	0.0596	0.0015	0.0043
F1	0.0366	0.0636	0.0018	0.0047

Tabulka 5.6 Naměřené chyby lokalizace u jednotlivých pozic při přítomnosti překážky

Pozice	AVG [m]	Max. AVG [m]	MSE [m^2]	Max. MSE [m^2]
1	0.0486	0.0687	0.0030	0.0060
2	0.0405	0.0622	0.0022	0.0048
4	0.0225	0.0294	0.0006	0.0009
3	0.0337	0.0437	0.0014	0.0023

Při porovnání těchto výsledků s hodnotami naměřenými při předchozím experimentu můžeme usoudit, že přítomnost překážky ve voliéře s drony nemá zásadní vliv na přesnost lokalizace systému. Tuto teorii podporuje i fakt, že výsledky jsou místy lepší než ty naměřené bez přítomnosti překážky.

5.3 Chyba horizontální navigační sekvence

V tomto experimentu se pokusíme kvantifikovat přesnost námi vytvořeného balíčku pro ovládání dronů při horizontálním pohybu. To provedeme postupně s přítomností reálné překážky a poté

bez ní, abychom mohli stanovit vliv její přítomnosti na kvalitu provedení letové sekvence. Ohodnocení bude provedeno porovnáním logované pozice dronu s předpokládanou pozicí postupně ve všech časových úsecích. Předpokládáme tedy lineární pohyb dronů a zanedbáváme případnou akceleraci.

Oba tyto lety provádíme současně vždy se stejným párem dronů čtyřikrát po sobě pro situaci s překážkou a následně bez překážky. Poté zaměníme drony za další pár, se kterým tento postup opakujeme. Výsledná data tedy byla nasbírána v průběhu šestnácti letů.

Ohodnocení letových tras můžeme vidět v tabulce 5.7. V té jsme pro každou z pozic (1) a (2) bez překážky a poté s překážkou (1 + P) a (2 + P) definovali AVG, tedy průměrnou odchylku reálné pozice od pozice očekávané a nejhorší a nejlepší naměřenou průměrnou odchylku. Také jsme ve vyhodnocení použili hodnotu střední kvadratické odchylky – nejlepší MSE a nejhorší MSE v rámci jednoho letu.

Z těchto výsledků lze usoudit, že letové trasy dronů za přítomnosti překážky mají mírně nižší kvalitu, nicméně rozdíl je v konečném důsledku zanedbatelný, jelikož se jedná o zhoršení v řádu několika procent.

Z grafů letů na obrázku 5.2 pak můžeme vidět, že drony při navigaci mají tendenci setrvávat v kritických bodech letové sekvence. To je pravděpodobně způsobeno implementací metody *fly_to*. Z vizualizací je navíc vidět, že drony končí letovou trasu před dosažením finální pozice. Tento jev je obtížně vysvětlitelný, jelikož v implementaci jsou dronům opakovaně vysílány příkazy s aktualizovaným zbývajícím časem k letu a drony by tedy měly ukončit let blíže cílové destinaci.

Tabulka 5.7 Chyby lokalizace naměřené u horizontálních cest

Pozice	AVG [m]	Nejhorší AVG [m]	Nejlepší AVG [m^2]	Nejhorší MSE	Nejlepší MSE [m^2]
1	0.3004	0.3167	0.2886	0.1164	0.0945
2	0.3034	0.3143	0.2931	0.1083	0.0968
1 + P	0.3112	0.3222	0.3013	0.1180	0.1035
2 + P	0.3091	0.3134	0.3010	0.1145	0.0972

5.4 Chyba vertikální navigační sekvence

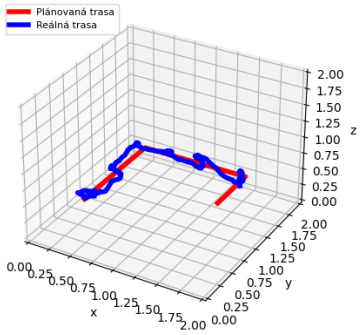
Posledním experimentem bude otestování naší implementace v situacích, kdy se musejí drony pohybovat vertikálně. V tomto experimentu opět provedeme opakovaně jednu letovou sekvenci, kterou budou současně provádět dva drony. Metodika bude také stejná jako v předchozím pokusu, tedy budeme používat stejný pár dronů, se kterým provedeme letovou sekvenci čtyřikrát po sobě, poté obměníme drony za druhý pár a sekvenci opakujeme. Nakonec provedeme stejný pokus s přítomností překážky. Tou v tomto experimentu bude znovu molitanová deska, nicméně tentokrát o rozměru $2 \times 1 \times 0.1$m. Výsledky budou ohodnoceny stejnými metrikami jako v předchozím experimentu.

Tabulka 5.8 Chyby lokalizace naměřené u vertikálních cest

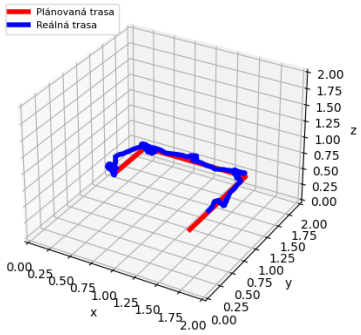
Pozice	AVG [m]	Nejhorší AVG [m]	Nejlepší AVG [m^2]	Nejhorší MSE	Nejlepší MSE [m^2]
1	0.2854	0.2963	0.2753	0.0996	0.0864
2	0.2869	0.3046	0.2731	0.1024	0.0851
1 + P	0.2810	0.3024	0.2647	0.1026	0.0814
2 + P	0.2854	0.3027	0.2704	0.1036	0.0821

Výsledky měření lze vidět v tabulce 5.8. Z těchto výsledků můžeme vyvodit závěr, že přítomnost překážky opět dramaticky nepostihuje kvalitu letových vlastností dronů. V porovnání

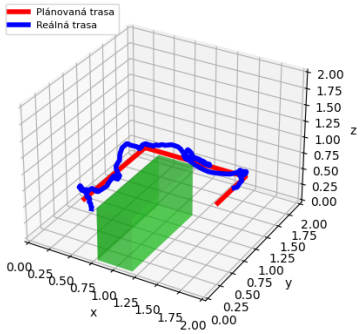
s chybou naměřenou u horizontálních letových sekvencí zde navíc můžeme vidět mírné zlepšení. Z vizualizací na obrázcích 5.3 můžeme vidět, že nejproblematičtější část trasy je v úsecích, kdy dron stoupá nebo klesá. Tento fakt je pravděpodobně způsoben pokusem o dočasné setrvání v bodě navigační sekvence, což narušuje souvislost dané akce.



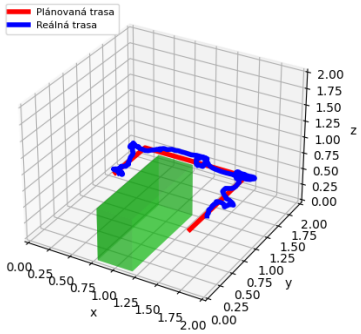
(a) Trasa dronu na pozici 1 bez překážky



(b) Trasa dronu na pozici 2 bez překážky

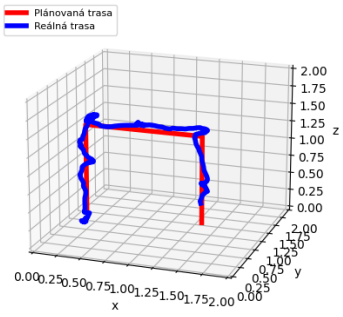


(c) Trasa dronu na pozici 1 s překážkou

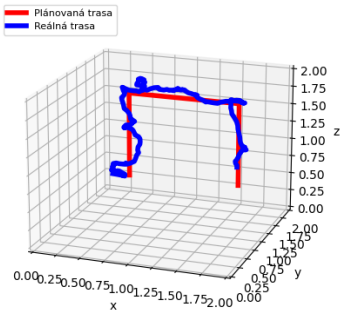


(d) Trasa dronu na pozici 2 s překážkou

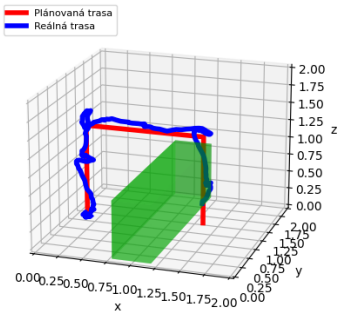
Obrázek 5.2 Vizualizace letových tras dronů u horizontálních cest



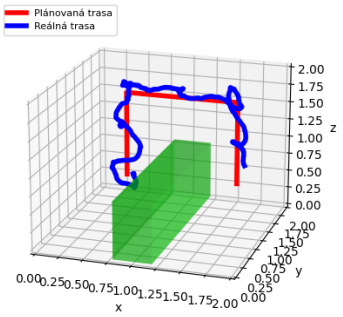
(a) Trasa dronu na pozici 1 bez překážky



(b) Trasa dronu na pozici 2 bez překážky



(c) Trasa dronu na pozici 1 s překážkou



(d) Trasa dronu na pozici 2 s překážkou

Obrázek 5.3 Vizualizace letových tras dronů u vertikálních cest

Kapitola 6

Závěr

Prvním z cílů práce bylo nastudování problému multiagentního hledání cest v prostoru. Toho jsme docílili v první kapitole, kde jsme definovali problematiku jednoagentního hledání cest. Tu jsme poté použili jako stavební kámen k představení multiagentního hledání cest v diskrétním čase, princip konfliktů, cenových funkcí a pro příklad fungování jsme uvedli algoritmus „Conflict-Based Search“, který je v současnosti jedním z oblíbených řešičů používaných v MAPF problematice. Následně jsme představili variantu MAPF pro spojitý čas a prostor, která je vhodněji formulována pro použití v reálném prostoru. V této části jsme také uvedli algoritmus SMT-CBS^{R}, který později používáme v našem navigačním systému.

V druhé části práce se jsme se seznámili s prostředím pro ovládání dronů Crazyflie 2.1, specificky pak s navigačním systémem Loco Positioning. Představili jsme zde tři módy tohoto systému, které je možné použít k lokalizaci dronu v prostoru. V této části byla také představena laboratoř RoboAgeLab. V té nám byl umožněn přístup k dronům a systémům nutným pro jejich ovládání a lokalizaci. Zde také byly prováděny experimenty v poslední části této práce.

V následující kapitole jsme pak představili námi vytvořený balíček *crazyflie_controller*, který obsahuje modul *commander* určený pro ovládání dronů a provádění letových sekvencí. Tyto sekvence byly generovány MAPF řešičem SMT-CBS^{R}, který je součástí balíčku *boOX*, který je aktivně vyvíjen na fakultě FIT ČVUT pro řešení MAPF problematiky. Tento balíček se bohužel omezuje na řešení problémů MAPF na ploše, což nám v práci postačilo k předvedení funkcionality našeho systému, nicméně implementace MAPF řešiče pro problémy v 3D by byla vhodná k dalšímu pokračování této práce.

Dále jsme zde představili dva pomocné moduly *plotter* a *eval*. Tyto moduly slouží k vizualizaci a vyhodnocování provedených letových sekvencí. Kapitolu jsme zakončili příkladem provedení navigační sekvence za pomoci námi vytvořeného balíčku.

Na závěr práce jsme provedli několik experimentů, ve kterých jsme mimo jiné zjistili, že vliv přítomnosti překážek na kvalitu lokalizace dronů je zanedbatelný. Tento fakt implikuje, že systém Loco Positioning je vhodný pro navigaci dronů i za přítomnosti překážek. Jelikož se námi vytvořený balíček opírá o lokalizaci za použití tohoto systému, je tedy použitelný pro navigaci dronů v prostředí s překážkami. Dále jsme v této kapitole prezentovali výsledky mapující kvalitu provedení letových sekvencí našeho navigačního balíčku.

Bibliografie

1. DICTIONARY, Merriam-Webster. *Drones Are Everywhere Now, But How Did They Get Their Name?* [Online]. [B.r.] [cit. 2022-11-19]. Dostupné z: `https://www.merriam-webster.com/words-at-play/how-did-drones-get-their-name`.
2. SILVER, David. Cooperative pathfinding. In: *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*. 2005, sv. 1, s. 117–122. Č. 1.
3. GAUTIER, Anna; STEPHENS, Alex; LACERDA, Bruno; HAWES, Nick; WOOLDRIDGE, Michael J. Negotiated Path Planning for Non-Cooperative Multi-Robot Systems. In: *AAMAS*. 2022, s. 472–480.
4. *Informované prohledávání stavového prostoru: Dijkstra, heuristiky, Greedy search, A** [online]. [B.r.] [cit. 2022-11-18]. Dostupné z: `https://courses.fit.cvut.cz/BI-ZUM/media/lectures/BI-ZUM_lecture-03_heuristic.pdf`.
5. DECHTER, Rina; PEARL, Judea. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*. 1985, roč. 32, č. 3, s. 505–536. ISSN 0004-5411. Dostupné z DOI: `10.1145/3828.3830`.
6. HART, Peter E.; NILSSON, Nils J.; RAPHAEL, Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107. Dostupné z DOI: `10.1109/TSSC.1968.300136`.
7. LI, Jiaoyang; SURYNEK, Pavel; FELNER, Ariel; MA, Hang; KUMAR, T. K. Satish; KOENIG, Sven. Multi-Agent Path Finding for Large Agents. In: *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*. 2019, s. 186–187. Dostupné také z: `https://aaai.org/ocs/index.php/SOCS/SOCS19/paper/view/18380`.
8. STERN, Roni; STURTEVANT, Nathan R; FELNER, Ariel; KOENIG, Sven; MA, Hang; WALKER, Thayne T; LI, Jiaoyang; ATZMON, Dor; COHEN, Liron; KUMAR, TK Satish et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In: *Twelfth Annual Symposium on Combinatorial Search*. 2019.
9. SURYNEK, Pavel. An optimization variant of multi-robot path planning is intractable. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2010, sv. 24, s. 1261–1263. Č. 1.
10. SHARON, Guni; STERN, Roni; FELNER, Ariel; STURTEVANT, Nathan R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. 2015, roč. 219, s. 40–66. ISSN 0004-3702. Dostupné z DOI: `https://doi.org/10.1016/j.artint.2014.11.006`.

11. SURYNEK, Pavel. Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 2019, s. 1177–1183. Dostupné z DOI: 10.24963/ijcai.2019/164.
12. ANDREYCHUK, Anton; YAKOVLEV, Konstantin S.; ATZMON, Dor; STERN, Roni. Multi-Agent Pathfinding (MAPF) with Continuous Time. *CoRR*. 2019, roč. abs/1901.05506. Dostupné z arXiv: 1901.05506.
13. SURYNEK, Pavel. Multi-agent Path Finding with Continuous Time Viewed Through Satisfiability Modulo Theories (SMT). *CoRR*. 2019, roč. abs/1903.09820. Dostupné z arXiv: 1903.09820.
14. SURYNEK, Pavel; FELNER, Ariel; STERN, Roni; BOYARSKI, Eli. Efficient SAT Approach to Multi-Agent Path Finding Under the Sum of Costs Objective. In: *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*. 2016, s. 810–818. Dostupné z DOI: 10.3233/978-1-61499-672-9-810.
15. BITCRAZE AB. *Loco Positioning System Tools* [online]. [B.r.]. Ver. 2018.10 [cit. 2022-04-19]. Dostupné z: https://www.bitcraze.io/documentation/repository/.
16. LANGLEY, Richard B et al. Dilution of precision. *GPS world*. 1999, roč. 10, č. 5.
17. SURYNEK, Pavel. Pushing the Envelope: From Discrete to Continuous Movements in Multi-Agent Path Finding via Lazy Encodings. *CoRR*. 2020, roč. abs/2004.13477. Dostupné z arXiv: 2004.13477.
18. HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*. 2007, roč. 9, č. 3, s. 90–95. Dostupné z DOI: 10.1109/MCSE.2007.55.

Obsah přiloženého média

```
README.md ..... Stručný popis repozitáře
commander.py ..... Modul pro navigaci dronů
plotter.py ..... Modul pro vizualizaci letů
eval.py ..... Modul pro vyhodnocování experimentů
cf_scripts/ ..... Složka obsahující vybrané skripty
src/ ..... Složka obsahující pomocné moduly
experiments/ ..... Složka obsahující logy experimentů
thesis.pdf ..... text práce ve formátu PDF
```