FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Lokalizace robotů při vykonávání plánů multi-agentního hledáního cest s OZOBOTy
Student:	Silvestr Láník
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout systém lokalizace robotů, kteří vykonávají plán multi-agetního hledání cest (MAPF), tj. roboti se současně bezkolizně pohybují do svých cílových pozic. Předpokládá se navázání na existující práce využívající pro testování vykonávání plánů MAPF malé mobilní roboty OZOBOT Evo v reflexním režimu. Jako vhodné hardwarové řešení lokalizace se nabízí využití webové kamery. Úkoly pro uchazeče jsou následující:

1. Prostudujte existující techniky pro multi-agentní hledání cest, související přístupy pro vykonávání plánů na skutečných robotech a metody počítačového vidění se zaměřením na sledování pohybujících se objektů.

2. Navrhněte lokalizaci robotů typu OZOBOT Evo v existujícím systému pro vykonávání plánů na těchto robotech s využitím webové kamery. Využijte přitom vhodných metod počítačového vidění.

3. Otestujte navržený lokalizační systém v relevantních scénářích s použitím skutečných robotů.

–

[1] David Silver: Cooperative Pathfinding. AIIDE 2005: 117-122

[2] Ján Chudý, Nestor Popov, Pavel Surynek: Emulating Centralized Control in Multi-Agent Pathfinding Using Decentralized Swarm of Reflex-Based Robots. SMC 2020: 3998-4005

[3] OpenCV. Open Source Computer Vision Library. https://opencv.org/, [leden 2021]

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 31. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Lokalizace robotů při vykonávání plánů multi-agentního hledáního cest s OZOBOTy

Silvestr Láník

Katedra aplikované matematiky
Vedoucí práce: doc. RNDr. Pavel Surynek, Ph.D.

13. května 2021

Poděkování

Děkuji především svému vedoucímu za trpělivost a cenné rady. Dále děkuji Aničce, že mi fandila a své rodině za veškerou podporu během studia.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Láník, Silvestr. *Lokalizace robotů při vykonávání plánů multi-agentního hledáního cest s OZOBOTy*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Ve své práci se zabývám vytvořením lokalizačního systému Ozobotů, během provádění simulace multi-agentního hledání cest. Hlavním důvodem vzniku této práce jsou občasné chyby při provádění simulace, způsobené špatnou detekcí promítané trasy ze strany Ozobota. Ke sledování a lokalizaci Ozobotů používám webovou kameru a známé poznatky z oblasti počítačového vidění. Vytvořený lokalizační systém dokáže v obraze detekovat oblast simulace a dále detekovat a sledovat Ozoboty, kteří se v ní pohybují a navracet jejich pozice v této simulaci. Dle provedených měření má lokalizační systém úspěšnost 97 % a v budoucnu se počítá s jeho využítím v komplexnějším dohledovém mechanismu, což povede k zpřesnění provádění simulace.

Klíčová slova OpenCV, MAPF, Ozobot, sledování, počítačové vidění, simulace

Abstract

In my work, I focus on creating a localization system of Ozobots while they performing a multi-agent pathfinding simulation. The main reason for this work is the occasional errors during the execution of the simulation, caused by poor detection of the projected path by the Ozobot. I use a webcam and known insights from computer vision to track and locate Ozobots. The developed localization system can detect the simulation area in the image and further detect and track Ozobots moving in the area and return their positions in the simulation. According to the measurements made, the localization system has a success rate of 97 % and in the future it is planned to use it in a more complex surveillance mechanism, which will lead to a more accurate execution of the simulation.

Keywords OpenCV, MAPF, Ozobot, tracking, Computer Vision, simulation

Obsah

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Úvod

Má bakalářská práce navazuje na existující simulaci multi-agentního hledání cest pomocí Ozobotů *ESO-OzoNav Prototype*, která vznikla v práci [1]

Motivací k vytvoření lokalizace Ozobotů v systému *ESO-OzoNav Prototype* jsou občasné chyby Ozobotů při následování vedoucích čar, které jsou zobrazovány na povrchu, po němž se Ozoboti během simulace pohybují. Tyto chyby následně vedou ke kolizi Ozobotů nebo k odjetí Ozobota mimo oblast, ve které je simulace prováděna. Realizací systému lokalizace bude umožněn budoucí vznik dohledového mechanismu, který bude využit k detekci a následné opravě chyb v provádění simulace.

Cíl práce

Hlavním cílem této práce je realizovat rozpoznání a lokalizaci mobilních robotů v obraze při provádění simulace multi-agentního hledání cest (MAPF). Rešeršní část práce analyzuje současné metody počítačového vidění sloužících k detekci a sledování pohybujících se objektů. Dále se věnuje analýze předchozí bakalářské práce, v níž vznikl systém simulace MAPF pomocí Ozobotů *(ESO-OzoNav Prototype)*. Za pomoci poznatků získaných v této části je pak možné přistoupit k implementaci lokalizačního systému.

Cílem implementační části práce je navrhnout a implementovat lokalizaci Ozobotů pomocí webové kamery v systému *ESO-OzoNav Prototype*. Prvním dílčím cílem je tedy detekovat samotnou oblast simulace v zachyceném obrazu z webové kamery. Druhým dílčím cílem této části je realizovat sledování Ozobotů ve videu. Po splnění těchto cílů je pak možné přistoupit k cíli poslednímu a to namapování pozice Ozobotů ve videu na pozici v prováděné simulaci. Ozoboti tak budou jednoznačně lokalizováni.

Kapitola 1

Východiska

Pokud na celý systém simulace budeme konceptuálně pohlížet, tak jak je znázorněno na obrázku 1.1, je cílem mé práce vytvořit základ jednoduchého dohledového systému. Ten bude v budoucnu předávat informace o svých pozorováních plánovači, který tak bude mít zpětnou vazbu, jak dobře jsou vykonávány plány, které agentům předal. Plánovač poté bude moci adekvátně reagovat například úpravou plánů.

V této kapitole se věnuji analýze částí systému, které vznikly v předchozích pracích a poté analýze metod, které by mohly být využity k vytvoření lokalizačního systému. Konkrétně tedy nejdříve popisuji problém MAPF a algoritmy určené k jeho řešení. Dále se zabývám analýzou simulátoru *ESO-OzoNav Prototype*, na který má práce navazuje. Protože velká část zadání mé práce je tvořena instancí obecného problému MOT, rozebírám také nejdůležitější poznatky z této oblasti. Pozornost věnuji rovněž rozboru již existujícího řešení sledování Ozobotů za pomoci webové kamery.

1.1 MAPF

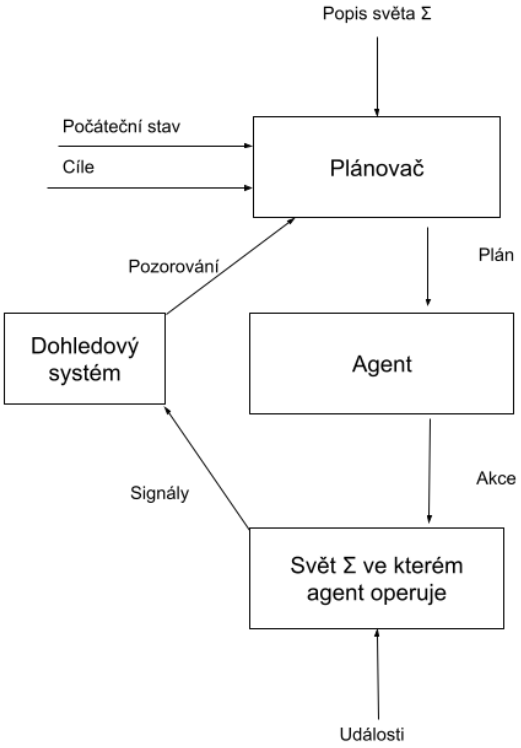
1.1.1 Definice

V [4] je MAPF definováno jako problém, při kterém jednotliví agenti musí najít nekolidující cesty do svých cílů, za předpokladu, že mají kompletní informace o cestách ostatních agentů.

Dále v [5] je MAPF definován následovně:

Předpokládáme mapu, která je reprezentována grafem $G = (V, E)$ a množinu robotů $R = \{r_1, ..., r_k\}$. O mapě dále předpokládáme, že:

- Mapa je zkonstruována tak, že roboti se střetnou pouze v případě, kdy se současně pokusí přemístit do stejného vrcholu mapy. Vrcholy tedy musí být dostatečně vzdálené, aby se roboti mohli nacházet na libovolných dvou různých vrcholech, bez toho aniž by se střetli.

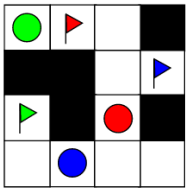


Obrázek 1.1: Konceptuální model. Inspirováno z [2] [3]

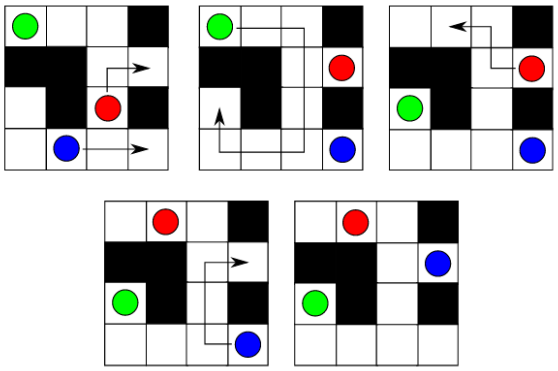
- Robot se může přemístit z vrcholu v_i do sousedního vrcholu v_j, pokud je vrchol v_j volný a žádný jiný robot do něj současně nevjíždí nebo jej neopouští. Roboti na jiných vrcholech mapy tuto akci nijak neovlivní.
- Startovní a cílové pozice všech robotů se nachází na mapě.

Dále máme dvě zobrazení $S_0, S^+ : R \to V$, kde $S[r_i] \neq S[r_j]$, pro všechna $i \neq j$, reprezentující startovní a cílové pozice jednotlivých robotů.

S pomocí těchto informací chceme získat množinu plánů $P = \{P_r | r \in R\}$, za pomoci kterých všichni roboti dorazí do svých cílů. Zároveň konstruujeme částečné uspořádání $\prec$ mezi jednotlivými kroky plánů, tak aby dva roboti nikdy současně neobsadili stejný vrchol mapy.



Obrázek 1.2: MAPF. Převzato z [6]



Obrázek 1.3: Možné řešení MAPF. Převzato z [6]

Efektivita *řešení MAPF* je nejčastěji posuzována na základě jedné z následujících funkcí:

- *Makespan*: Počet časových úseků nutných k tomu aby se všichni agenti dostali do svých cílových pozic Pro řešení problému MAPF $\pi = \{\pi_1, ..., \pi_k\}$ je *makespan* π definován jako $max_{1 \leq i \leq k} |\pi_i|$
- *Sum of costs (SOC)*: Suma všech časových úseků každého agenta, které jsou nutné k tomu, aby se všichni agenti dostali do svých cílových pozic. *Sum of costs* π je definována jako $\sum_{1 \leq i \leq k} |\pi_i|$

Ukázkové zadání problému MAPF je zobrazeno na obrázku 1.2, kde kruhy představují startovní pozice a vlajky pozice cílové. Možné řešení je pak vyobrazeno v 1.3

1.1.2 Algoritmy řešící MAPF

1.1.2.1 Konfliktní prohledávání

Popis algoritmu konfliktního prohledávání (dále CBS) jsem převzal z [7]. Velmi srozumitelné shrnutí bylo prezentováno také v práci [8].

Algoritmus 1 Konfliktní prohledávání (CBS)

1: $R.constraints \leftarrow \varnothing$
2: $R.solution \leftarrow$ cesty nalezené pomocí dolní vrstvy
3: $R.\mu \leftarrow$ cena($R.solution$)
4: $OPEN \leftarrow R$
5: **while** $OPEN \neq \varnothing$ **do**
6: $N \leftarrow$ uzel z $OPEN$ s nejnižší cenou
7: **if** $N.solution$ nemá konflikt **then**
8: **return** $N.solution$
9: $C \leftarrow$ první konflikt(a_i, a_j, v, t) z $N.solution$
10: **for** $a \in \{a_i, a_j\}$ **do**
11: $U \leftarrow$ nový uzel
12: $U.solution \leftarrow N.solution \cup \{(a, v, t)\}$
13: $U.constraints \leftarrow N.constraints$
14: $U.solution(a) \leftarrow$ cesta pro a
15: $U.\mu \leftarrow$ cena($U.solution$)
16: $OPEN \leftarrow OPEN \cup Q$

CBS je rozdělen na horní a dolní vrstvu. Dolní vrstva hledá nejkratší cestu pro jednotlivé agenty se zadanými omezeními. V horní vrstvě algoritmus prohledává *constraint tree* (CT) a řeší případné konflikty. Pseudokód horní vrstvy je popsán v 1

CT je binární strom, kde každý uzel N stromu obsahuje:

- množinu omezení pro jednotlivé agenty (N.constraint)
- řešení (N.solution)
- cenu řešení (N.μ)

Omezení (*constraints*), která se v uzlech stromu nachází jsou tvaru (a_i, v, t) a značí, že agent a_i nemůže v čase t navštívit vrchol v.

Uzel N označíme za cílový, pokud množina jeho řešení N.solution neobsahuje žádný konflikt.

Kořen stromu je na počátku inicializován množinou nejkratších cest pro každého agenta. Množina omezení je v kořenu stromu prázdná a cena řešení je rovna součtu cen jednotlivých nejkratších cest. Následně je každý uzel zpracováván prioritně dle ceny řešení. Pokud je zpracovávaný uzel označen jako cílový je navrácena jeho množina řešení a algoritmus je zastaven. Pokud uzel není označen za cílový, tak obsahuje konflikt. Konflikt C je definován jako čtveřice (a_i, a_j, v, t), kde a_i, a_j jsou kolidující agenti, v je vrchol, ve kterém ke kolizi došlo a t je časový úsek konfliktu. Konfliktnímu uzlu jsou přidáni dva následovníci, jeden za každou konfliktní cestu. Do každého z těchto nově

vzniklých uzlů je přidáno nové omezení na konfliktní vrchol a konfliktní cesta je přepočítána.

Algoritmus dokáže pracovat také s hranovým konfliktem, který je definován jako pětice (a_i, a_j, v_1, v_2, t), což značí, že se agenti snaží prohodit si pozice. Tj. mezi časy t a $t+1$ se agent a_i snaží dostat z vrcholu v_1 do vrcholu v_2, zatímco agent a_j se snaží dostat z vrcholu v_2 do vrcholu v_1.

1.1.2.2 Převedení na problém splnitelnosti

Přístup tohoto řešení zpočívá v převedení problému MAPF na problém splnitelnosti výrokové formule. Formule je splnitelná pouze v případě, že řešení MAPF problému existuje. Na takto převedený problém může být použit některý ze SAT řešičů. Pokud řešení existuje a je nalezeno může být následně zpětně převedeno. Hlavním úkolem tohoto algoritmu je tedy vhodné zakódování problému do výrokové formule.

1.2 Analýza simulátoru

Ve své práci [1] Ján Chudý vytvořil MAPF simulátor s využitím Ozobotů a unikátní metody jejich navigace, kterou bylo nutné použít k vypořádání se s některými nedostatky Ozobotů (nemožnost paralelního programování, neexistence textově orientovaného programovacího jazyka pro Ozoboty aj.)

V práci vzniklo několik verzí simulátoru *ESO-OzoNav Prototype*. Verze na sebe logicky navazovaly a snažily se řešit problémy verzí předchozích. V následující analýze se zaměřuji především na finální verzi, která je pro mou práci nejpodstatnější. V místech, kde je to dle mého názoru nezbytné pro pochopení, prezentuji i varianty představené v předchozích verzích. Tato skutečnost je vždy výslovně zmíněna.

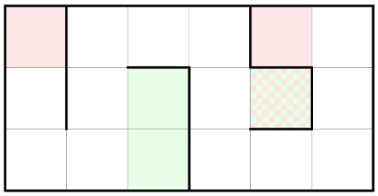
1.2.1 Zobrazené prostředí

Simulace byla vytvářena pro grafy, které lze zobrazit jako mřížku. Graf lze pak jednoduše převést z abstraktní grafové reprezentace do mřížky zobrazitelné na monitoru či televizi.

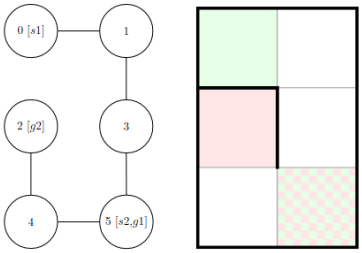
Ukázková mřížka je vyobrazena na obrázku 1.4.

Celá oblast simulace je vždy obehnána stěnami, které vymezují perimetr prováděné simulace.

Každý vrchol abstraktní grafové reprezentace je v simulaci zobrazen jako jedna dlaždice. Pokud mezi dvěma sousedními vrcholy neexistuje v grafu hrana, je v zobrazené simulaci mezi příslušnými dlaždicemi nakreslena stěna. V případě, že mezi vrcholy hrana existuje, stěna nakreslena není. Je ale nutné od sebe jednotlivé dlaždice i přes to nějak vizuálně odlišit, a proto je mezi nimi zobrazena hraniční čára. Tloušťka hraniční čáry je nastavena na naprosté minimum (1 pixel) aby ji Ozobot nedetekoval jako část promítané trasy.



Obrázek 1.4: Ukázková mřížka. Převzato z [1]



Obrázek 1.5: Vztah mezi grafem a promítanou mapou. Převzato z [1]

Všechny startovní dlaždice jsou označeny zelenou barvou a navíc doplněny o malou šipku, určující směr, jímž má směřovat čelo Ozobota při startu simulace. Všechny cílové dlaždice jsou označeny barvou červenou. V případě, že dlaždice je jak startovní tak cílová, je tato skutečnost reprezentována zeleno-červenou šachovnicí zobrazenou na této dlaždici.

Zde je nutné zmínit, že barvy startovních/cílových dlaždic musí mít vysokou průhlednost, aby nebyly Ozoboty chybně detekovány jako čáry určené k následování.

Ukázkové převedení grafu na mapu simulace je zobrazeno na obrázku 1.5.

1.2.2 Použité zobrazovací zařízení

Simulace může být promítána na libovolném displeji. Velikost displeje ale samozřejmě omezuje maximální možnou velikost promítané mapy. Jelikož se navíc displeje mohou lišit v rozlišení a rozměrech je potřeba tyto informace simulaci poskytnout. Simulaci jsou tyto informace dodány v konfiguračním souboru s rozměry a rozlišením použitého displeje. Při záměně monitorů je potřeba na tuto skutečnost myslet a nezapomenout konfigurační soubor upravit, v opačném případě může dojít k nesprávnému škálování simulace.

Za pomoci konfiguračního souboru pak simulace zajistí požadované reálné rozměry částí simulace (např. tloušťka stěn v milimetrech). Rozměry se samozřejmě mohou od požadovaných mírně lišit, jelikož jsou zaokrouhlovány na celé pixely. Přesnost tedy v tomto případě závisí především na jemnosti displeje.

```
1  V =
2  (0,1,0)
3  (1,0,0)
4  (2,0,2)
5  (3,0,0)
6  (4,0,0)
7  (5,2,1)
8  E =
9  {0,1}
10 {1,3}
11 {2,4}
12 {3,5}
13 {4,5}
```

Obrázek 1.6: Textová reprezentace mapy. Převzato z [1]

1.2.3 Mapy

Mapy jednotlivých simulací jsou uchovávány jako textové soubory obsahující seznam vrcholů a hran.

Vrchol je reprezentován jako trojice *(v, s, g)*, kde v je číslo vrcholu, s a g pak číslo agenta, který má v daném vrcholu svou startovní (resp. cílovou) pozici. Číslo 0 představuje nepřítomnost startovní/cílové pozice agenta v daném vrcholu.

Hrana je vždy reprezentována jako dvojice $\{v_i, v_j\}$, kde v_i, v_j představují čísla vrcholů.

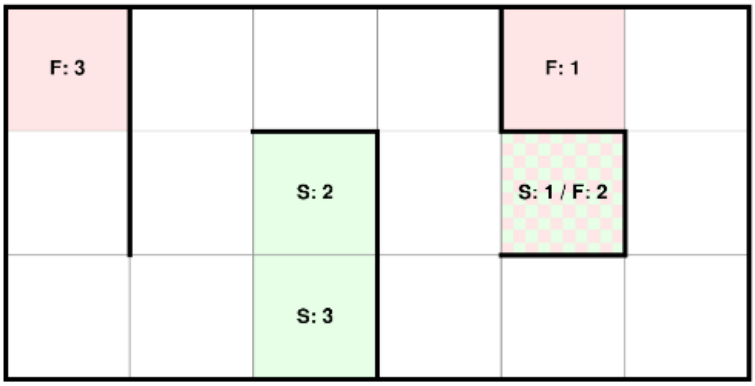
Na obrázcích 1.5 a 1.6 je vidět ilustrativní případ převzatý z původní práce.

1.2.4 Editor map

Jelikož tvorba map jen pomocí psaní textové reprezentace do souboru by byla značně nepohodlná, obsahuje práce kromě samotného simulátoru i rozhraní určené k tvorbě nových map.

Použití je velmi přímočaré. Před spuštěním samotného editoru je uživatel vyzván k zadání *výšky* a *šířky* nově vytvářené mapy a *počtu agentů*, kteří by se měli na mapě nacházet. Po nastartování editoru je zobrazena ohraničená plocha o velikosti zadaných rozměrů a uživatel může přepínat mezi třemi módy stiskem uvedené klávesy

- umisťování stěn (klávesa **w**)
- umisťování startovních pozic (klávesa **s**)



Obrázek 1.7: Prostředí editoru. Převzato z [1]

- umisťování cílových pozic (klávesa **f**)

Startovní a cílové pozice jsou umisťovány ve vzestupném pořadí.

Jakmile jsou umístěny všechny startovní a cílové pozice je možné editor ukončit stiskem klávesy **Enter** a následně už jen zvolit jméno pro nově vytvořený soubor s mapou nebo potvrdit programem navrženou možnost.

Editor je kdykoli možné ukončit stiskem klávesy **Esc**.

Ukázkové prostředí editoru je zobrazeno na obrázku 1.7 převzatém z původní práce.

1.2.5 MAPF solver

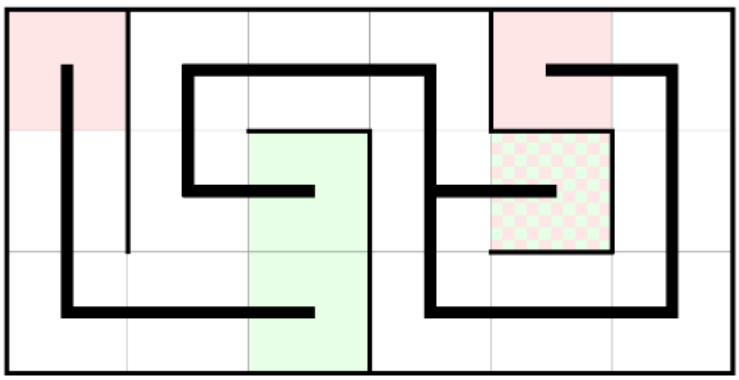
Simulace využívá již existujícího programu **boOX** [9], který implementuje několik MAPF algoritmů. Simulace konkrétně používá algoritmus **SMT-CBS** [10]. Mapa konkrétní simulace, ve formátu popsaném v sekci Mapy, je předána programu **boOX**, který nalezne řešení, jež má být následně simulováno.

Abstraktní třída *SubprocessSolver* však poskytuje rozhraní, které může být jednoduše implementováno jinými programy, řešícími MAPF problém.

1.2.6 Promítání trasy

Po umístění správně orientovaných Ozobotů na jejich startovní pozice, je uživatelem na displeji spuštěno promítání trasy, která by každého Ozobota měla dopravit do jeho cílové pozice.

Promítání trasy prošlo mezi jednotlivými verzemi největšími úpravami, proto bych zde rád zmínil i některé předchozí verze a jejich nedostatky, které řeší verze finální.



Obrázek 1.8: Promítnutí celého řešení najednou. Převzato z [1]

1.2.6.1 ESO-OzoNav 0

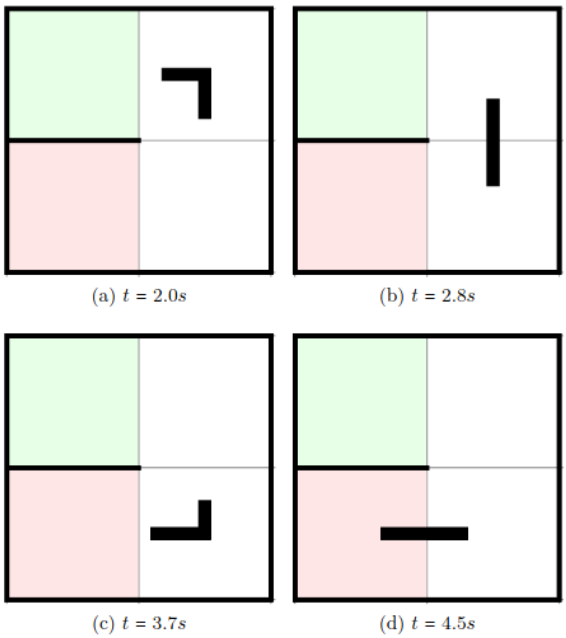
Verze, která slouží jen jako demonstrace, že nalezené řešení nemůže být zobrazeno najednou, je zachycena na obrázku 1.8. Ozoboti by v takovém případě nevěděli, kde počkat a nechat projet ostatní, protože se zastaví pouze na konci promítané čáry. Tato skutečnost činí tuto verzi nepoužitelnou pro simulaci MAPF, kde je vyžadováno aby každý agent mohl provést akci *čekání*. Zároveň díky tomu, že na každé křižovatce si Ozobot vybírá další směr náhodně, tak by chování simulace bylo nedeterministické a nebylo by možné zaručit její správný chod.

1.2.6.2 ESO-OzoNav 1

Problémy ESO-OzoNav 0 byly vyřešeny postupným zobrazováním částí promítané trasy (dále *animace trasy*). Animace trasy je zobrazována pod Ozobotem v průběhu jeho pohybu po mapě (obrázek 1.9). Je tak zamezeno jak problému s křižovatkami, tak problému s neexistující akcí *čekání*.

Objevili se však další problémy, které bylo potřeba vyřešit:

- Ozoboti nezastaví přesně na konci čáry, ale mírně za ním
- Zastavení na dlaždici s promítanou zatáčkou vede ke ztrátě trasy, kvůli nedokonalému zastavení popsanému v předchozím bodě
- Neexistuje možnost Ozobota na místě otočit o 180°
- Trasa se zatáčkami je následována delší dobu než rovná trasa, kvůli nutným otočením Ozobota



Obrázek 1.9: Prvotní implementace *animace trasy*. Převzato z [1]

1.2.6.3 ESO-OzoNav 2

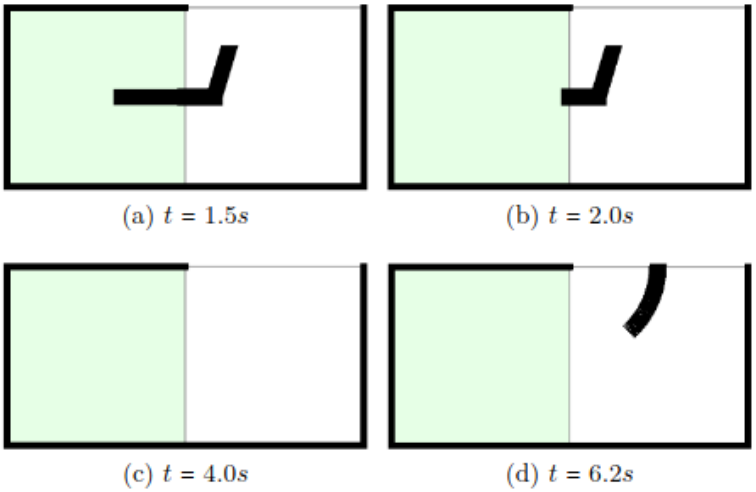
V této verzi byly učiněny následující změny:

- Problém, kdy Ozobot nezastaví přesně na konci čáry, je vyřešen dřívějším promítnutím konce čáry.
- Ozobotům je umožněno se na dlaždici otočit o 180° za pomoci promítání barevných kódů, které byly získány revezním inženýrstvím.
- Zatáčka je v této verzi zobrazována jako křivka nikoli ostrý zlom.
- Ztráta trasy při zastavení na dlaždici se zatáčkou je vyřešena pomocí promítnutí speciálně upravené části trasy, viz obrázek 1.10, která Ozobota zastaví na správném místě a se správnou orientací k pokračování v následování zatáčky.

Problém s desynchronizací není v této verzi uspokojivě vyřešen.

1.2.6.4 ESO-OzoNav 3 a finální verze

V těchto verzích je řešen problém desynchronizace pomocí barevných kódů a úpravy chování jednotlivých Ozobotů.



Obrázek 1.10: Speciální část trasy, určená k zastavení v zatáčce. Převzato z [1]

V ideálním případě by měl být Ozobot vždy uprostřed promítané části čáry, kterou má následovat. Ozoboti v tomto ale selhávají kvůli desynchronizaci. Je tedy nutné upravit jejich rychlost aby v případě, kdy se nachází na konci animované části zrychlili a animovaná část jim neujela. A naopak v případě, že se nachází na začátku animované části zpomalili, aby animovanou část nepředjeli.

Tato úprava rychlosti je realizována úpravou barvy promítané části trasy, kdy na konci je zobrazována barva červená, která značí příkaz ke zrychlení a na začátku barva modrá, která označuje příkaz ke zpomalení. Uprostřed je pak nadále zobrazována černá barva, tedy ideální rychlost.

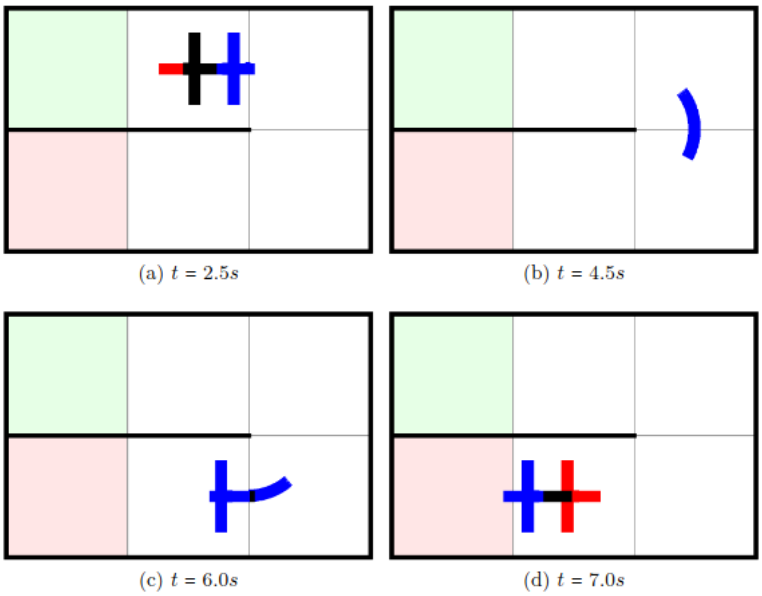
Jelikož Ozoboti nepodporují paralelní zpracování příkazů není možné jim zároveň dát instrukci k pohybu a k načítání barvy trasy. Je tedy nutné na trase navíc zobrazovat křižovatky a instruovat Ozoboty aby dojeli k další křižovatce, načetli promítanou barvu, upravili podle ní svoji rychlost a pokračovali k další křižovatce. Animace trasy ve finální verzi je zachycena na obrázku 1.11

Vyžadovaná úprava chování Ozobotů je detailně vysvětlena v následující kapitole.

1.2.7 Chování Ozobotů

Ke správnému chodu simulace je rovněž nutno upravit defaultní chování jednotlivých Ozobotů. Je požadováno aby Ozoboti následovali animovanou trasu, ale také aby dokázali správně přizpůsobit svoji rychlost dle promítané barvy trasy.

Za tímto účelem je vytvořen algoritmus, zajišťující požadované chování Ozobotů 2. Algoritmus lze jednoduše převést na Ozobot program v editoru



Obrázek 1.11: Ukázka navigace ve finální verzi. Převzato z [1]

Ozoblockly a poté jej hromadně nahrát do všech Ozobotů.

Jak je z přiloženého kódu vidět Ozoboti poté v nekonečném cyklu vždy volí směr rovně a následují promítanou čáru až do další křižovatky či konce čáry a mezi jednotlivými křižovatkami (resp. konci čar) upravují svoji rychlost za pomocí algoritmu 3.

Cyklus nemá žádnou ukončovací podmínku a lze jej tedy ukončit pouze manuálně.

Algoritmus 2 Chování Ozobotů [1]

```
1: while True do
2:     nastav rychlost na rychlostDleBarvy() mm/s
3:     if existuje cesta rovně then
4:         vyber směr: rovně
5:     else
6:         zastav motory
7:     následuj čáru k další křižovatce nebo konci čáry
```

1.2.8 Příčiny selhání prováděné simulace

Chyby, které v simulaci vznikají jsou v práci rozděleny následovně:

- *Missed intersection* (MI) K této chybě dochází pokud senzory Ozobota nezaregistrují zobrazenou křizovatku. Ozobot tedy neupraví svou rych-

Algoritmus 3 Funkce upravující rychlost [1]

```
1:  function rychlostDleBarvy()
2:      barva ← načti barvu povrchu
3:      if barva = červená then
4:          rychlost ← 37
5:      else if barva = černá then
6:          rychlost ← 30
7:      else if barva = černá then
8:          rychlost ← 23
9:      else
10:         rychlost ← 21
        return rychlost
```

lost a následně vjede do zatáčky příliš rychle, což vede k tomu, že není schopen následovat promítanou zatáčku a vypadne ze své trasy.

- *Severe collision* (SC) Jedná se o ojediněle se vyskytující problém, kdy dva Ozoboti najedou proti sobě a následně se přetlačují. Oběma Ozobotům tedy ujede jejich promítaná trasa a již nejsou schopni se z této chyby vzpamatovat.

- *Color Code* (CC) Zde se jedná o chybu přečtení obrazového kódu, který se zobrazuje na konkrétní dlaždici v případě, že ji má Ozobot opustit stejným směrem, kterým na ni dorazil (tj. otočit se o 180°). Ozobot buď kód nepřečte vůbec nebo ho přečte dvakrát. V každém případě skončí čelem v nesprávném směru a není schopen pokračovat v simulaci.

- *Wait on a turn* (WoT) Ozobot je v tomto případě zastaven během následování zatáčky, jelikož musí na dané dlaždici počkat, než projedou ostatní Ozoboti. Stává se, že se Ozobot při zastavení otočí o 180° a není tedy poté schopen pokračovat po následně promítané části trasy.

- *Exit a curve* (EC) Zřídka se vyskytující problém, kdy Ozobot i přes správné přizpůsobení rychlosti ztratí promítanou zatáčku a vypadne tak ze své trasy.

Kořenovou příčinou všech chyb, které při simulaci vznikají, je tedy špatná detekce promítané trasy ze strany Ozobota, a to často buď kvůli nepříznivým světelným podmínkám nebo špatné kalibraci senzoru.

1.3 Segmentace obrazu a lokalizační systém

Segmentace obrazu rozděluje původní obraz na oblasti se společnými vlastnostmi nebo oblasti oddělené nějakou hranicí. Cílem segmentace je především

Obrázek 1.12: Adaptivní prahování. Převzato z [12]

odfiltrovat objekty, které nás zajímají, od pozadí, které nás ve velké většině případů nezajímá. Takto odfiltrované objekty lze následně zpracovat dalšími metodami počítačového vidění. Dále vysvětluji základní techniky segmentace obrazu, které by bylo možné využít k realizaci lokalizačního systému. Všechny dále zmíněné techniky jsou implentovány v knihovně OpenCV [11]

1.3.1 Prahování

Jedná se o techniku, která staví na předpokladu, že objekty v popředí mají jinou úroveň jasu než pozadí. Jednotlivé pixely jsou pak na základě své úrovně jasu a zvoleného prahu T rozděleny na dvě skupiny dle vztahu:

$$g(i,j) = \begin{cases} 1 & f(i,j) \geq T, \\ 0 & f(i,j) < T \end{cases} \tag{1.1}$$

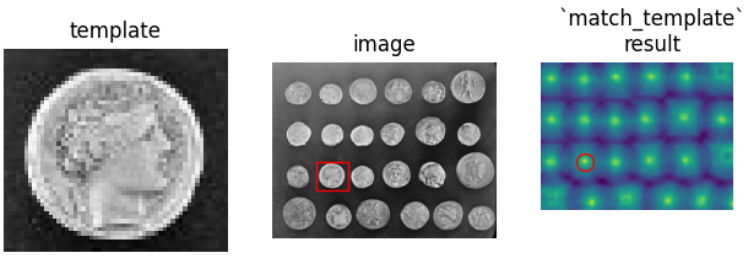
Metoda je velmi nenáročná jak na pochopení tak na výpočet. Problémem je hlavně správné určení prahu, podle kterého by mělo dělení probíhat. Navíc pouhý jeden práh neposkytuje dobré výsledky v případech, kdy je zachycený obraz nerovnoměrně nasvícen. V takovém případě je vhodné použít *adaptivní prahování*.

Adaptivní prahování rozdělí zachycený obraz na menší části a pro každou z těchto částí určí vhodnou úroveň prahu, která je následně použita k segmentaci v této části.

Vliv adaptivního prahování je zachycen na obrázku 1.12. Jako práh byl v tomto případě použit průměr jasu pixelů z posuzované části.

1.3.2 Srovnání se vzorem

Tento přístup hledá objekt, jehož vzor je mu poskytnut, v zadaném obraze. Kvůli své jednoduchosti je velmi citlivý na jakékoliv transformace hledaného



Obrázek 1.13: Srovnání se vzorem. Převzato z [13]

vzoru v zadaném obraze. Pokud se například vzor vyskytuje v zadaném obraze příliš natočený, zvětšený, zmenšený, zkosený atd., není již tato metoda schopna pomocí vzoru objekt nalézt.

Aby byla schopna detekce i v takovýchto podmínkách musela by brát v úvahu všechna natočení, zvětšení a další transformace vzoru, což by ji činilo nepoužitelnou kvůli výpočetní náročnosti.

Díky šumu, který je v každém obrazu přítomný nehledáme v obraze dokonalou kopii vzoru, ale pro jednotlivé body měříme podobnost nejčastěji pomocí vztahu

$$C(u,v) = \frac{1}{\sum_{(i,j)\in V}(f(i+u,j+v) - h(i,j))^2} \tag{1.2}$$

Vztah udává vzájemnou podobnost obrazu f a vzoru h umístěného/přiloženého na pozici (u, v). Podobnost spočteme pro každou pozici vzoru h v obraze f a následně určíme maximum, které představuje pozici hledaného vzoru.

Na obrázku 1.13 je znázorněno použití této metody pro nalezení zadané mince v obraze. I přesto, že kandidátů na vhodnou minci je více, maximum se nachází právě uprostřed hledané mince.

1.3.3 Dělení a spojování oblastí

Metoda rozdělování a spojování (split and merge) obraz posuzuje na základě kritéria homogenity. To může být definováno na základě mnoha ukazatelů hodnot jednotlivých oblastí (průměr hodnot pixelů, rozptyl, atd.). Obraz je postupně dělen na stále menší části, dokud všechny části nesplňují podmínku homogenity (často je děleno na *čtyři* části). Jakmile skončí fáze rozdělování, jsou naopak zpět spojovány sousední homogení části. Ukázkové rozdělení je vyobrazeno na obrázku 1.14. Podmínka homogenity může být definována ji-

nak pro fázi rozdělování a jinak pro fázi spojování. Pseudokód algoritmu je zachycen v 4.

Algoritmus 4 Rozdělení a spojování (Split and Merge)

```
1: R ← vstupní obrázek
2: TO_SPLIT ← R
3: TO_MERGE ← ∅
4: RESULT ← ∅
5: while TO_SPLIT ≠ ∅ do
6:     O ← TO_SPLIT.POP()
7:     if O nesplňuje kritérium homogenity then
8:         rozděl O na čtyři stejné části R1, R2, R3, R4
9:         TO_SPLIT ← O ∪ {R1, R2, R3, R4}
10:    else
11:        TO_MERGE ← TO_MERGE ∪ {O}
12: while TO_MERGE ≠ ∅ do
13:    P ← TO_MERGE.POP()
14:    for S ∈ sousedi(P) do
15:        if S ∪ P splňuje kritérium homogenity then
16:            TO_MERGE.POP(S)
17:            TO_MERGE ← TO_MERGE ∪ {S ∪ P}
18:            continue
19:    RESULT ← RESULT ∪ {P}
20: return RESULT
```

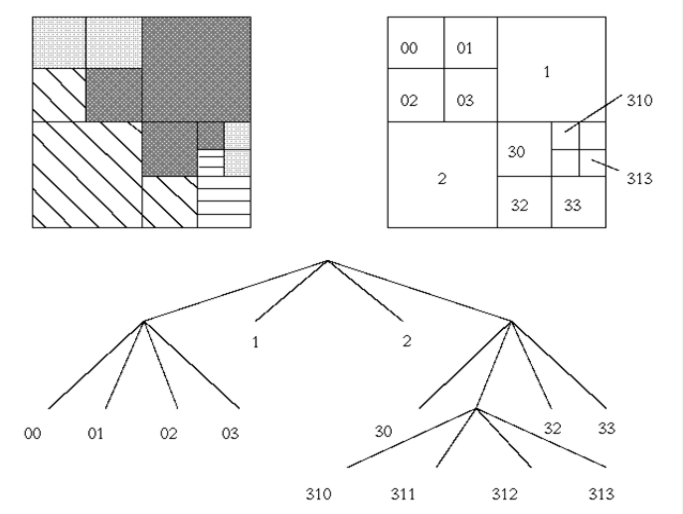
1.4 Sledování více objektů

Při psaní této kapitoly jsem vycházel primárně z [15] [16].

Problém sledování více objektů ve videu (dále MOT) můžeme definovat následovně: „Pro každý snímek ve videu lokalizuj a identifikuj všechny objekty zájmu, tak aby identity objektů byly konzistentní napříč celým videem.“ [17]

MOT získává v dnešní době na důležitosti, a to jak ze strany průmyslu, tak v akademických kruzích. Většina prací na toto téma se však zaměřuje především na sledování osob (ať již chodců [18] nebo sportovců [19]), automobilů [20] a zvířat [21].

Minimálně 70 % prací, které shromáždili v [15], se věnovalo právě sledování chodců. Zaměření na chodce je logické z pohledu využití v průmyslu, ať se jedná o využití při autonomním řízení aut nebo sledování osob při pohybu po nákupních centrech.



Obrázek 1.14: Dělení a spojování oblastí. Převzato z [14]

1.4.1 Metody inicializace

MOT lze rozdělit na dvě kategorie, dle toho jak je proces sledování zpočátku inicializován a to na:

- *sledování založené na detekci* dále DBT
- *sledování bez detekce* dále DFT

1.4.1.1 DBT

Při této metodě jsou v prvním zachyceném snímku všechny objekty, které mají být sledovány, automaticky označeny za použití konkrétního detektoru. Objekty jsou následně ve videu sledovány. Výhoda použití detektoru spočívá v tom, že může být v pravidelných intervalech spoušťen znovu a detekovat tak objekty, které se na počátečním snímku nenacházely. Detektor je samozřejmě možné takto použít i u každého zachyceného snímku.

Na druhou stranu u použití této metody je překážkou právě samotný detektor, jelikož musí být natrénovaný předem a musí být v detekci co nejspolehlivější. Pro sledování objektů, které nejsou v popředí výzkumného zájmu (obličeje, osoby, auta) tyto detektory většinou neexistují a je nutné natrénovat si vlastní, což vyžaduje spoustu výpočetního výkonu a co největší databázi snímků detekovaného objektu pořízených za různých podmínek.

1.4.1.2 DFT

Narozdíl od přechozí metody je fixní počet objektů, které mají být ve videu sledovány, vybrán v prvním zachyceném snímku manuálně. Sledování tedy

nezávísí na žádném externím detektoru. U použití pro větší počet objektů však může být tato metoda zdlouhavá.

1.4.2 Metody zpracování

MOT dle použitého zpracování dělíme na:

- *Online*: tato metoda má přístup pouze k aktuálnímu snímku a všem snímkům předchozím a musí pro každý zachycený snímek poskytnout výstup
- *Offline*: naopak této metodě je poskytnuto celé video a má tím pádem u každého snímku informace jak z předchozích tak z následujících snímků

1.4.3 Komponenty MOT

1.4.3.1 Model pohybu

Tato komponenta modeluje předpokládaný pohyb sledovaných objektů a tím redukuje prostor, který je nutné ve snímku prohledat. Od objektů se očekává, že se většinu času budou pohybovat souvisle až na ojedinělé náhlé změny.

1.4.3.2 Model vzhledu

Pokud se MOT nespoléhá na předem natrénovaný detektor, který používá k detekci v každém snímku, tak si buduje vlastní jednoduchou reprezentaci sledovaných objektů na základě jejich vzhledu. Bere v úvahu např. velikost a barvu sledovaného objektu. Model objektu bude proti předem natrénovanému detektoru vždy podstatně jednodušší a nepřesnější, protože máme k dispozici pouze snímky zachycené z videa a nikoli obrovské databáze snímků, které jsou používány k trénování detektorů.

1.4.3.3 Model interakce

Tento model bere v úvahu i další objekty nacházející se ve videu a jejich vzájemný vliv. Bývá využíván například u sledování skupin a davů lidí či zvířat. Často zde předpokládáme, že se jednotlivé objekty snaží nesrazit, nebo že se naopak přitahují a této skutečnosti využíváme k predikci příští pozice jednotlivých objektů.

1.5 Sledování Ozobotů v jiných pracích

Podařilo se mi dohledat jen jedinou práci, která se zabývá přímo sledováním Ozobotů. V následující části stručně popisuji metody použité v této práci, jejich vhodnost ke sledování Ozobotů v obraze a případné nedostatky, na které jsem v práci narazil.

1.5.1 Analýza existujícího řešení sledování Ozobotů

V práci [22] se autor věnuje detekci a sledování Ozobotů v čase pomocí webkamery v grafu, který je reprezentován mřížkou natištěnou na papíře. K detekci Ozobotů v obraze je v této práci použit detektor založený na kaskádových klasifikátorech (*Cascade Classifier*).

Jelikož natrénovaná kaskáda pro detekci Ozobotů není volně dostupná rozhodl se autor vytvořit si vlastní.

K natrénování vlastní kaskády je nutné získat relativně velké množství dat. Autor uvádí 550 snímků obsahujících Ozoboty a 1400 snímků, jenž Ozoboty neobsahují.

Uvedených 550 snímků pak bylo využito k automatickému vygenerování většího množství pozitivních snímků (1400) za pomoci různých transformací pozadí.

Získání negativních snímků pro autora znamenalo pořídit 1400 fotografií různých pozadí za různých podmínek. Pořízení těchto snímků je relativně nenáročné.

Získání dostatečného počtu snímků obsahujících Ozoboty považuji za největší slabinu zvolené metody. Stejně jako v předchozím případě není získání tohoto množství snímků za odlišných podmínek (osvětlení, úhel, pozadí, atd.) veskrze žádný problém. Avšak pro trénování je nutné každý jednotlivý snímek doplnit o informace, kolik objektů se na konkrétním snímku nachází a kde přesně se na snímků nachází (souřadnice jejich ohraničujících obdélníků). V této chvíli samozřejmě ještě není možné Ozoboty automaticky detekovat a doplnění těchto informací tedy představuje spoustu manuální a monotónní práce.

Požadovaný počet snímků lze, i s potřebnými informacemi, automaticky vygenerovat s dodáním mnohem menšího množství pozitivních snímků. Avšak tato metoda vede k výrazně horším výsledkům a autor ji proto zavrhl.

Další nevýhodou zvolené metody je doba potřebná k natrénování kaskády. Jako finální řešení byla zamýšlena natrénovaná kaskáda o dvaceti vrstvách. Výstupem je však kaskáda o šestnácti vrstvách, protože došlo k přerušení trénování z důvodu vysoké časové náročnosti „Při jedné fázi trénování totiž dochází k průchodu všech negativních obrázků a zkoušení, zdali na nich dosavadní kaskáda pozná Ozobota, když tam žádný není. Tato fáze při mém trénování trvala v konečných fázích neúnosně dlouho, u jedné vrstvy kaskády to bylo přes 24 hodin“ [22]

Autor zároveň nepoužívá žádnou z funkcí knihovny OpenCV [11] určenou přímo ke sledování objektů, jenž jsou pro použití k dispozici, ale sledování Ozobotů řeší vlastní implementací.

Ta spočívá v detekci Ozobotů v každém jednotlivém snímku a následném napárování pozic detekovaných Ozobotů na pozice z předchozího snímku. Jedná se tedy o jednoduchý případ metody DBT představené v předchozí kapitole. Zvolená implementace se ukazuje jako dostačující, jelikož Ozoboti mezi jednotlivými snímky neurazí velkou vzdálenost.

Existuje však autorem zdokumentovaná chyba při párování pozic, ke které občas dochází v případě, kdy jsou dva Ozoboti velmi blízko sebe. V takové situaci může dojít k záměně těchto dvou Ozobotů a tedy vnesení chyby do sledování.

Zde se ukazuje slabina zvoleného algoritmu, kdy jediné informace, které má algoritmus u všech snímků (2. počínaje) jsou pouze aktuální pozice Ozobotů a pozice Ozobotů v předchozím snímku.

Chybě by se šlo dle mého názoru vyhnout právě použitím jedné z tříd OpenCV [11] určených přímo ke sledování objektů. K tomuto názoru mě vede všeobecný princip sledovacích tříd, které narozdíl od tříd určených k detekci uchovávají mezi jednotlivými snímky dodatečné informace o sledovaném objektu (viz předchozí kapitola).

I přes časovou náročnost nutnou k realizaci a mírné nedostatky však výsledná implementace poskytuje uspokojivé výsledky při sledování Ozobotů a jejich lokalizaci v grafu.

Kapitola 2

Implementace lokalizace

Jak jsem zmínil již dříve, k úspěšné lokalizaci Ozobotů v prováděné simulaci je nutné splnit následující body:

- detekovat promítanou simulaci v obrazu získaném z webkamery
- detekovat všechny Ozoboty a nepřerušeně je sledovat v průběhu simulace
- namapovávat pozici Ozobotů v obraze na pozici v simulaci

Následující text se věnuje popisu toho, jak jsem přistoupil k implementaci těchto dílčích cílů.

2.1 Omezení problému

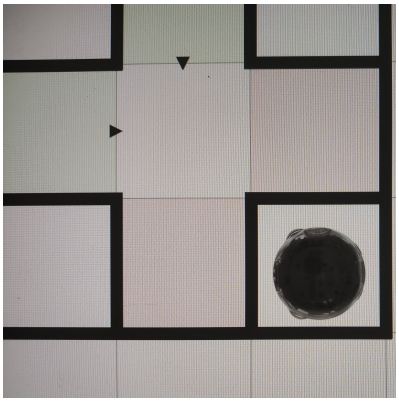
Pro usnadnění návrhu a implementace řešení je problém omezen následovně:

- Pozice kamery i promítané simulace je po celou dobu sledování fixní
- Monitor, na kterém je promítána simulace, zabírá co největší část zachyceného obrazu z webkamery
- Kamera se nachází co nejvíce kolmo nad promítanou simulací
- Pozice kamery je co nejvíce rovnoběžná s pozicí monitoru

2.2 Úprava simulátoru

V původním programu *ESO-OzoNav Prototype* jsem provedl následující změny:

- Upravení tloušťky promítaných stěn
- Zapsání základních informací o mapě do souboru.



Obrázek 2.1: Dostatek místa po rozšíření stěn

- Úprava konfiguračního souboru

Změny a důvody, které mě k těmto změnám vedly jsou dále rozepsány.

2.2.1 Upravení tloušťky promítaných stěn

K této změně jsem přistoupil především z důvodu usnadnění detekce ohraničujících stěn oblasti simulace. Tloušťka stěn byla zvětšena o 150 %.

Na prováděnou simulaci tato změna nemá žádný negativní vliv. Ozobotům nadále zbývá dostatek místa na manévrování 2.1, a protože stěny vymezují prostor, ve kterém by se Ozoboti měli pohybovat, tak tato úprava nevnáší žádné chyby do stávající simulace.

Napadlo mě zde také rozšířit i tloušťku hraničních čar jednotlivých dlaždic a jednoduše tak detekovat všechny dlaždice. Toto ale není možné. Tloušťka hraničních čar dlaždic je schválně nastavena na naprosté minimum (1 pixel).

Její úpravou bych se vystavil riziku, že by Ozoboti hraniční čáry detekovali jako promítanou trasu, což by vedlo k nefunkčnosti celé simulace.

2.2.2 Zapsání základních informací o mapě do souboru

K této úpravě jsem přistoupil z důvodu usnadnění spouštění při vývoji lokalizačního systému. Jak je vysvětleno v souboru *README.md* mnou vytvořená implementace vyžaduje na vstupu rozměry simulace a počet Ozobotů, které má sledovat.

Ruční zadání těchto informací z příkazové řádky je samozřejmě možné, ale při spouštění velkého množství různých map je tento přístup zdlouhavý. *ESO-OzoNav Prototype* má po zpracování souboru s mapou všechny tyto informace k dispozici, a proto jsem do něj pouze dodal funkci, která tyto informace zapíše do souboru, odkud si je po spuštění načte má implementace.

2.2.3 Úprava konfiguračního souboru

Jelikož se rozměry displeje, na němž jsem prováděl simulaci, lehce lišily od rozměrů uvedených v původním konfiguračním souboru pro monitor, upravil jsem konfigurační soubor, tak aby odpovídal mírám mého monitoru.

2.3 Reprezentace oblasti simulace

Simulace je v mé implementaci reprezentována třídou **Grid**. Tato třída si drží následující informace:

- výšku a šírku simulace (v počtu dlaždic)
- vertikální a horizontální body, které určují hranice jednotlivých dlaždic
- obrazovou reprezentaci simulace

Zároveň třída obsahuje metodu *position_in_grid*, která vrátí pozici zadané ohraničující oblasti v simulaci.

2.4 Detekce oblasti simulace

K detekci ohraničujících stěn simulace v obraze jsou nejdříve pomocí metody *HoughLinesP* z OpenCV [11] detekovány všechny čáry. Následně jsou tyto čáry rozděleny na vertikální a horizontální. Jednotlivé detekované čáry ale neodpovídají přímo čárám reálným. Detekované čáry představují spíše úseky reálných čar.

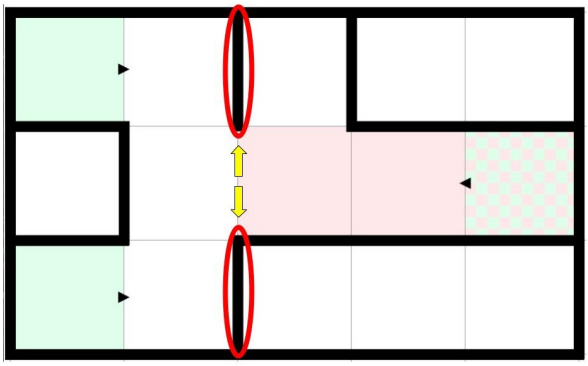
Vertikální čáry jsou proto rozděleny do skupin podle toho jak moc jsou od sebe vzdáleny na horizontální ose. Jedna skupina je pak považována za čáru reálnou. Toto dělení samozřejmě není ideální, protože reálné rozdělené vertikální čáry budou na základě své vzdálenosti na horizontální ose považovány za jedinou čáru. Viz obrázek 2.2. Jelikož je ale tento přístup využit pouze k detekci ohraničujících stěn, které nejsou nijak přerušeny, považuji tuto metodu za dostačující.

Následně jsou vybrány skupiny s druhou nejmenší a druhou největší horizontální souřadnicí. Skupiny s nejmenší a největší souřadnicí v tomto případě představují okraje displeje. Obdobně je pak postupováno v případě čar horizontálních.

Detekovaná oblast je pak na základě zadané výšky a šířky rozdělena na čtvercovou síť.

2.5 Detekce Ozobotů v obraze

Jelikož se kromě Ozobotů a promítané souřadnicové sítě s vodícími čarami ve sledované simulaci žádné jiné objekty nenachází, rozhodl jsem se k počáteční



Obrázek 2.2: Spojené části

detekci Ozobotů využít prostou metodu srovnání šablony. Jedná se o funkci knihovny OpenCV, jejímiž hlavními parametry jsou vstupní obrázek a hledaná šablona.

Šablonu Ozobota vytvoří sám uživatel, označením jednoho z Ozobotů na prvním zachyceném snímku z webkamery.

Funkce poté jako svůj výstup vrátí obraz ve stupních šedi, kde každý pixel představuje váhu s jakou jeho sousedi odpovídají poskytnuté šabloně.

Pomocí funkce *minMaxLoc* následně získám pixel s maximální hodnotou a jeho souřadnice. Tento pixel je považován za střed detekované šablony. Pozice šablony ve vstupním obrázku je tak tímto jednoznačně určena.

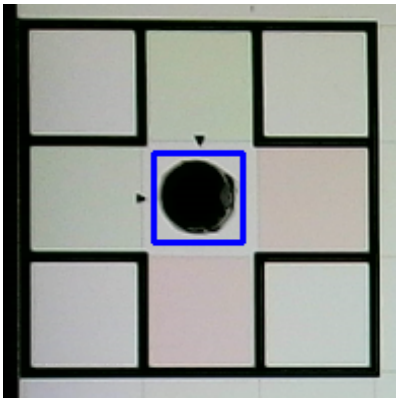
Abych tuto metodu mohl využít k detekci více objektů v jednom obraze, tak jsem jednotlivé oblasti s Ozoboty po jejich úspěšné detekci vždy přebarvil, aby nebyly detekovány znovu.

Pokud bych toto neučinil byl by stále znovu označován Ozobot nejvíce odpovídající zadané šabloně, tedy ten původně vybraný.

2.6 Sledování Ozobotů v obraze

Po prvotní detekci Ozobotů v obraze jsou za pomoci ohraničujících obdélníku inicializovány jednotlivé trackery pro každého Ozobota. Tracker je třída knihovny OpenCV [11] využívající principů MOT popsaných v kapitole 1.4. Použití trackeru je velmi přímočaré, ze začátku je mu dodána pozice objektu, které má ve videu sledovat a následně je mu předáván každý zachycený snímek z kamery. Tracker pak jako svůj výstup vrací pozici objektu, který mu byl předán při inicializaci, v dodaném snímku.

Použití trackeru mi také zajišťuje jednoznačnou identifikaci objektu po celou dobu sledování. Nemusím tedy řešit, která pozice patří konkrétnímu Ozobotovi, narozdíl od implementace zmíněné v kapitole 1.5.



Obrázek 2.3: Uvnitř jedné dlaždice

2.7 Lokalizace Ozobotů v simulaci

Simulace je reprezentována svou souřadnicovou sítí dlaždic. Jelikož se jedná o část roviny, nabízí se jednotlivé dlaždice reprezentovat pomocí souřadnic na osách x a y.

V simulaci jsou dlaždice očíslovány (nulou počínaje) v rostoucím pořádí zleva do prava a shora dolů. Převod mezi souřadnicemi a číslem dlaždice je pak dán jednoduchým vzorcem:

$$cislo\_dlazdice = sirka\_simulace \cdot y + x \tag{2.1}$$

V případě, že Ozobot v nějaké ose vyjede mimo oblast simulace je tato skutečnost reprezentována číslem -1 a takováto pozice je pak rovněž převedena na číslo dlaždice -1

Pozice každého Ozobota v obraze je jednoznačně určena jeho ohraničujícím obdélníkem.

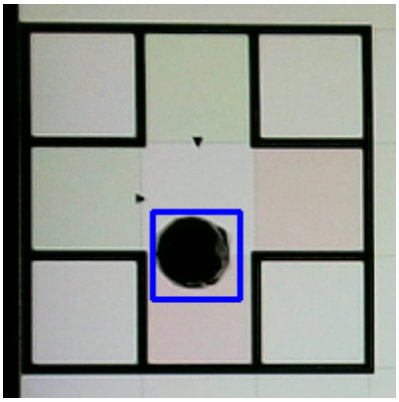
Pozice tohoto obdélníku na jednotlivých dlaždicích pak může být rozdělena na následující případy:

- Obdélník se nachází zcela uvnitř jediné dlaždice
- Obdélník se částečně nachází na dvou sousedních dlaždicích zároveň
- Obdélník se částečně nachází na čtyřech sousedních dlaždicích zároveň

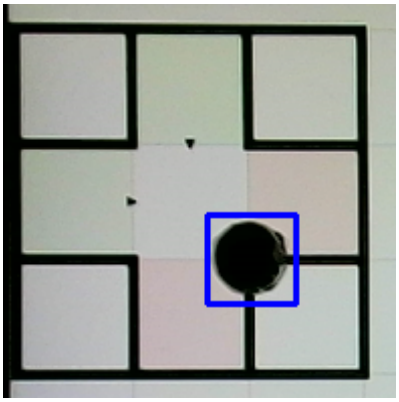
Jednotlivé případy jsou znázorněny na obrázcích 2.3 2.4 2.5.

Ozobot se tedy v jeden časový okamžik může nacházet na více dlaždicích zároveň, což je potřeba při reprezentaci jeho pozice zohlednit.

Pozici Ozobota v grafu jsem se rozhodl reprezentovat jako pole souřadnic dlaždic, na nichž se nachází ohraničující obdélník Ozobota. Každá souřadnice dlaždice je pak doplněna o číslo, které udává kolik procent z obsahu ohraničujícího



Obrázek 2.4: Uvnitř dvou sousedních dlaždic



Obrázek 2.5: Uvnitř čtyř sousedních dlaždic

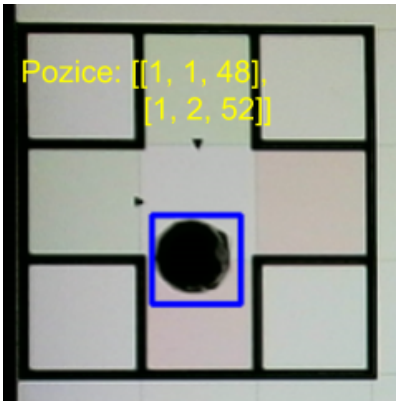
obdélníku se na konkrétní dlaždici nachází. Ilustrativní případ je zachycen na obrázku 2.6

Zjednodušený popis činnosti celého lokalizačního systému je shrnut v pseudokódu 5.

2.8 Detekce Ozobota mimo simulaci

Způsob reprezentace pozice poskytuje bohaté informace a je na uživateli mého systému jak s těmito informacemi dále naloží. Například je možné uvažovat jako pozici Ozobota vždy jen dlaždici na níž se nachází největší část ohraničujícího obdélníku. Na druhou stranu lze také využít informace i o dalších dlaždicích na nichž se nachází ohraničující obdélník a detekovat tak třeba vyjetí Ozobota mimo oblast simulace.

Detekce vyjetí mimo oblast simulace je velmi jednoduchá. Je nutné se pouze podívat, zda seznam pozic ohraničujícího obdélníku pro daného Ozo-



Obrázek 2.6: Spojené části

Algoritmus 5 Lokalizační systém

```
1:  F ← první zachycený snímek z webkamery
2:  Nech uživatele označit prvního Ozobota v F
3:  Pomocí srovnání šablony nalezni ostatní Ozoboty v F
4:  A ← detekovaná oblast simulace v F
5:  while True do
6:      P ← ∅
7:      S ← další zachycený snímek z webkamery
8:      O ← ohraničující obdélníky Ozobotů v S
9:      for a ∈ O do
10:         p ← pozice a v A
11:         P ← P ∪ {p}
12:     zašli P na požadovaný výstup
```

bota neobsahuje souřadnici -1 a v případě, že tomu tak je, spočítat jak velká část ohraničujícího obdélníku na této pozici leží.

Tuto detekci jsem ve své práci realizoval, přičemž jsem za vyjetí ze simulace pokládal situaci, kdy se mimo simulaci nacházelo více jak 12 % z ohraničujícího obdélníku Ozobota. Videozáznam této detekce je zachycen na videu *mimosimulaci.avi* ve složce *mereni*, která je součástí přílohy.

Kapitola 3

Měření

Měření porovnávalo plánované trasy, které měli Ozoboti projet a trasy, které byly zachyceny mou implementací.

Při provádění měření jsem v každém snímku jako pozici Ozobota uvažoval vždy dlaždici na nichž se nacházela největší část z ohraničujícího obdélníku Ozobota.

Pro účely měření je trasa reprezentována čísly dlaždic, kterými Ozobot během simulace projel. Prototyp lokalizuje Ozoboty v každém zachyceném snímku, ale pro účely měření je podstatná trasa, kterou lze rekonstruovat ze zachycených snímků. Pokud je tedy pozice Ozobota na aktuálně zachyceném snímku stejná jako na snímku předchozím nepřidávám tuto pozici do seznamu dlaždic trasy, kterou Ozobot projel. Tedy např. seznam zachycených pozic *[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2]* bude redukován na trasu *[0, 1, 2]*.

Tato redukce bere samozřejmě v úvahu, že se Ozobot může vrátit na dlaždici, kterou již v minulosti navštívil. Např. zachycené pozice *[0, 0, 0, 0, 1, 1, 1, 1, 0, 0]* budou redukovány na trasu *[0, 1, 0]*

Měření bylo realizováno na sedmi mapách různých velikostí a s různým počtem Ozobotů. Na každé z těchto map bylo provedeno deset měření. Ke každému z těchto měření byl pořízen videozáznam.

V případě úspěšného průběhu simulace jsem pouze srovnal seznam dlaždic, které měl Ozobot navštívit dle plánu a seznam dlaždic, které pro daného Ozobota zaznamenala má implementace. Pokud tyto seznamy shodují jedná se o úspěšnou lokalizaci.

V případě, kdy se v simulaci vyskytne nějaká chyba, je pro mě vyhodnocení mírně náročnější. Nemůžu se spolehnout na pouhé porovnání seznamů plánovaných a zaznamenaných pozic, ale musím najít seznamy, které se neshodují. Seznam zaznamenaných pozic pak musím porovnat a ověřit zda odpovídá zachycené skutečnosti na videozáznamu.

Toto je jednodušší v případě, kdy robot pouze ztratí svou trasu a zůstane až do konce simulace na dlaždici/ích, kde trasu ztratil. Zaznamenané pozice jeho trasy by tak měly odpovídat části seznamu pozic plánovaných až do

Tabulka 3.1: Výsledky měření

Mapa	Úspěch	Neúspěch	Chybné simulace
5x3_3a_ordering	10	0	6
5x5_4a_rotation	9	1	5
6x3_4a_corridor	10	0	5
8x3_6a_swap	10	0	3
9x5_6a_roundabout	10	0	3
10x2_6a_snake	10	0	1
10x5_6a_evacuation	9	1	7

okamžiku ztráty trasy.

Ale v případě, že Ozobot na dlaždici/ích nesetrvá (například proto, že dlažicí prochází i trasa jiného Ozobota, který jej posune na sousední dlaždici) je nutné zaznamenané pozice porovnat pozici po pozici s videozáznamem.

Pokud se pozice shodují s pozicemi, které jsem odvodil z videozáznamu považuji lokalizaci za úspěšnou.

Výsledky měření jsou uvedeny v tabulce 3.1. Jak je v tabulce vidět vytvořené prototyp se mýli pouze ve dvou ze sedmdesáti případů (97 % úspěšnost). Tyto chybné lokalizace byly zapříčiněny chybnou detekcí oblasti simulace v obraze, jak je vidět z přiložených videozáznamů. Pro zajímavost je ve sloupci **Chybné simulace** uveden také počet měření při nichž došlo k některé z chyb zmíněných v sekci 1.2.8.

Všechny videozáznamy a zaznamenané trasy, ke každému jednotlivému měření jsou součástí přílohy (složka *mereni*).

Závěr

Cílem mé práce bylo navrhnout, implementovat a s využitím skutečných Ozobotů otestovat lokalizační systém Ozobotů během provádění simulace multi-agentního hledání cest.

V své práci jsem analyzoval výsledky práce přechozí. Dále jsem se věnoval analýze metod určených k řešení problému MAPF a technikám určených ke detekování a sledování objektů v obraze.

Za pomoci poznatků z těchto analýz jsem přistoupil k tvorbě prototypu lokalizačního systému.

Výsledkem je funkční prototyp, který dokáže v kontrolovaných podmínkách v obraze rozpoznat promítanou simulaci, detekovat v ní Ozoboty a ty v průběhu provádění simulace sledovat a navracet jejich pozice v simulaci.

Prototyp byl otestován na mapách různých velikostí a s různým počtem Ozobotů. Jeho přesnost dosahuje dle testů 97 %.

Vytvořením systému lokalizace je umožněn vznik budoucích prací, které mohou využít informací o zachycených pozicích k rozpoznání možných chyb v simulaci a opravě simulace.

V práci bylo prezentováno detekování jedné z těchto možných chyb a to vyjetí Ozobota mimo simulaci. Vytvořený program lze však využít i k detekování chyb jiných.

Literatura

[1] Chudý, J.: *Simulation of Centralized Algorithms for Multi-Agent Path Finding on Real Robots*. Bachelor's thesis, Czech Technical University in Prague,Faculty of Information Technology, 2020.

[2] Ghallab, M.; Nau, D.; Traverso, P.: *Automated planning and acting*. Cambridge University Press, 2016.

[3] Nau, D.; Au, T.-C.; Ilghami, O.; aj.: Applications of shop and shop2. *Intelligent Systems, IEEE*, ročník 20, 04 2005: s. 34 – 41, doi:10.1109/MIS.2005.20.

[4] Silver, D.: Cooperative Pathfinding. *Aiide*, ročník 1, 2005: s. 117–122.

[5] Ryan, M.: Graph Decomposition for Efficient Multi-Robot Path Planning. 01 2007, s. 2003–2008.

[6] Švancara, J.: *Multi-agentní hledání cest v orientovaných prostředích*. Diplomová práce, Univezita Karlova, Matematicko-fyzikální fakulta, 2016. Dostupné z: `https://dspace.cuni.cz/handle/20.500.11956/83051`

[7] Sharon, G.; Stern, R.; Felner, A.; aj.: Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, ročník 219, 2015: s. 40–66, doi:https://doi.org/10.1016/j.artint.2014.11.006.

[8] Zabrodskaya, Y.: *Řešení kolizí mezi geometrickými roboty ve spojitém prostoru*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

[9] Surynek, P.: Lazy Modeling of Variants of Token Swapping Problem and Multi-agent Path Finding through Combination of Satisfiability Modulo Theories and Conflict-based Search. 2018, `1809.05959`.

[10] Surynek, P.: 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-19*, International Joint Conferences on Artificial Intelligence Organization, 7 2019, s. 1177–1183, doi:10.24963/ijcai.2019/164.

[11] Itseez: Open Source Computer Vision Library. `https://github.com/itseez/opencv`, 2015.

[12] Image Thresholding. [online], [cit. 2021-04-31]. Dostupné z: `https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html`

[13] Template Matching. [online], [cit. 2021-04-31]. Dostupné z: `https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_template.html`

[14] Image Processing and Pattern Recognition 2012. [online], [cit. 2021-04-31]. Dostupné z: `https://imageprocessing2012.wordpress.com/`

[15] Luo, W.; Xing, J.; Milan, A.; aj.: Multiple Object Tracking: A Literature Review. 2017, `1409.7618`.

[16] Yang, H.; Shao, L.; Zheng, F.; aj.: Recent advances and trends in visual tracking: A review. *Neurocomputing*, ročník 74, č. 18, 2011: s. 3823–3831, ISSN 0925-2312, doi:https://doi.org/10.1016/j.neucom.2011.07.024.

[17] Murray, S.: Real-time multiple object tracking-a study on the importance of speed. *arXiv preprint arXiv:1709.03572*, 2017.

[18] Dai, C.; Zheng, Y.; Li, X.: Pedestrian detection and tracking in infrared imagery using shape and appearance. *Computer Vision and Image Understanding*, ročník 106, č. 2, 2007: s. 288–299, ISSN 1077-3142, doi:https://doi.org/10.1016/j.cviu.2006.08.009, special issue on Advances in Vision Algorithms and Systems beyond the Visible Spectrum.

[19] Xing, J.; Ai, H.; Liu, L.; aj.: Multiple Player Tracking in Sports Video: A Dual-Mode Two-Way Bayesian Inference Approach With Progressive Observation Modeling. *IEEE Transactions on Image Processing*, ročník 20, č. 6, 2011: s. 1652–1667, doi:10.1109/TIP.2010.2102045.

[20] Wu, B.-F.; Lin, S.-P.; Chen, Y.-H.: A real-time multiple-vehicle detection and tracking system with prior occlusion detection and resolution. In *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005.*, 2005, s. 311–316, doi:10.1109/ISSPIT.2005.1577115.

[21] Spampinato., C.; Chen-Burger., Y.; Nadarajan., G.; aj.: Detecting, Tracking and Counting Fish in Low Quality Unconstrained Underwater Videos. In *Proceedings of the Third International Conference on Computer Vision Theory and Applications - Volume 1: VISAPP, (VISIGRAPP 2008)*, INSTICC, SciTePress, 2008, ISBN 978-989-8111-21-0, ISSN 2184-4321, s. 514–519, doi:10.5220/0001077705140519.

[22] Lechovský, A.: *Vizuální sledování skupiny robotů*. Bakalářská práce, Univezita Karlova, Matematicko-fyzikální fakulta, 2020. Dostupné z: `https://dspace.cuni.cz/handle/20.500.11956/109064`

Příloha **A**

Seznam použitých zkratek

MAPF Multi-Agent Pathfinding

DFT Detection-Free Tracking

DBT Detection-Based Tracking

CBS Conflict-based search

MOT Multiple object tracking

Příloha **B**

Obsah přiložené SD karty

- `ozobot-mapf-simulator` ..... upravený program simulace
- `mereni` ..... záznamy měření
- `boOX.zip` ..... komprimovaný program boOX
- `src`
  - `src.py` ..... zdrojový kód implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `README.md` ..... návod ke spuštění
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF