České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Vizuální inspekce 3D infrastruktury s využitím bezpilotních prostředků

Diplomová práce

Jiří František

Program: Otevřená informatika
Obor: Softwarové inženýrství
Vedoucí práce: Ing. Milan Rollo, Ph.D.

Praha, Prosinec 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **František** Jméno: **Jiří** Osobní číslo: **406892**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vizuální inspekce 3D infrastruktury s využitím bezpilotních prostředků

Název diplomové práce anglicky:

Visual inspection of 3D infrastructures using unmanned aerial vehicles

Pokyny pro vypracování:

1. Seznamte se se stávajícími přístupy k vizuální inspekci 3D infrastruktur s využitím UAV
2. Zpracujte přehled existujících metod pro stanovení inspekčních bodů
3. Navrhněte algoritmus pro stanovení inspekčních bodů pro obecný 3D objekt popsaný ve formě polygonové sítě. Algoritmus musí brát v potaz parametry kamery (rozlišení snímače a úhel záběru) a požadavek na kvalitu výstupních dat (rozlišení snímků v cm/px a překryv snímků)
4. Navržený algoritmus implementujte a otestujte jeho vlastnosti v simulaci
5. Experimentálně ověřte vlastnosti algoritmu nasazením na reálný bezpilotní prostředek
6. Diskutujte kvalitu výstupů a faktory ovlivňující celý proces vizuální inspekce

Seznam doporučené literatury:

[1] Hallermann, Norman, and Guido Morgenthal. "Visual inspection strategies for large bridges using Unmanned Aerial Vehicles (UAV)." Proc. of 7th IABMAS, International Conference on Bridge Maintenance, Safety and Management. 2014.
[2] Chloupek M.: Plánování trajektorie bezpilotního prostředku pro inspekci 3D objektů, Bakalářská práce, ČVUT v Praze, 2016.
[3] Morgenthal, G., and N. Hallermann. "Quality assessment of unmanned aerial vehicle (UAV) based visual inspection of structures." Advances in Structural Engineering 17.3 (2014): 289-302.
[4] Ham, Youngjib, et al. "Visual monitoring of civil infrastructure systems via camera-equipped Unmanned Aerial Vehicles (UAVs): a review of related works." Visualization in Engineering 4.1 (2016).

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Milan Rollo, Ph.D., centrum umělé inteligence FEL

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

Datum zadání diplomové práce: **11.02.2020** Termín odevzdání diplomové práce: **05.01.2021**

Platnost zadání diplomové práce: **30.09.2021**

Ing. Milan Rollo, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Milanovi Rollovi, Ph.D. za jeho ochotu, pomoc a rady, které mi pomohly s dokončením práce. Poděkování patří také Ing. Vojtěchu Kaiserovi za pomoc při řešení technických problémů s 3D zobrazovacím frameworkem, který jsem používal pro vypracování praktické části 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.

V Praze, Prosinec 2020

Abstrakt

Tato práce se zabývá úlohou vizuální inspekce 3D objektů za použití bezpilotních prostředků. Na základě vstupního polygonálního modelu a požadovaných parametrů, které by měl výstup splňovat (kvalita snímků, překrytí snímků), jsou vygenerovány inspekční body v prostoru kolem modelu. Tyto body slouží jako pozice pro dron, ze kterých se objekt vyfotografuje. Z inspekčních bodů jsou vybrány nejvhodnější, kterými se vytvořila cesta pro dron. Algoritmus byl otestován na různých modelech s různým nastavením vstupních parametrů. Výstup algoritmu je výsledná cesta pro dron, která se skládá z bodů průletu a bodů sloužících k pořízení snímku.
K praktické části práce jsem jako základ použil vizualizační systém z frameworku AgentFly, který obsahuje podporu pro práci s polygonálními modely a kamerou v 3D prostoru.

Klíčová slova: inspekce, dron, bezpilotní prostředek, polygonální model, inspekční body, 3D prostor, kamera, trajektorie pro inspekci

Abstract

The following work is focused on the implementation of the visual inspection of 3D objects using UAVs task. Based on the input mesh model and required parameters (photo quality, photo overlap) inspection points were generated around the model. These points are positions from which drone takes a picture of inspected object. Out of all of these inspection points only the most suitable ones were chosen for the final path. Algorithm was tested on different models with different input parameters. Final output of this work is a path for a UAV. The path is constructed from inspection points and fly-through points.
For the practical part of this work I used visualisation system from AgentFly framework which contains support for work with mesh models and camera in 3D space.

Keywords: inspection, drone, UAV, mesh model, inspection points, 3D space, camera, inspection path

Seznam tabulek

Seznam obrázků

Seznam zkratek

AWT (Abstract Window Toolkit) Java knihovna pro grafické uživatelské rozhaní.

BPA (Ball-pivoting algorithm) Algoritmus sestavující trojúhelníkovou síť z mračna bodů v prostoru.

FOV (Field of view) Úhel záběru, Zorné pole.

ILP (Integer linear programming) Celočíselné lineární programování.

SWT (Standard Widget Toolkit) Java knihovna pro grafické uživatelské rozhaní.

TSP (Travelling salesman problem) Problém obchodního cestujícího.

UAV (Unmanned aerial vehicle) Bezpilotní prostředek, dron.

Obsah

Kapitola 1

Úvod

1.1 Motivace

Všechny objekty průmyslové infrastruktury podléhají stárnutí a procesy s ním spojenými, které ovlivňují jeho strukturu. Mimo stárnutí může být struktura objektu narušena působením vnějších vlivů (např. zemětřesení), špatnými technologickými postupy, lidskou chybou a dalšími způsoby. Proto je potřeba objekty, které chceme, aby vydržely co nejdéle a nezpůsobily žádnou újmu na životech nebo majetku, kontrolovat. Mezi takové objekty můžeme zařadit například mosty, budovy, kulturní památky atd.
Protože samotnou inspekci musí provádět specializovaný odborník, může být problematické (jako například kontrola špatně dostupných míst) a nákladné, aby osobně zkontroloval celý objekt z bezprostřední vzdálenosti. Proto dává smysl využít k inspekci takovýchto objektů bezpilotních prostředků (UAV, dron), které jsou s vývojem moderních technologií stále dostupnější. Takový dron může celý objekt (například již zmíněný most) obletět, nafotografovat a fotografie může prohlédnout odborník, aniž by se vystavoval rizikům spojeným s osobní kontrolou přímo na místě. Další výhody tohoto přístupu jsou rychlost inspekce a nižší celkové náklady, protože se například nemusí kolem objektu stavět lešení.

1.2 Cíl práce

Cílem diplomové práce je prozkoumat již existující přístupy inspekce 3D infrastruktury. Dále navržení vlastního algoritmu pro nalezení inspekčních bodů, výběr nejvhodnějších inspekčních bodů a naplánování trajektorie jdoucí těmito body pro dron. Poté otestovat chování algoritmu změnami vstupních dat a parametrů. Nakonec výslednou trajektorii vyzkoušet na simulátoru letu.

1.3 Struktura práce

Práce je rozdělena do následujících kapitol. První kapitola je o seznámení se s problémem a motivací daný problém řešit. Druhá kapitola je přehled existujících přístupů. Ve třetí kapitole se nachází definování pojmů, které jsou v práci použity. Čtvrtá kapitola popisuje AgentFly framework vyvinutý na katedře počítačů, fakultě elektrotechnické, ČVUT, který jsem použil k řešení práce. Pátá kapitola popisuje navržený algoritmus na vyhledávání inspekčních bodů. Šestá kapitola popisuje sestavení trajektorie pro dron z inspekčních bodů. V sedmé kapitole se nachází popis implementace. Osmá kapitola se zabývá otestováním algoritmu změnami vstupních parametrů a nasimulování průletu výsledné trajektorie. V závěrečné deváté kapitole se nachází zhodnocení algoritmu, seznam problémů, návrhy na zlepšení a možnosti pokračování práce.

Kapitola 2

Současný stav řešené problematiky

S rozrůstající se dostupností bezpilotních prostředků (neboli dronů) se navyšuje i jejich využití. Tématikou se zabývá řada odborných publikací. Všechny tyto práce mají stejný základ, tedy najít body, ze kterých se bude provádět inspekce (inspekční bod) a pak těmito body vytvořit cestu. Většina prací neobsahuje, jak přesně se tyto body hledají. Pravděpodobně z toho důvodu, že se toto řešení dá komerčně využít, jako to dělá například světový gigant v oboru IT Intel [1]. Jedním z řešení je práce [2] ve které se využíval jako vstupní data soubor s mesh modelem. Pro dokonalé pokrytí modelu bylo určeno, že každý trojúhelník z modelu musí mít vlastní inspekční bod, bohužel nebylo řečeno, jak přesně se tyto body hledají. Pozice těchto bodů se dále optimalizují na co nejmenší vzdálenosti pro výslednou cestu za pomocí konvexního programování a TSP. Výsledná cesta skrz inspekční body se obecně vytváří za pomocí TSP [3].
Další z existujících přístupů k tomuto problému je použití plánovacího softwaru, který podle známých GPS údajů inspektovaného objektu naplánuje cestu pro dron tak, aby létal konstantní rychlostí v konstantní vzdálenosti od objektu a pořizoval snímky v předem daných časových intervalech [4]. Tento přístup funguje dobře v případě, že máme velké a relativně rovné plochy. Část objektu, která by byla hůře dostupná a neležela přímo před dronem takto nebude nejspíš nasnímána.
Obdobný přístup byl také zvolen pro hledání inspekčních bodů pro inspekci městských struktur [5]. V tomto článku jsou jako inspekční body zvoleny takové body, které leží na kružnicích se středem v snímaném objektu. Kružnic je zvoleno několik s různým poloměrem a s různou vzdáleností od země Body se na kružnicích nachází v takové vzdálenosti od sebe, aby se každé 2 snímky vedle sebe překrývaly z 80 %. Tento způsob má také pouze omezené využití. V případě, že budeme chtít provést inspekci nějakého nepravidelného objektu, tak se některé inspekční body budou nacházet v moc velké vzdálenosti a výsledný snímek nesplní požadovanou kvalitu snímaného objektu.
Další řešení problému vizuální inspekce za použití UAV se vztahuje na inspekci sloupů velmi vysokého napětí [6]. Zde se nejdříve manuálně létá a za pomocí LIDARu se hledá, v jakém prostoru se nachází sloup, případně elektrické vedení. Během tohoto letu se manuálně vybírají inspekční body. Po tomto letu se kolem sloupu vytvoří softwarová bariéra ve formě kvádru, kam UAV v automatickém režimu nesmí. Poté se UAV vyšle v automatickém režimu do inspekčních bodů. Pokud se nějaký bod nachází uvnitř bariéry, tak se UAV přepne do manuálního režimu a pilot provede let do inspekčního bodu a ven z bariéry. Takto se pokračuje, dokud se neprovede celá inspekce. Plánovací software se postupně učí cestu včetně té, která byla provedena v manuálním režimu. Zde se počítá s tím, že se inspekce bude provádět na větším množství sloupů, na prvním sloupu se software naučí trasu a na dalších už bude let dronu probíhat bez zásahu člověka. Tento přístup stále vyžaduje zásahy člověka, a to jak během pilotování, tak s výběrem inspekčních bodů. Navíc plně automatické získávání snímků k inspekci je možné až od inspekce druhého objektu, který musí být stejný, jako první.
Článek řešící vzdálenou inspekci budov [7] popisuje nevhodnost využití automatického letu z

důvodu nepřesnosti výšky letu za použití GPS a doporučuje manuální ovládání UAV a létání okolo snímaného objektu po rovnoběžných úsečkách.
Práce zabývající se inspekcí mostů navrhuje vybrat jako inspekční body takové body, aby osa kamery byla kolmo ke snímané ploše [8]. Tento přístup nemusí pokrýt celý objekt, protože některé části objektu nemusí být vůbec pokryty ze žádného inspekčního bodu.
Jedna z dalších prací řešící vizuální inspekci komplexních struktur [9], navrhuje prolétnout okolo objektu po spirále konstantní rychlostí a snímky pořizovat v určitých časových intervalech. Další návrh stejných autorů je náhodně generovat body v okolí objektu a za pomocí konvexního programování optimalizovat ideální rozložení těchto bodů. Náhodný přístup nezaručuje, že se v případě složitějšího objektu pokryje úplně celý.
Přístup manuálního výběru inspekčních bodů navržený v práci [10] umožňuje inspekci pouze vybraných míst. Autoři práce navrhují nejdříve obletět objekt a za použití LIDARu vytvořit jeho model. Poté v softwaru manuálně vytvořit inspekční body. Stále je zde problém, že proces neprobíhá zcela automaticky, stále zde musí být pilot, který provede prvotní let a pak osoba vybírající inspekční body.
Další práce zabývající se nalezením inspekčních bodů pro vizuální inspekci budov [11] navrhuje vytvořit v blízkosti budovy zakázaný prostor ve tvaru krychle. Do tohoto prostoru dron nesmí. Inspekční body jsou zvoleny na stěnách této krychle ve stejné vzdálenosti od sebe. Lze si to představit tak, že tyto body vyplňují mřížku ležící na stěně krychle reprezentující zakázaný prostor pro dron. Tento přístup obdobně jako předchozí neřeší složitější objekty a hůře dostupná místa objektu.
Poslední zmíněná práce, která řeší tento problém je vypracovaná Bakalářská práce na ČVUT FEL [12]. V této práci ze začíná mračnem bodů pořízených LIDARem, ze kterého je zrekonstruovaný polygonální (trojúhelníkový) model za pomocí BPA algoritmu. Ke všem trojúhelníkům se vytvoří kolmice z jejich těžiště a jako inspekční bod se zvolí bod na této kolmici ve vzdálenosti x ve směru od modelu ven. Body, které se nachází blízko modelu nebo uvnitř modelu se smažou. Dále se vytvoří stavový prostor, který slouží ke kontrolování kolize UAV s modelem. V této fázi se mezi inspekčními body vypočítají vzdálenosti A* algoritmem a najde se mezi body cesta za pomocí TSP.

Kapitola 3

Formalizace problému

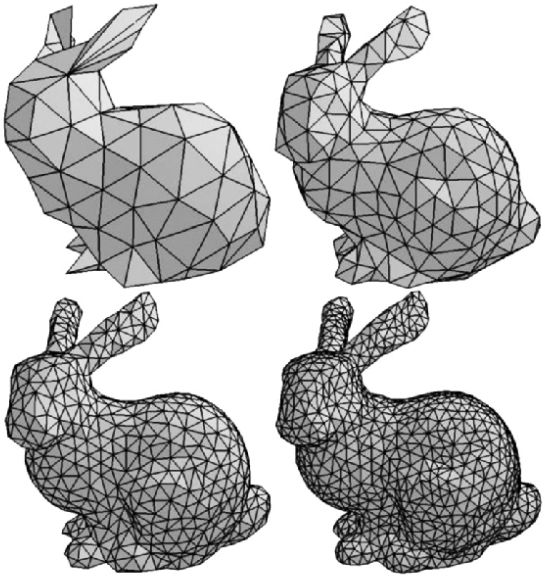
V této části si zadefinujeme použité pojmy, řešenou úlohu a výstup úlohy.

3.1 Polygonální síť

Polygonální síť neboli mesh je způsob definování 3D objektů v počítačové grafice. Skládá se z vrcholů, hran a ploch, které dohromady definují tvar objektu. Nejčastěji používanou polygonální sítí v počítačové grafice je trojúhelníková síť, kterou budu také využívat v této práci. Vrchol je reprezentován souřadnicemi (x, y, z) v Kartézském souřadnicovém systému, hrana je reprezentována jako spojnice dvou vrcholů a plocha je ohraničena třemi hranami. Trojúhelník má 2 plochy, z nichž nás zajímá ta, která směřuje směrem ven z objektu neboli vnější plocha objektu a její normála neboli vektor, který je kolmý na tuto plochu a směřuje také směrem ven z objektu. Tento směr je definovaný normálou k trojúhelníku, které musí být buď definována ve vstupních datech nebo ji lze zjistit z vrcholů polygonů, které bývají podle konvence uloženy v pořadí proti směru hodinových ručiček. Polygonální síť reprezentuje pouze povrch objektu z čehož vychází největší nevýhoda této reprezentace dat a to, že se nedají přesně reprezentovat zakřivené plochy. Zakřivená plocha se v polygonální síti vytvoří rozpadem polygonu na menší, které spolu svírají různé úhly. Tyto menší polygony, které jsou vzájemně pod různými úhly, vytvoří žádaný tvar. Čím kvalitnější 3D model chceme mít, tím větší množství polygonů je potřeba. Při modelování některých objektů s velkým počtem polygonů dochází k razantnímu zpomalení celého výpočetního algoritmu, který s modelem pracuje.
Proto se 3D modely aproximují, aby měly menší počet polygonů a stále vypadaly co nejrealističtěji.

Polygonální model

Polygonální model nebo také objekt je tvořen množinou trojúhelníků, které jsou tvořeny vrcholy v R^3, spojenými úsečkami. Tyto trojúhelníky mají definovanou normálu směřující ve směru vnější plochy a velikost této plochy. Objekt musí být volumetricky uzavřený. Ukázka polygonálního modelu s různým počtem trojúhelníků je na obrázku 3.1.



Obrázek 3.1: Ukázka trojúhelníkové sítě. Jde o stejný objekt s různým počtem trojůhelníků [13]

3.2 Kamera

Kamera je optický přístroj, který pořizuje snímky. Dnešní moderní digitální kamery i fotoaparáty mohou pořizovat jednotlivé snímky nebo video (sekvence mnoha snímků v krátkém časovém úseku). Pro případ této práce se budou používat jednotlivé snímky. Hlavní vlastnosti kamery jsou rozlišení, v případě objektivu kamery jde o ohniskovou vzdálenost (nebo FOV) a světelnost.

3.2.1 Rozlišení

Rozlišení udává počet buňek na snímači kamery, které zároveň definují rozlišení snímku. Udává se jako počet sloupců a řádků (např.: 1920 x 1080 říká, že fotografie má v horizontálním směru 1920 pixelů a ve vertikálním 1080). Z rozlišení je možné vypočítat poměr stran snímku/čipu, který se používá k převádění vertikálního FOV na horizontální a obráceně.

$$r = \frac{h}{v}$$

Kde r je poměr stran, h je počet horizontálních pixelů a v je počet vertikálních pixelů.

3.2.2 FOV

FOV (neboli úhel záběru) je část prostoru, který je schopna kamera zachytit. Neboli ta část prostoru, ze které do objektivu kamery přichází paprsky světla. Existuje několik typů FOV:

- horizontální
- vertikální
- diagonální

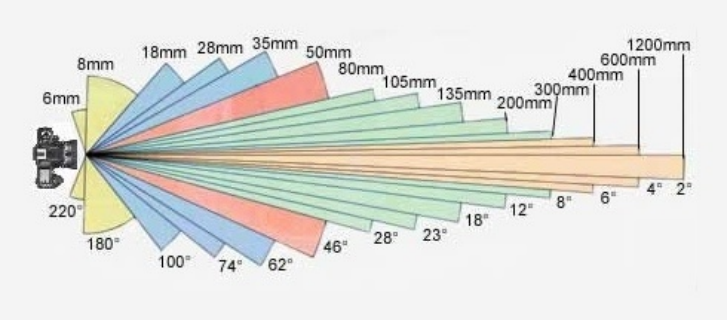
Tyto FOV jdou mezi sebou převádět, musíme k tomu ale znát rozlišení nebo poměr stran snímku (čipu). Výpočet horizontálního FOV z vertikálního:

$$FOV_h = FOV_v \cdot r$$

Kde r je poměr stran. A výpočet diagonálního FOV z horizontálního a vertikálního:

$$FOV_d = \sqrt{FOV_v^2 + FOV_h^2}$$

FOV říká, jaký je úhel mezi krajními body, které je kamera schopna zachytit [14].
FOV je úzce spojeno s ohniskovou vzdáleností. To je vzdálenost mezi optickým středem objektivu a rovinou, na které se protínají paprsky světla, které prochází objektivem [15]. Ohnisková vzdálenost se značí písmenem **f** a nejčastěji bývají použité jednotky milimetry. Ukázka snímaného prostoru kamerou v závislosti na nastavení FOV nebo ohniskové vzdálenosti objektivu je na obrázku 3.2.



Obrázek 3.2: Snímaný prostor v závislosti na FOV a ohniskové vzdálenosti[1]

Ohnisková vzdálenost se dá přepočítat na FOV ale je k tomu potřeba znát rozměry čipu.

$$FOV_h = 2arctan\frac{h}{2f}$$

$$FOV_v = 2arctan\frac{v}{2f}$$

$$FOV_d = 2arctan\frac{d}{2f}$$

Kde h, v a d jsou horizontální, vertikální a diagonální velikost čipu a f je ohnisková vzdálenost objektivu.

3.3 Vstupní parametry

Případného zákazníka, který by chtěl využít tohoto řešení, zajímá hlavně co má být inspektováno, s jakou kvalitou to má být nasnímáno a kolikanásobný má být překryv snímků.

- V případě toho, co má být inspektováno jde o trojúhelníkový model objektu.
- Kvalita snímku udává jeho požadované minimální rozlišení. Nejde zde o rozlišení kamery, ale o maximální nasnímanou plochu objektu jedním pixelem. Pokud pixel na výsledném snímku nasnímal větší plochu objektu než udává tento parametr, tak nebyl splněn požadavek na kvalitu snímku.
- Překryv snímků definuje z kolika inspekčních bodů má být trojúhelník nasnímán. Jinak řečeno, každý trojúhelník z modelu musí být nasnímán z alespoň požadovaného množství inspekčních bodů. Taktéž v práci nazýváno redundancí snímků.

[1] Obrázek stažen z `https://www.alza.cz/slovnik/ohniskova-vzdalenost-art4717.htm`

3.4 Inspekční bod

Inspekční bod je takový bod v prostoru, ze kterého bude UAV provádět inspekci (pořídí fotografii). Takovýto bod musí být v dostatečné vzdálenosti od objektu, aby nedošlo ke kolizi UAV a inspektovaného objektu, ale nesmí být příliš daleko, kvůli tomu, aby bylo UAV schopno poskytnout fotografii s dostatečnou (předem definovanou) kvalitou zkoumané části objektu.
Inspekční bod se skládá z údajů:

- **Pozice** - jde o pozici v prostoru definovanou souřadnicí v R^3.
- **Směr** - jde o vektor v R^3, který nám říká, jakým směrem má být nasměrovaná kamera na dronu.
- **Seznam viditelných trojúhelníků** - seznam trojúhelníků, které jsou z daného bodu kompletně viditelné.
- **Seznam částečně viditelných trojúhelníků** - seznam trojúhelníků, které jsou z daného bodu vidět jen částečně spolu se záznamem z jak velké části jsou vidět.
- **Seznam trojúhelníků nesplňujících požadovanou kvalitu** - seznam trojúhelníků, které jsou z daného bodu aspoň částečně viditelné, ale nesplňovaly by na snímku minimální požadovanou kvalitu.

3.5 Skenování

Skenováním je v práci označeno vysílání paprsků z kamery a zaznamenávání nasnímané plochy trojúhelníků u každého pixelu. Tyto paprsky reprezentují pixely výsledného snímku. Skenování se provádí pro zjištění viditelnosti trojúhelníků modelu z vybraných bodů v prostoru. Jak se provádí skenování je popsáno v sekci 7.3.

3.5.1 Viditelný trojúhelník

Trojúhelník je označený za viditelný, pokud k němu existuje inspekční bod, který je schopen nasnímat předem definovanou část jeho plochy s alespoň požadovanou kvalitou. Stejný význam má slovní spojení, že trojúhelník je pokrytý.

3.5.2 Trojúhelník nesplňující požadovanou kvalitu

Trojúhelník, který nesplňuje požadovanou kvalitu z inspekčního bodu, je takový trojúhelník, který z daného inspekčního bodu byl nasnímán aslepoň jedním pixelem, který pokryl větší plochu tohoto trojúhelníku než je definováno parametrem kvality snímku.

3.5.3 Částečně viditelný trojúhelník

Trojúhelník je označen za částečně viditelný z daného inspekčního bodu, pokud je z něj zasažen minimálně jedním paprskem a zároveň nespadá do kategorie viditelného trojúhelníku a trojúhelníku nesplňujícího požadovanou kvalitu.

3.5.4 Nenasnímaný trojúhelník

Nenasnímaný nebo také trojúhelník, který není vidět je takový trojúhelník, ke kterému neexistuje ani jeden inspekční bod, ze kterého by ho během skenování zasáhl aspoň 1 paprsek.

3.6 Definice problému

Úkolem je najít takovou množinu inspekčních bodů, aby tato množina byla co nejmenší a zároveň, aby každý trojúhelník ze vstupního trojúhelníkového modelu byl viditelný alespoň z požadovaného množství inspekčních bodů a přes tuto množinu inspekčních bodů sestavit co nejkratší cestu.

3.7 Výstup

Výstupem je trajektorie pro dron, která je definována seznamem bodů průletu (bod v R^3, tímto bodem se musí proletět) a inspekčními body (v nichž se fotografuje), které jsou seřazeny v pořadí, v jakém jsou dronem postupně navštíveny. Tato trajektorie dále zajišťuje bezkolizní let.

Kapitola 4

AgentFly framework

Tato práce je založena na simulačním frameworku AgentFly[1]. Framework slouží k simulaci civilní i bezpilotní letecké dopravy, decentralizovanému řešení kolizí s využitím detailních modelů letadel i okolního prostředí. Tento framework je napsán v programovacím jazyce Java[2].
Pro potřeby této práce byla využita pouze jeho vizualizační část. Tato část slouží k vizualizaci 3D modelů, práci s kamerou a vykreslování základních útvarů. Framework je postaven na OpenGL[3] verze 4.5 za použití JOGL[4] (Java OpenGL), která dovoluje použití OpenGL v jazyce Java, který tuto funkci v základu nepodporuje. Dále podporuje AWT a SWT. Více informací o frameworku lze najít v práci [16].
Framework načítá 3D objekty ze souborů ve formátu .obj[5], které potom vykresluje do scény. Dále umožňuje práci s kamerou, jako například nastavení pozice a směru kamery, nastavení atributů kamery (rozlišení, FOV, ...), vyslání paprsků po jednotlivých pixelech do prostoru. K uložení trojúhelníků modelu obsahuje podporu ve formě datové struktury k-d stromu, nad kterou je možno provést dotaz zasažení trojúhelníku paprskem.
V práci jsem využil následujících tříd z frameworku:

- **LayerProvider** je základní prvek pro vykreslení objektů do scény. Každý Layer provider má přístup ke svému vlastnímu kořenu scény, objektu kamery, se kterou může manipulovat a uživatelskému vstupu implementovaném přes posluchače událostí. Layer provider obsahuje vykreslované objekty, které se vykreslí po závěrečném zápisu (commit).
- **LeafGroup** slouží k uložení elementů na vykreslení, převážně načtených .obj souborů.
- **KDTree** datová struktura, která slouží k uložení trojůhelníků z modelu a provádění dotazů na průnik paprsku s trojúhelníkem.
- **CameraSensor** třída, která obaluje funkce senzoru kamery a dovoluje skenování a detekování objektů uložených v KDTree za použití vysílání paprsků.
- **TransformGroup** slouží k uložení elementů (vyjmenovány níže) na vykreslení, na nichž mohou být provedeny nějaké transformace.
- **Lines** třída reprezentující křivku definovanou seznamem souřadnic. Křivce se může definovat různá barva a tloušťka čáry.
- **Points** třída reprezentující body v prostoru, které jsou definované seznamem souřadnic. Body mohou mít definovanou barvu a velikost (průměr koule reprezentující bod v prostoru).

[1] `https://www.agentfly.com/`
[2] `https://www.oracle.com/cz/java/`
[3] `https://www.opengl.org/`
[4] `https://jogamp.org/jogl/www/`
[5] `https://en.wikipedia.org/wiki/Wavefront_.obj_file`

- **Mesh** třída reprezentující model objektu s vrcholy, barvou a texturami, normálami a dalšími atributy.
- **Triangle3f** třída reprezentující trojúhelník z mesh modelu objektu.
- **Ray3D** třída reprezentující paprsek v prostoru definovaný jeho pozicí a vektorem směru.

Kapitola 5

Algoritmus hledání inspekčních bodů

V této kapitole je popsán algoritmus vyhledávání inspekčních bodů.

5.1 Vstupní data

Algoritmus vyhledávající inspekční body obsahuje mnoho vstupních dat, která ovlivní výsledek a běh tohoto algoritmu. Tyto data se dělí do dvou skupin:

- Zákaznická data jsou požadavky od potenciálního zákazníka, který chce využít tohoto řešení k inspekci objektu.
- Uživatelská data jsou parametry programu, které se vhodně zvolí na základě dat a požadavků od zákazníka.

Uživatelské vstupní parametry volí osoba, která je seznámena s fungování algoritmu. Tyto parametry budou popsány až později v místě využití. Zákaznické parametry byly představeny již dříve v sekci s definicí problému (3.6).
Je potřeba si ještě blíže specifikovat vstupní model. Vzhledem k využití AgentFly frameworku, který načítá a zobrazuje .obj soubory je hlavním vstupem práce trojúhelníkový objekt v tomto formátu. Důležité je, aby tento model byl volumetricky uzavřený (to znamená, že nemá ve svém povrchu nikde žádnou díru nebo chybějící trojúhelníky) z důvodu, který bude popsán později. Dále tento model musí splňovat podmínku, že jeho osa z v kartézském souřadnicovém systému reprezentuje výšku nad terénem v reálném světě. U modelu se musí dávat pozor na jeho měřítko, podle zvoleného modelu se dále upravují další vstupní parametry. Bylo by vhodné, aby model neobsahoval zbytečné trojúhelníky uvnitř nebo kompletně zakryté jinou částí modelu. Takovýto model lze získat fotogrammetrií, ale musí se dát pozor, aby výsledek fotogrammetrie byl uzavřený, popřípadě model upravit v nějakém softwaru (například Blender[1]).

Fotogrammetrie

Fotogrammetrie je vědní disciplína, která se zabývá zpracováváním informací z fotografií. Například zjištění rozměrů nebo polohy objektu na fotografii, ale také třeba rekonstrukci celých objektů do 3D. V případě pouze jedné fotografie můžeme zjistit souřadnice objektu ve 2D, pokud chceme zjistit souřadnice ve 3D potřebujeme minimálně 2 fotografie stejného objektu, které jsou vyfoceny z různých míst [17].

5.2 Kolize

K zajištění bezpečného letu dronu je zapotřebí, aby všechny inspekční body ležely v bezpečné vzdálenosti od objektu. Zjištění kolize dvou objektů (model, UAV) v prostoru patří mezi výpočetně

[1] `https://www.blender.org/`

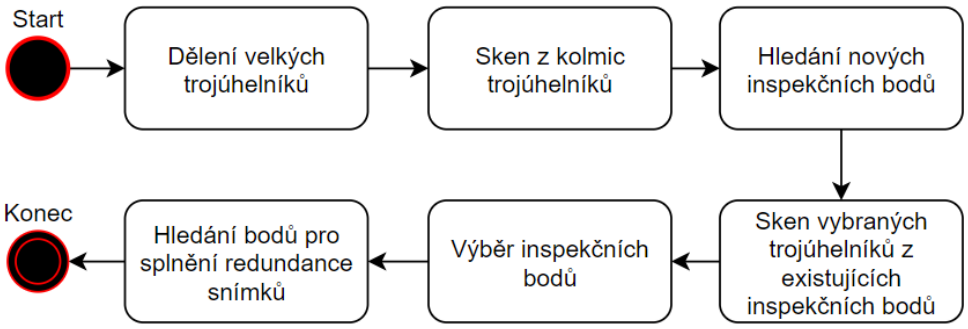
náročné problémy, obzvláště pokud by se mělo v každém bodě zkoušet iterovat přes všechny trojúhelníky z modelu. Další možností je kolize se zemí. Jako úroveň země byla v práci zvolena hladina se souřadnicí $z = 0$. Vše, co je pod tuto hladinu se bere, jako nedostupné. Ke zjišťování kolize se využije voxelizace modelu, viz. kapitola 7.2.

5.2.1 Voxelizace modelu

Voxelizace je proces transformace objektu z jednoho datového typu reprezentujícího 3D model do jeho objemové reprezentace za pomocí 3D pole. Jde o ekvivalent rasterizace ve 2D. Buňce ve voxelizovaném modelu se říká voxel. Voxelizovaný model je transformován do 3D pole (kvádr vyplněný voxely), se záznamy o tom, zda-li je voxel plný (model se v tomto prostoru nachází) nebo prázdný (model se v tomto prostoru nenachází) [18].
Voxelizace je důvod, proč musí být vstupní polygonální model uzavřený, pokud by nebyl uzavřený, tak se nevygenerují plné voxely uvnitř modelu.

5.3 Zvolený postup hledání inspekčních bodů

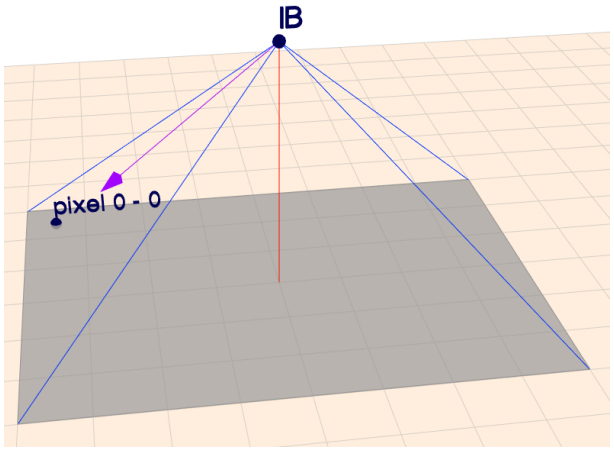
Obdobně jako v pracích popsaných na začátku řekněme, že objekt je kompletně nasnímaný, pokud je každý jeho trojúhelník nasnímaný. To znamená, že pro každý trojúhelník t z modelu objektu existuje inspekční bod takový, že množina trojúhelníků, která jsou z něj označené jako viditelné obsahuje t. Musí se tedy najít takové inspekční body, ze kterých se pokryje celý objekt. Na začátek jsem zvolil stejný přístup jako autor v [12], tedy zvolit inspekční body ležící na normálách trojúhelníků procházející jejich těžištěm. Tento přístup ovšem vystačí pouze na nejjednodušší tvary, což v případě zmíněné Bakalářské práce již nebylo dále řešeno, protože její postup byl testován na krychli a kouli. Pokud si vezmeme složitější objekt, třeba most, budovu s přístřeškem nebo s pilířem, tak zjistíme, že velké množství trojúhelníků by nebylo pokryto žádným inspekčním bodem (trojúhelník by nebyl z inspekčního bodu viditelný), protože inspekční bod ležící na kolmici by byl v kolizi s objektem. Před hledáním inspekčních bodů se provede dělení velkých trojúhelníků. Po skenu z kolmic přichází na řadu algoritmus vyhledávající inspekční body pro hůře přístupné trojúhelníky, který je popsán níže. Tento algoritmus se pustí v několika cyklech, mezi nimiž se provádí dělení trojúhelníků, pokud to u nich má cenu, viz sekce 5.5.2.
Po fázi hledání adeptů na inspekční body přichází na řadu výběr inspekčních bodů pro ideální pokrytí modelu, jelikož inspekčních bodů je vygenerování veliké množství, v ideálním případě pro každý trojúhelník jeden, tak se tento počet musí zredukovat, protože většina inspekčních bodů jsou redundantní s dalšími inspekčními body. Následuje poslední vyhledávání inspekčních bodů pro splnění redundance snímků trojúhelníků (překryvu snímků). Diagram znázorňující celý proces získávání finálních inspekčních bodů lze vidět na obrázku 5.1.



Obrázek 5.1: Diagram fází řešení problému inspekce

5.3.1 Maximální vzdálenost inspekčního bodu od trojúhelníku

Maximální vzdálenost pro inspekční bod od těžiště trojúhelníku, který se snažíme pokrýt, je definována, jako taková vzdálenost, aby kamera ležící na kolmici k rovině trojúhelníku v maximální vzdálenosti od této roviny byla schopna svým rohovým pixelem (například [0,0]) nasnímat tuto rovinu stále ještě s požadovanou kvalitou. Jinak řečeno, rohový pixel kamery, která je nasměrována kolmo na snímanou rovinu, musí pokrýt maximální plochu takovou, aby byla stále splněna podmínka kvality snímku. Ilustrace je na obrázku 5.2. Tato vzdálenost se používá jako referenční vzdálenost pro vyhledávání inspekčních bodů. Je definována tímto způsobem, protože se snažíme v ideálním případě pokrýt z inspekčního bodu co největší množství trojúhelníků. Výpočet této vzdálenosti je uveden v kapitole 7.3.4.



Obrázek 5.2: Ilustrace výpočtu maximální vzdálenosti inspekčního bodu[2]
Pozn.: IB = inspekční bod, fialový vektor = vyslaný paprsek z pixelu [0,0], červená úsečka = hledaná maximální vzdálenost

5.4 Prvotní dělení trojúhelníků

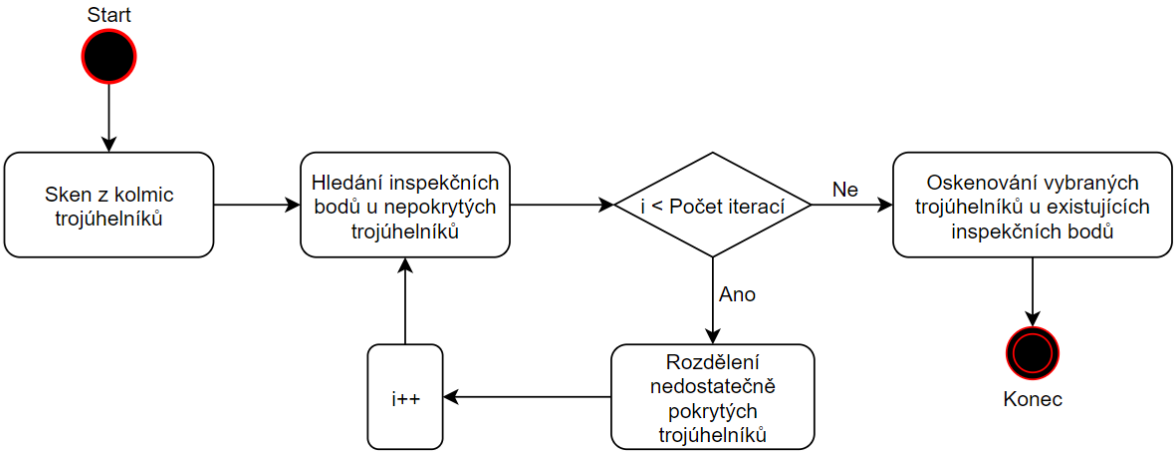
Dělení se provádí rozpůlením trojúhelníku přes jeho nejdelší stranu. Z původního trojúhelníku se vytvoří 2 nové, přidáním hrany jdoucí z bodu nacházejícího se v polovině jeho nejdelší hrany do protilehlého vrcholu (těžnice jdoucí z nejdelší strany). Na začátku algoritmu se rozdělí velké trojúhelníky, které by se nevešly na snímek. Rozdělené trojúhelníky se po rozdělení hned otestují na velikost a případně se dělí dále.

5.5 Hledání inspekčních bodů

Ideálního pokrytí modelu lze dosáhnout, pokud bude mít každý trojúhelník svůj vlastní inspekční bod, ze kterého bude pokryt. Směr kamery v inspekčním bodě je vždy vektor směřující z pozice inspekčního bodu do těžiště trojúhelníku, vůči kterému byl inspekční bod vytvořen. Jako základní pokrytí objektu přichází na řadu umístění inspekčního bodu na kolmici k těžišti trojúhelníku do vypočítané maximální vzdálenosti ve směru od objektu. Vzhledem k možné větší členitosti inspektovaného objektu se tímto způsobem pokryje jenom část trojúhelníků. Poté přichází na řadu vyhledávání inspekčních bodů pro nepokryté trojúhelníky, tento algoritmus se provádí v iteracích. Počet iterací je definován uživatelským vstupním parametrem. Mezi těmito iteracemi se provádí dělení trojúhelníků, které nebyly dostatečně pokryty ze žádného inspekčního bodu. Po dokončení vyhledávání inspekčních bodů dochází ke skenování konkrétních

[2]Obrázek vyroben pomocí aplikace `https://www.matheretter.de/geoservant/en/`

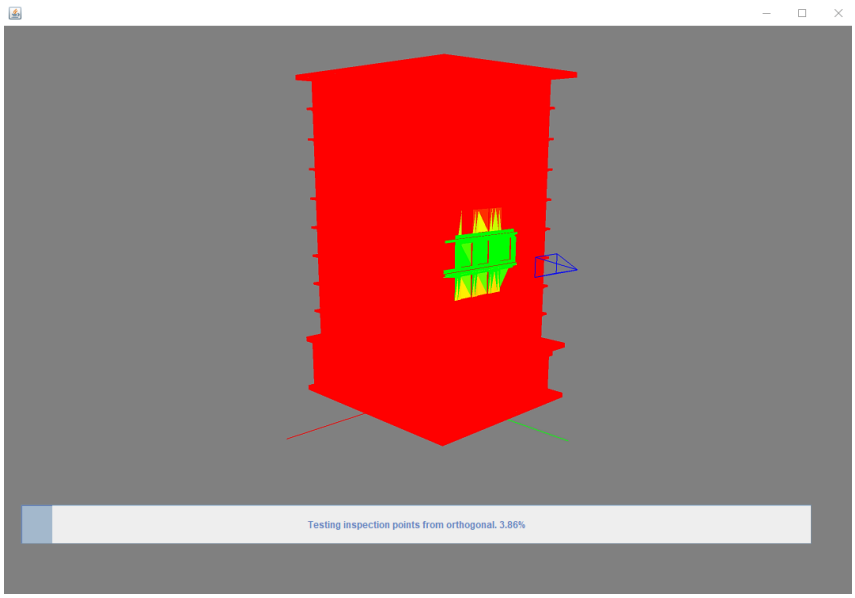
trojúhelníků v záběru již existujících inspekčních bodů.
Inspekční body se nevyhledávají pro příliš malé trojúhelníky. Hranice pro příliš malý trojúhelník je definována vstupním parametrem, který definuje maximální plochu pro tyto malé trojúhelníky. Jde o takové trojúhelníky, které nemá cenu explicitně skenovat, protože je jen malá šance, že je zasáhne dostatečné množství paprsků, aby skenování vrátilo relevantní hodnoty. Tento parametr může zároveň sloužit ke chtěné eliminaci skenování maličkých trojúhelníků (například lana visutého mostu), které zákazník nemusí požadovat. Parametr je volen na základě rozlišení kamery, FOV kamery a vstupního modelu. Diagram znázorňující vyhledávání inspekčních bodů ke konkrétním trojúhelníkům je na obrázku 5.3.



Obrázek 5.3: Diagram algoritmu vyhledávání inspekčních bodů

5.5.1 Základní pokrytí modelu z kolmic trojúhelníků

Jako první se ke každému trojúhelníku najde inspekční bod na kolmici k jeho těžišti. Tento bod se nachází v maximální vzdálenosti, vypočtené v části 7.3.4, ve směru normály trojúhelníku. Pokud se bod nenachází v kolizi s objektem nebo se zemí, tak se provede sken celé scény. Ukázka provedení takového skenu je na obrázku 5.4. Tímto způsobem se najde základní pokrytí modelu inspekčními body.

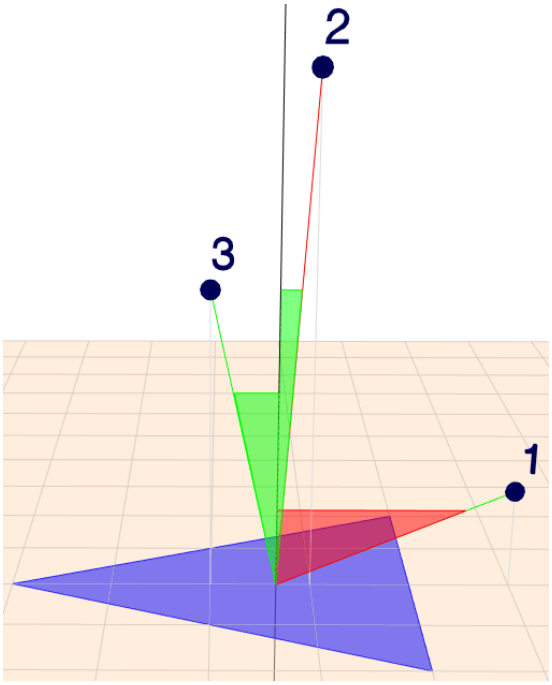


Obrázek 5.4: Ukázka skenu z kolmice v použitém frameworku

5.5.2 Vyhledávání inspekčních bodů u nepokrytých trojúhelníků

Pokud se provádí inspekce členitějšího objektu, tak základní sken z kolmic nebude dostačující. V takovém případě se musí najít algoritmus, který vzhledem ke skenovanému trojúhelníku najde co nejlepší pozici pro inspekční bod, ideálně takovou, aby z něj byl trojúhelník celý viditelný. Pro všechny nepokryté trojúhelníky (kromě příliš malých trojúhelníků) se zkusí zvolí nový výchozí adept na inspekční bod. Tento bod leží opět na kolmici k těžišti trojúhelníku, ale nyní ve vzdálenosti 70 % maximální vzdálenosti a zkusí se provést sken trojúhelníku. 70 % maximální vzdálenosti se zvolilo proto, protože algoritmus vyhledávající lepší pozici pro inspekční bod se pohybuje v rovině, která je paralelní s rovinou trojúhelníku. Pokud by se zvolil výchozí bod v maximální vzdálenosti, tak všechny ostatní body, které byalgoritmus našel, by měly větší vzdálenost k těžišti trojúhelníku, než je maximální povolená vzdálenost a algoritmus by se ukončil. Nyní může nastat několik situací:

- adept na inspekční bod patřící k tomuto trojúhelníku je v kolizi s objektem nebo zemí;
- trojúhelník není vidět kvůli překážce;
- trojúhelník je vidět jen částečně;
- trojúhelník je pokryt;

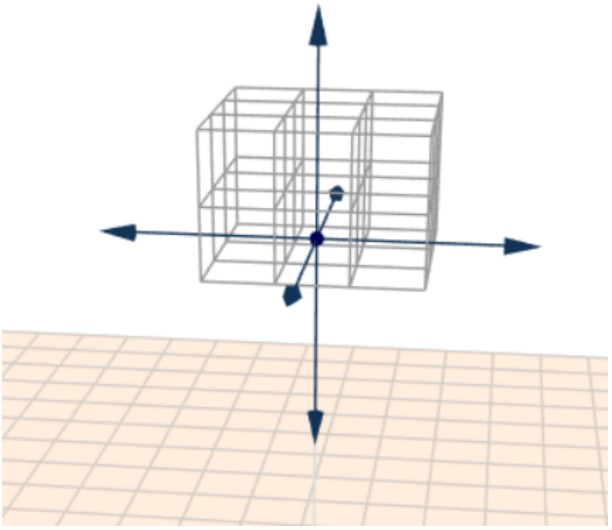


Obrázek 5.5: Ukázka výběru vhodného bodu před provedením skenu
1 = směr kamery v bodě 1 leží pod moc velkým úhlem k trojúhelníku a snímek by nesplnil požadavek kvality, 2 = bod je moc daleko od trojúhelníku, 3 = vhodný bod pro sken

První 2 body mohou přecházet v problém částečně viditelného trojúhelníku.
Jako výchozí bod pro vyhledávání inspekčních bodů se bere bod na kolmici ve vzdálenosti 70-ti procent maximální vzdálenosti V případě, že trojúhelník je celý viditelný, se provede sken scény a pokračuje se dalším trojúhelníkem.
Před provedením skenu trojúhelníku se nově nalezený adept na inspekční bod otestuje, zda má význam provést tento sken, to znamená, že bod nesmí být vzdálený od těžiště trojúhelníků více než je maximální vzdálenost a zároveň se zkusí, jestli z tohoto bodu splňují vrcholy zkoumaného trojúhelníku požadavek na kvalitu snímku. Ukázka vhodně vybraného bodu je na obrázku 5.5.

Výchozí inspekční bod je v kolizi

Že se výchozí bod nachází v kolizi znamená, že buď je v objektu nebo blízko něj anebo se bod nachází v kolizi se zemí. V případě, že se nachází v kolizi se zemí, se tento bod pouze přesune do bezpečné vzdálenosti nad úroveň země. V případě, že se bod nachází v kolizi s objektem, musíme najít takové místo, aby inspekční bod pokryl chtěný trojúhelník a nebyl v kolizi s objektem.
Z voxelu, ve kterém se nachází počáteční adept na inspekční bod se prochází voxely v obou směrech všech os souřadnicového systému (obrázek 5.6), pokud se najde prázdný voxel, tak se souřadnice odpovídající středu tohoto voxelu přidá do seznamu bodů na vyzkoušení. V případě, že jde o krajní voxel, se do seznamu přidají body ležící ve směrech procházení voxelů, ale jsou již mimo voxely. V každém směru se do seznamu přidají 4 body v různých vzdálenostech od modelu.



Obrázek 5.6: Směry procházení voxelů v případě bodu v kolizi

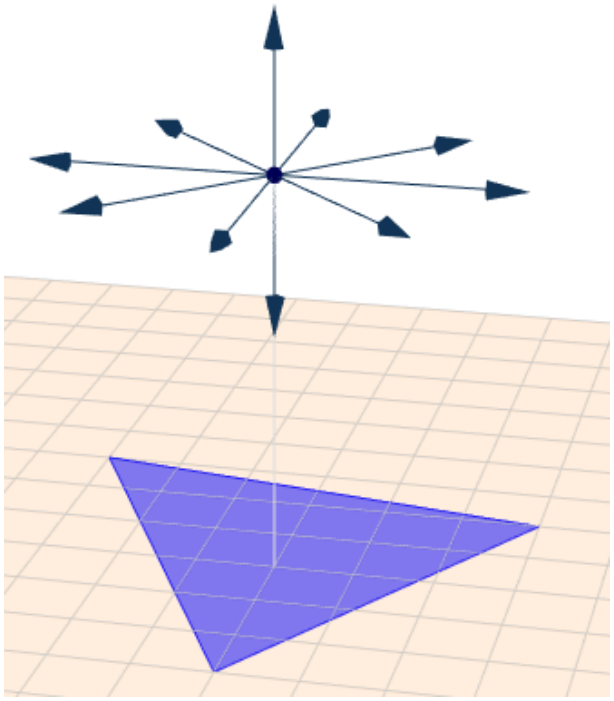
Po vyhledání bodů, které se nenachází v kolizi dojde k vyfiltrování nevhodných bodů, to znamená bodů, které jsou moc daleko od trojúhelníku a bodů, které jsou pod velkým úhlem a sken by nevyústil v požadovanou kvalitu. Vzdálenost se porovnává s maximální napočítanou vzdáleností. Zda trojúhelník z daného bodu splňí požadavek na kvalitu snímku se zjistí vypočtením ploch, které by pokryly pixely pokrývající vrcholy trojúhelníku a porovnáním s maximální povolenou plochou, kterou může pixel pokrýt. Body, které projdou filtrem se otestují na potenciální inspekční body tak, že se provede z jejich pozice ve směru trojúhelníku sken pouze hledaného trojúhelníku. Pokud se nenajde vhodný bod, tak se zmenší vzdálenost původního výchozího bodu k trojúhelníku a začne se znovu. Takto se pokračuje, dokud se nenajde vhodný bod, to znamená bod, ze kterého je trojúhelník zasažen alespoň jedním paprskem. Pokud je trojúhelník vidět provede se sken scény. V případě, že není celý vidět, změní se problém na nějaký z níže popsaných.

Hledání vhodného bodu zmenšováním vzdálenosti původního adeptu na inspekční bod a trojúhelníku je omezeno uživatelským vstupním parametrem minimální vzdálenosti inspekčního bodu k trojúhelníku. Tento parametr je z důvodu použití čočky na kameře, která má určitou minimální vzdálenost zaostření.

Trojúhelník není vidět

Že trojúhelník není vidět znamená, že dosud nebyl ani částečně pokrytý žádným inspekčním bodem. V tomto případě se hledají nové body v rovinách paralelních s rovinou trojúhelníku. K pohybu bodem v prostoru jsem napočítal 8 vektorů (diskretizace prostoru), které leží v paralelní rovině s rovinou trojúhelníku a sousední vektory mezi sebou svírají úhel 45 stupňů. Vektory mají počáteční bod společný (výchozí bod ležící na kolmici). K těmto vektorům jsem ještě přidal 2 další, a to vektor směřující kolmo k rovině trojúhelníku a kolmo od roviny trojúhelníku. Vizualizace je vidět na obrázku 5.7.

Algoritmus provádí sken trojúhelníku z bodů, které leží ve směrech vektorů v různých vzdálenostech. Jinak řečeno, zkouší se body ležící na kružnicích s různým poloměrem a se středem ve výchozím bodě. Maximální zkoušený poloměr této kružnice je definován uživatelským parametrem stejně jako krok zvětšení tohoto poloměru. Pokud se najde nějaký bod, ze kterého je

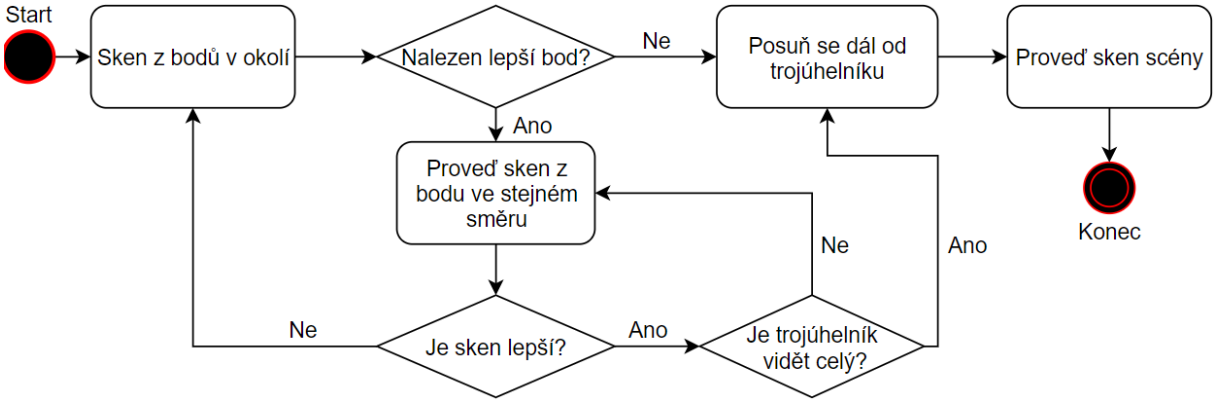


Obrázek 5.7: Směry použité pro vyhledávání inspekčního bodu

trojúhelník zasažen paprskem, změní se problém na problém částečně viditelného trojúhelníku. V případě, že se takový bod nenajde, tak obdobně jako u řešení bodu v kolizi, se zmenší vzdálenost původního výchozího bodu k trojúhelníku a hledání se opakuje.

Částečně viditelný trojúhelník

Pokud se algoritmus dostane do stavu, že už z nějakého bodu je trojúhelník alespoň částečně viditelný, zmenší se prostor, který se prohledává pro nalezení inspekčního bodu. V prvním kroku se zkusí provést sken z bodů v okolních bodech kolem výchozího bodu, ležících ve směrech stejných jako v případě zakrytého trojúhelníku (viz.: obrázek 5.7). Vzdálenost, ve které se nachází tyto nové body od původního je definována vstupním parametrem. Z nových bodů se vybere ten, který má nejlepší sken trojúhelníku. Pokud žádný takový nový bod není, algoritmus se ukončí a udělá se sken scény z bodu, který poskytuje aktuálně nejlepší sken trojúhelníku. Sken scény se provede jen v případě, že hledaný trojúhelník je z aktuálního bodu naskenovaný aspoň z předem definované části. Tato část je definovaná uživatelským parametrem.
Pokud se ale našla v okolí výchozího bodu lepší pozice, tak se tato nová pozice vezme jako nový výchozí bod. V dalším kroku se udělá sken z bodu ležícího ve stejném směru, jako byl nalezený nový výchozí bod vzhledem k předchozímu. Takto se pokračuje, dokud není trojúhelník celý viditelný nebo dokud se v daném směru nezhorší pokrytí trojúhelníku. Pokud se zhorší pokrytí trojúhelníku ve zkoumaném směru, tak se provede sken ze všech okolních bodů a pokračuje se v algoritmu dále. Algoritmus končí až když se najde bod, ze kterého je trojúhelník pokrytý nebo se nenajde v okolí posledního bodu s nejlepším pokrytím žádný lepší bod. Jedná se o variaci gradientního algoritmu. Pokud je trojúhelník vidět celý, zkusí se ještě s bodem posouvat směrem od těžiště trojúhelníku, aby se se pokrylo co nejvíce prostoru. Posouvá se dozadu, do té doby, dokud je trojúhelník z nových bodů stále vidět s požadovanou kvalitou a nepřekročí se maximální vzdálenost od těžiště trojúhelníku nebo dokud se nenarazí na plný voxel. Poté se provede sken scény z nově nalezeného inspekčního bodu. Popsaný algoritmus ve formě diagramu je na obázku 5.8.



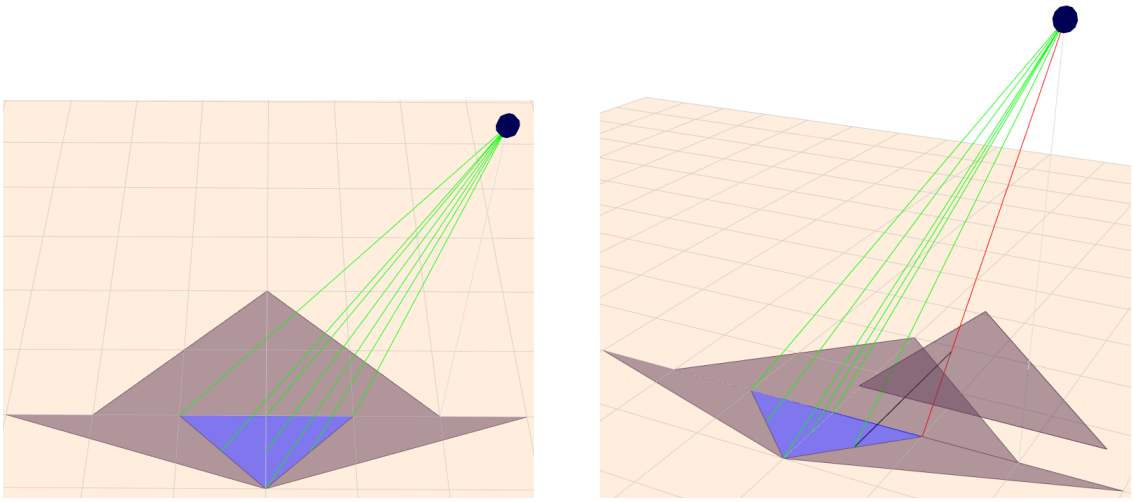
Obrázek 5.8: Diagram běhu algoritmu pro nalezení inspekčního bodu u částečně viditlných trojúhelníků

Dělení neodstatečně pokrytých trojúhelníků

Mezi iteracemi probíhá dělení nedostatečně pokrytých trojúhelníků. Dělí se trojúhelníky, pro které existuje nějaký inspekční bod, ze kterého jsou vidět alepoň z předem definované části, ale neexistuje žádný inspekční bod, který by trojúhelník kompletně pokryl. Z kolika procent mají být trojúhelníky pokryty, aby se dělily definuje další uživatelský vstupní parametr. V případě, že trojúhelník neprošel dělením, ale prošel hledáním na inspekční body, se už nepřidá do seznamu trojúhelníků, které projdou další iterací vyhledávání inspekčních bodů.

5.5.3 Sken vybraných trojúhelníků z inspekčních bodů

V této části jsou známé všechny trojúhelníky, které se podařilo pokrýt inspekčními body z předchozích postupů. Jenže tyto inspekční body stále mohou obsahovat záznamy o trojúhelnících, které byly rozděleny v pozdějším běhu algoritmu. Z tohoto důvodu se u všech inspekčních bodů najdou trojúhelníky, které byly někdy rozděleny a udělá se sken z daného inspekčního bodu obou nových trojúhelníků (případě více, pokud byl trojúhelník dělen vícekrát). Zároveň se u inspekčních bodů vyberou malé trojúhelníky, které byly zasaženy nějakým paprskem z daného inspekčního bodu a zkusí se provést sken těchto malých trojúhelníků.Výsledky těchto skenů se zaznamenají zpátky do inspekčního bodu, ze kterého byl sken prováděn. Jelikož u malých trojúhelníků je menší šance, že je zasáhne dostatečný počet paprsků, tak se zkusí vyslat paprsky na hrany malého trojúhelníku a jeho těžiště s body okolo těžiště. Zaznamená se, jaké trojúhelníky byly zasaženy a vzdálenosti od kamery k zasaženým bodům. Nyní se porovnají nejbližší vzdálenosti zásahů různých trojúhelníků, pokud se nezasáhne nějaký jiný trojúhelník ve vzdálenosti bližší, než je vzdálenost skenovaného trojúhelníku - (0,1 * měřítko modelu), tak se trojúhelník označí za viditelný. Jelikož jde pouze o malé trojúhelníky, tak se minimalizuje šance toho, že by byl trojúhelník zakrytý a při porovnávání vzdáleností se na to nepřišlo. Zjednodušená vizualizace skenování malých trojúhelníků se nachází na obrázku 5.9.



(a) pokrytý (b) nepokrytý

Obrázek 5.9: Zjednodušená vizualizace sknování malých trojúhelníků
Pozn.: černá úsečka = porovnávaná vzdálenost mezi zasaženými trojúhelníky

5.6 Výběr inspekčních bodů

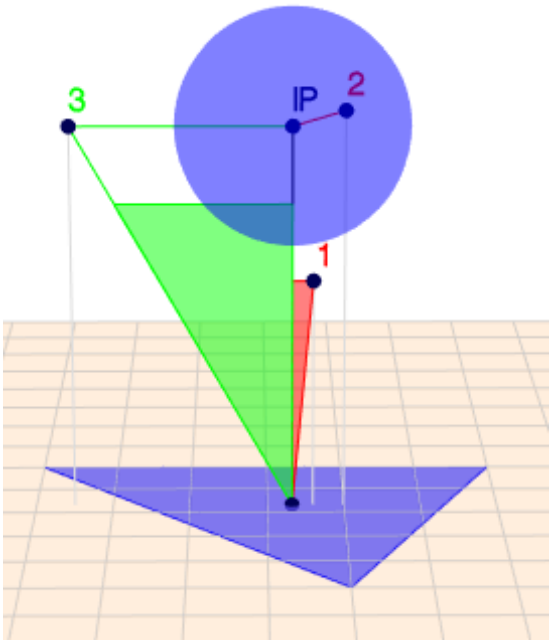
Nyní přichází na řadu výběr nejvhodnějších inspekčních bodů. Protože inspekčních bodů může být nalezeno tisíce až desetitisíce, musí se jejich počet zredukovat. Mnoho inspekčních bodů pokrývá stejnou část modelu a obsahují velmi podobné pokrytí trojúhelníků, proto je možné z takovýchto bodů vybrat jen určité množství, zároveň výsledná cesta nebude tak dlouhá a zkrátí se čas letu dronu, tím pádem dron nebude potřebovat tak velké baterie, popřípadě během inspekce dobíjet. Výběr nejvhodnější množiny inspekčních bodů je ve své podstatě problém pokrytí minimální množinou.
Pokrytí minimální množinou (anglicky Set cover problem) se řadí mezi NP-úplné problémy. Jde o úlohu, ve které máme množinu $A = \{1,2,...,n\}$ a seznam S skládající se z m množin, jejichž sjednocení se rovná množině A. Nyní se hladá taková nejmenší podmnožina z kolekce S, aby se její průnik stále rovnal množině A. Tento problém se řeší buď hladovým (greedy) algoritmem a jeho modifikacemi v podobě lokálního vyhledávání lepších výsledků (local search) nebo přes ILP [19].

5.6.1 Algoritmus výběru inspekčních bodů s úpravou pro splnění redundance

V případě tohoto řešení se musí brát ohled na zákaznický požadavek redundance snímků. To znamená, že každý trojúhelník musí být nasnímaný alespoň z určitého počtu inspekčních bodů. Použil jsem klasický hladový algoritmus s úpravou pro splnění redundance. Algoritmus běží ve smyčce. V každé iteraci najde inspekční bod, který pokrývá co nejvíce trojúhelníků, které ještě nebyly označené, jako pokryté. U nalezeného inspekčního bodu se projdou všechny viditelné trojúhelníky a zkontroluje se, jestli u nich bod splňuje podmínky na vzdálenost a úhel od ostatních inspekčních bodů, které již byly vybrány v předchozích iteracích a pokrývají iterovaný trojúhelník. Pokud jsou podmínky splněny, tak se u trojúhelníku zapamatuje inspekční bod a zároveň se přidá do výsledné množiny inspekčních bodů. Nyní se projdou všechny inspekční body a pokud se některé nachází v blízkosti vybraného, odeberou se z jejo množiny pokrytých trojúhelníků ty trojúhelníky, které vybraný inspekční bod pokrývá. V případě, že už trojúhelník má v záznamu požadovaný počet inspekčních bodů, tak se označí, jako pokrytý. Takto algoritmus pokračuje do té doby, dokud neexistuje další inspekční bod, který by pokryl další trojúhelníky, nebo dokud nejsou všechny trojúhelníky pokryté.

5.7 Hledání inspekčních bodů pro splnění redundance snímků

Teď už pouze chybí najít další inspekční body u trojúhelníků, které nemají splněný požadavek na redundanci snímků. Trojúhelníky v této fázi algoritmu již musí být pokryty aspoň z jednoho inspekčního bodu. Takže pokud chceme najít nějaký další inspekční bod, který pokrývá stejný trojúhelník, má cenu vyhledávat pouze v okolí již existujících inspekčních bodů. Body se hledají v rovinách paralelních s rovinou trojúhelníku, jako při hledání bodu u trojúhelníku, který je zakrytý překážkou (5.5.2). Opět se zkouší hledat ve stejných směrech, jako u zakrytých trojúhelníků s tou změnou, že se nezkouší vektory směrem do a od roviny trojúhelníku. Vyhledává se opět po kružnicích o různých poloměrech, střed kružnice je vždy na přímce, která prochází inspekčním bodem a těžištěm trojúhelníku. Tímto způsobem se zkusí hledat v okolí všech inspekčních bodů, které byly vybrány pro daný trojúhelník. Poté se zkusí provést sken z těchto pozic. Pokud je trojúhelník celý viditelný a splňuje požadovanou kvalitu snímku, tak byl nalezen nový inspekční bod. Nicméně nové inspekční body musí splňovat podmínku minimální vzdálenosti a minimálního úhlu, který svírají přímky jdoucí z těžiště trojúhelníku do inspekčního bodu a z těžiště trojúhelníku do testovaného bodu. Tyto podmínky jsou zavedeny z důvodu požadovaného překryvu snímků, pokud by se zvolily jakékoliv body, které by pokrývaly stejný trojúhelník, mohlo by se stát, že by tyto body byly těsně vedle sebe nebo třeba na stejné přímce. V takovém případě by výsledkem byl téměř totožný snímek. Podmínka minimální vzdálenosti určuje, jakou minimální vzdálenost mezi sebou musí mít 2 inspekční body pokrývající stejný trojúhelník. Podmínka úhlu určuje jaký má být minimální úhel mezi přímkou procházející vybraným inspekčním bodem a těžištěm trojúhelníku a přímkou procházející testovaným inspekčním bodem a těžištěm trojúhelníku.



Obrázek 5.10: Výběr bodů pro splnění podmínky redundance snímků
Pozn.: IP = vybraný inspekční bod, 1 = bod nesplňující podmínku úhlu mezi inspekčními body, 2 = bod nesplňující podmínku vzdálenosti mezi inspekčními body, 3 = nový inspekční bod

Algoritmus se ukončí, pokud je splněna redundance snímků u trojúhelníku nebo pokud nebyl nalezen žádný nový inspekční bod. Ukázka výběru vhodného inspekčního bodu je na obrázku 5.10.

Kapitola 6

Hledání trajektorie pro UAV

Nyní se musí najít cesta procházející vybranými inspekčními body, kterou UAV proletí a bude v těchto bodech pořizovat snímky. Vytvoření takovéto cesty je ukázkový případ TSP. Ještě předtím, než je možné použít jakýkoliv známý algoritmus řešící TSP, je potřeba najít nejkratší cesty mezi jednotlivými inspekčním body.

6.1 Hledání nejkratší cesty

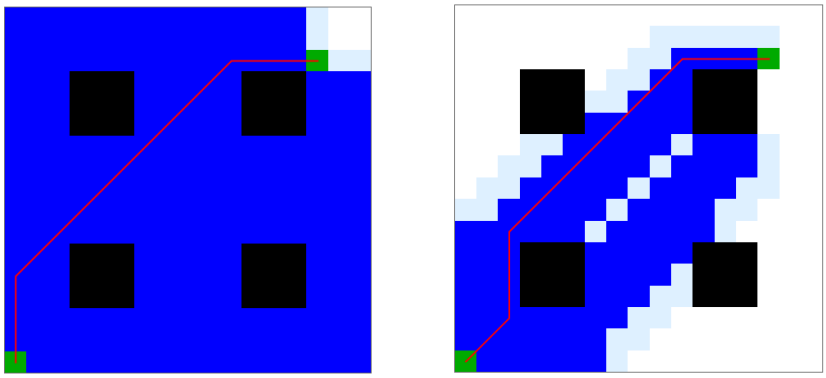
Hledání nejkratší cesty je běžný problém, který se řeší v Teorii grafů. Jde o úlohu, kde se snažíme najít nejkratší cestu mezi dvěma vrcholy grafu. Na tento problém existuje několik algoritmů, které řeší neohodnocené, ohodnocené, záporně nebo jenom kladně ohodnocené hrany grafu [20]. V případě této práce jde pouze o kladně ohodnocené hrany grafu. Na tento případ lze použít tyto algoritmy: Dijkstrův algoritmus nebo A*.
Oba tyto algoritmy pracují s frontou, do které se přidávají vrcholy grafu, ze kterých se bude v algoritmu postupovat dále a hledat cesta do cílového vrcholu a množinou již navštívených vrcholů, které se znovu do fronty už nepřidávají. Oba algoritmy začínají tak, že se do fronty přidá počáteční vrchol.

6.1.1 Dijkstrův algoritmus

Dijkstrův algoritmus už pracuje s váhami hran a volí vhodněji vrcholy, ze kterých se bude dále postupovat. Na rozdíl od BFS používá prioritní frontu, kde jsou vrcholy seřazené podle jejich vzdálenosti od startu. Takže v každém dalším cyklu se prohledává od vrcholu, ze který je nejblíže ke startu. Jinak funguje stejně jako BFS a algoritmus končí, až se narazí na koncový vrchol.

6.1.2 A*

A* je algoritmus, který slouží primárně k hledání nejkratší cesty mezi dvěma vrcholy v kladně ohodnoceném grafu. Je to rozšíření Dijsktra algoritmu o heuristiku. Stejně jako u Dijkstrova algoritmu se používá prioritní fronta, ta však není seřazena podle vzdálenosti vrcholů od počátečního, ale pořadí vrcholů je definováno funkcí $f(x) = g(x) + h(x)$ kde $g(x)$ je vzdálenost vrcholu od startu a $h(x)$ je heuristická funkce. Heuristickou funkcí je odhad vzdálenosti aktuálního vrcholu do konečného. Ve 3D prostoru může jít o euklidovskou vzdálenost mezi těmito body. A* je vhodnější pro úlohy s překážkami, protože při vhodně zvolené heuristice neprohledává zbytečné cesty a vždy najde nejkratší cestu (pokud taková cesta existuje). Vizualizace běhu algoritmu A* a Dijkstrova algoritmu je na obrázku 6.1.



(a) Dijkstrův algoritmus (b) A*

Obrázek 6.1: Porovnání výsledku Dijkstrova algoritmu s A* algoritmem [1]
Pozn.: modrá = navštívené buňky, černá = překážka, bílá = nenavštívené buňky, červená = výsledná cesta, světle modrá = buňky ve frontě.

Výpočet eulidovské vydálenosti d mezi body A a B:

$$d = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2}$$

6.2 Problém obchodního cestujícího

Problém obchodního cestujícího (TSP) patří mezi NP-těžké problémy. Jde o úlohu, ve které se snažíme mezi vrcholy ohodnoceného úplného grafu sestrojit co nejkratší Hamiltonovskou kružnici [21]. Hamiltonovská kružnice znamená, že se najde v grafu taková cesta, aby byl každý vrchol navštívený právě jednou a začíná ve stejném bodě, ve kterém končí (cesta je uzavřená). Existuje několik postupů řešící tento problém [22].

6.2.1 Přesné řešení

Na získání přesného řešení co nejkratší Hamiltonovské kružnice existuje několik přístupů. Bohužel vzhledem k výpočetní náročnosti se tyto postupy hodí pouze na malou sadu dat (desítky až nižší stovky vrcholů grafu).

- **Hrubá síla** - jde o vygenerování všech permutací cest a vybrání té nejlepší. Tento přístup se stává příliš pomalým už v případě nižších desítek vrcholů grafu.
- **Metoda větví a mezí** - jde o řešení pracující s grafem jako by to byl strom. Prohledávají se jednotlivé větve a zapamatovává nejlepší výsledek. Pokud je během procházení stromu v nějakém uzlu překročena hodnota aktuálně nejlepšího výsledku, v prohledávání se dále v této větvi nepokračuje.
- **ILP** - jde o způsob řešení optimalizačních úloh, kde se hledají minima nebo maxima proměnných lineární funkce, která je popsaná soustavou lineárních nerovnic.

Problém těchto přístupů je v tom, že se zvětšujícím se grafem (tisíce vrcholů) se tyto algoritmy stávají příliš pomalými.

[1] Obrázky byly pořízeny z aplikace stažené z `https://demonstrations.wolfram.com/DijkstrasAndASearchAlgorithmsForPathfindingWithObstacles/`

6.2.2 Heuristiky a aproximační řešení

Pro větší grafy se používají heuristiky a aproximační algoritmy, které se k optimálnímu výsledku přibližují. Pro použití některých aproximačních algoritmů již musíme znát nějaké řešení, které se dále vylepšuje. Postupy, které najdou řešení jsou například:

- **Náhodné řešení** - vrcholy se náhodně seřadí.
- **Nejližší soused** - jde o metodu, která do seznamu vrcholů přidává vždy nejbližší nenavštívený vrchol k poslednímu v seznamu.
- **Christofidesův algoritmus** - jde o 3/2 aproximační algoritmus, který zjistí cestu za použití minimální kostry grafu.

2-OPT algoritmus

Jde o algoritmus, který vylepšuje již známe řešení. 2-OPT je 2 aproximační algoritmus, to znamená že výsledek tohoto algoritmu bude nanejvýš 2krát horší než optimum. Tento algoritmus hledá nejkratší cestu tím způsobem, že v existující cestě vymění 2 hrany. Tyto hrany nesmí v původní cestě následovat za sebou. Vybere lepší z dvou cest a pokračuje stejným způsobem dál [23]. Algoritmus tedy hledá lokální minima. Jelikož není určen konec běhu algoritmu, tak se musí zvolit způsob ukončení, buď algoritmus omezit na počet iterací nebo časově.

Kapitola 7

Implementace

V této kapitole si popíšeme implementační část, použité technologie a knihovny.
Jelikož byl k implementaci použit AgentFly framework, který je postavený na jazyce Java, tak i k implementaci algoritmu byl použit programovací jazyk Java ve verzi 8. K vývoji bylo použito vývojové prostředí IntelliJ IDEA.

7.1 Práce s frameworkem

Abych mohl využít framework k práci s modelem a kamerou, je potřeba nainicializovat framework a předat mu trojúhelníkový model. K načtení modelu slouží třída *LeafGroup*, která načte model do třídy *Mesh*, z této třídy lze dostat všechny trojúhelníky s jejich vrcholy, ze kterých se počítají normály, plocha a těžiště trojúhelníků. Načtený model se přes *LayerProvider* předá frameworku. Po dokončení inicializace lze s modelem dále pracovat přes posluchače událostí a například ho měnit, vykreslovat do prostoru další tvary atd.

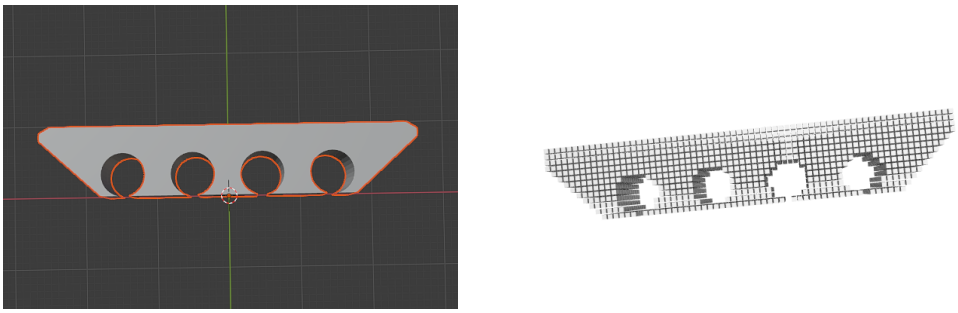
7.2 Řešení kolizí

Hledání kolizí se řeší voxelizací polygonového modelu. Voxelizovaný soubor je jedním ze vstupů programu. Musí být ve formátu JSON[1] a mít následující strukturu:

```
{
 "dimension": [
   {
    "width": "74",
    "height": "19",
    "depth": "13"
   }
 ],
 "voxels": [
   { "id": "voxel_0", "x": "11", "y": "0", "z": "0" },
   { "id": "voxel_1", "x": "12", "y": "0", "z": "0" },
   { "id": "voxel_2", "x": "13", "y": "0", "z": "0" },
   { "id": "voxel_3", "x": "14", "y": "0", "z": "0" },
   { "id": "voxel_4", "x": "15", "y": "0", "z": "0" },
   ...
   ]
}
```

[1] https://www.w3schools.com/whatis/whatis_json.asp

Kde *width* reprezentuje hodnoty na ose x, *height* na ose y a *depth* na ose z.
K tomuto účelu jsem použil online voxelizátor[2]. Voxelizátor přijímá jako vstup soubory formátu .obj. Uživatel si zde zvolí výstupní formát voxelizovaného modelu (pro naše účely JSON) a může zde nastavit kvalitu voxelizace a tloušťku stěn modelu. Kvalitu voxelizace je vhodné nastavit tak, aby voxelizovaný model (který se na stránce zobrazí v reálném čase, kdykoliv se změní nějaký vstupní parametr voxelizace) odpovídal tvarově co nejvěrněji trojúhelníkovému modelu, ale zároveň, aby výsledné dimenze voxelizovaného modelu nebyly moc velké (například 100 a více voxelů v jedné dimenzi). Důvod k tomu je, že nad tímto modelem se provádí velice často vyhledávání konkrétního voxelu a zároveň se tento model používá u hledání nejkratší cesty mezi inspekčními body. Proto je příhodné, aby prohledávaný prostor byl co nejmenší v závislosti na zachování kvality modelu. Ukázka voxelizovaného modelu v porovnání s polygonálním modelem je na obrázku 7.1.



(a) Mesh (b) Voxely

Obrázek 7.1: Porovnání modelu objektu v mesh a voxelech

Zvětšením nastavení tloušťky stěny se voxelizovaný model začne postupně vyplňovat plnými voxely. V případě této práce je zapotřebí, aby byly voxely uvnitř modelu označeny jako plné, takže je potřeba nastavit tento parametr na nejvyšší hodnotu. K načtení voxelizovaného modelu byla pro Javu použita knihovna JSON-simple[3].
Po načtení voxelizovaného modelu do 3D pole booleanů se musí zjistit, jaký prostor tyto voxely reprezentují. K tomu se použije trojúhelníkový model, ve kterém se najdou minimální a maximální souřadnice vrcholů na všech osách. Jelikož voxelizovaný model je pouze jinak reprezentovaný trojúhelníkový model, tak můžeme říct, že voxely se budou nacházet v prostoru kvádru, který je mezi těmito minimálními a maximálními souřadnicemi. Nyní se ještě musí zjistit velikost voxelů v různých dimenzích, k čemuž se použijí minimální a maximální souřadnice modelu a 3D pole s voxely.

$$x = (|min_x| + |max_x|)\,/s$$
$$y = (|min_y| + |max_y|)\,/v$$
$$z = (|min_z| + |max_z|)\,/h$$

Kde s je počet voxelů v modelu na šířku, v na výšku a h na hloubku.
Celá voxelizace má za úkol usnadnit zjišťování kolize UAV s objektem, z toho důvodu se nyní provede rozšíření plných voxelů o bezpečnou vzdálenost. Bezpečná vzdálenost je jeden z uživatelských parametrů, který říká, jaká má být minimální vzdálenost dronu od objektu. V implementaci se neuvažuje střed dronu, ale pozice kamery, tudíž je na uživateli, aby nastavil tuto vzdálenost podle použitého dronu. Rozšíření plných voxelů má za důsledek, že když vezmeme bod, který se nachází v prázdném voxelu, tak nemusíme kontrolovat okolní voxely, vždy máme jistotu, že nedojde ke kolizi. Pokud bychom brali v úvahu případ, kdy by se plné voxely

[2] `https://drububu.com/miscellaneous/voxelizer/?out=jso`
[3] `https://code.google.com/archive/p/json-simple/`

nerozšířily, tak i v případě, že se inspekční bod nachází v místě prázdného voxelu, tak se musí kontrolovat okolní voxely, jestli jsou také prázdné, a to v případě 3D znamená v každé pozici kontrolovat v nejhorším případě 26 okolních voxelů.
Rozšíření plných voxelů se provádí tak, že se prvně v každé ose zjistí, kolik voxelů k daném směru přidat.

$$plus\_x = \lceil d/x \rceil$$
$$plus\_y = \lceil d/y \rceil$$
$$plus\_z = \lceil d/z \rceil$$

Kde d je bezpečná vzdálenost, a x, y a z jsou vypočítané rozměry voxelu.
Poté se vytvoří nové 3D pole, které má dimenze zvětšné v každém směru o dvojnásobek vypočtených hodnot (dvojnásobek, protože musíme přidat bezpečnou vzdálenost na obě strany modelu). Nyní se nové pole vyplní původními hodnotami, a navíc se přidá plný voxel všude, kde se v okolních buňkách (podle počtu přidaných buněk v daných směrech) původně vyskytoval plný voxel. Adekvátně se vzhledem ke zvětšenému 3D poli s voxely změní i souřadnice okrajů voxelizovaného modelu, které byly původně minimální a maximální souřadnice vrcholů trojúhelníků z trojúhelníkového modelu.

7.2.1 Zjištění kolize

Samotné zjištění, zda se bod v prostoru nachází v prázdném voxelu je nyní jednoduché. Souřadnice bodu se skládají ze tří složek [x, y, z], pokud se tento bod nachází mimo voxely, tak to znamená, že k žádné kolizi nemůže dojít. To, že se bod nachází v nějakém voxelu se zjistí tak, že každá složka souřadnice se nachází mezi okrajovými souřadnicemi voxelů na korespondující ose. Pokud se bod nachází uvnitř voxelů, tak se musí zjistit, na jaké pozici. Zjistí se to následujícím algoritmem:

```
 while (x <= voxels.length) {
        float pos = bounds.minX + x * xStep;
        \\ xStep je rozměr voxelu v dané ose
        if (pos >= position.x) {
             break;
        }
        x++;
    }
x--;
```

Tento postup se zopakuje pro všechny složky souřadnice a výsledkem je pozice voxelu ve 3D poli, ve kterém se nachází aktuální bod. Pokud je voxel označený jako plný (true), tak by se bod nacházel v místě potenciální kolize UAV s modelem.

7.3 Skenování trojúhelníků

Skenování trojúhelníků je jedna z nejdůležitějších funkcí, protože se zde zjistí, jaké trojúhelníky jsou vidět z dané pozice s konkrétním nastavením kamery. A zároveň se kontroluje, s jakou kvalitou by byl trojúhelník nasnímán. K tomuto jsem využil frameworku ve kterém jsem pracoval. Jak již bylo řečeno, tento framework obsahuje implementaci práce s kamerou a paprsky vysílanými z kamery.

7.3.1 Nastavení kamery

Implementace kamery (třída *CameraSensor*) vyžaduje povinné vstupní parametry, aby se dala využít k účelům skenování prostoru před kamerou. Jsou to:

- Rozlišení kamery
- FOV v radiánech
- Up vektor kamery
- KDTree objekt
- Vzdálenost rovin vykreslování (Clipping plane)

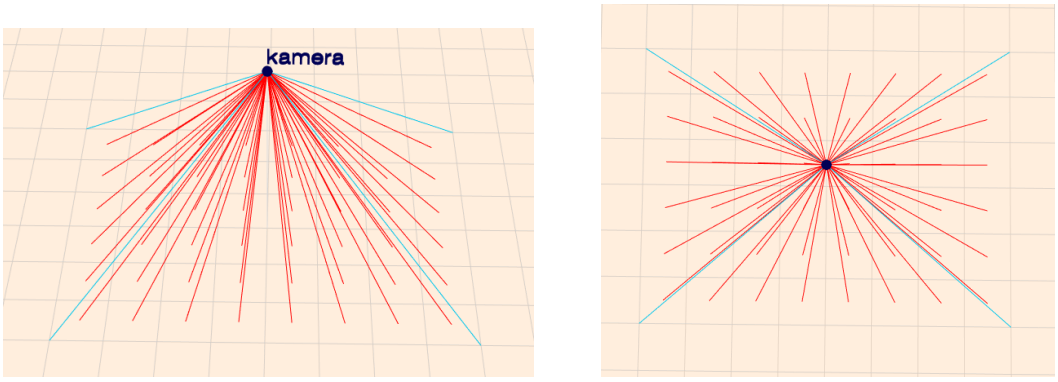
Rozlišení, vertikální FOV a Up vektor jsou uživatelské vstupní parametry, tyto parametry se nastaví podle použitého snímače na dronu, který bude provádět inspekci. Up vektor je parametr, který ovlivňuje natočení kamery. Tento vektor specifikuje, kterým směrem je ve scéně nahoru. V případě této práce byl vždy použit vektor (0,0,1). KDTree objekt je instance třídy *KDTree*, která je naimplementována v použitém frameworku. Před inicializací objektu kamery je potřeba vytvořit objekt KDTree, do kterého se jako vstupní parametr předají trojúhelníky ze modelu.

7.3.2 Skenování

Ke skenování scény kamerou jsem použil metodu *fullScan* implementovanou ve třídě *CameraSensor*. Před použitím této metody je potřeba nastavit kameře vektory pozice a směru kamery. Pak je už možné použít metodu *fullScan* s předpisem

```
public void fullScan(KDTree target, RayCastAction a),
```

kde *target* je instance třídy *KDTree*, kterou jsme již předali samotné kameře, takže se jí nemusí předávat znovu. A *a* je vložená metoda, která se zavolá s každým vyslaným paprskem.
Samotný sken probíhá vysíláním paprsků z pixelů kamery. Vizualizace vysílání paprsků lze vidět na obrázku 7.2. Metoda *fullScan* vyšle paprsky z každého pixelu, a zaznamená zasažený trojúhelník a v jaké vzdálenosti od kamery se tento trojúhelník nachází. Tyto informace se předají do vložené funkce *a*, která tyto data zpracovává.



(a) Z boku (b) Ze shora

Obrázek 7.2: Vysílané paprsky z jednotlivých pixelů kamery
Pozn.: červená = vyslané paprsky z kamery, modrá = ohraničení prostoru viditelného z kamery

Zpracování dat z vyslaných paprsků

Nejdřív se musí vypočítat, jakou plochu trojúhelníku pixel, reprezentovaný vyslaným paprskem, pokryl. K tomuto výpočtu se použije FOV a rozlišení kamery. Nejdříve je nutno zjistit, jaký je úhel záběru jednoho pixelu, nazvěme ho úhel γ . To se zjistí jednoduše vydělením FOV počtem pixelů. Při použití vertikálního FOV použijeme jako dělitel počet pixelů z rozlišení na výšku.

$$\gamma = \frac{FOV_{vert}}{y}$$

Kde y je počet pixelů na výšku. Teď lze za použití výpočtu v pravoúhlém trojúhelníku dopočítat plochu S, jakou zabere pixel na trojúhelníku, který je ve vzdálenosti t. Jde o výpočet protilehlé odvěsny ke známému úhlu, kde t je odvěsna přilehlá ke známému úhlu a zároveň vzdálenost kamery s bodem průniku paprsku a trojúhelníku.

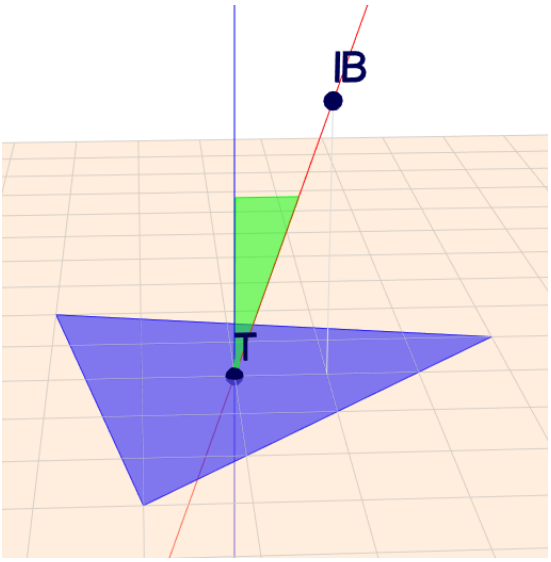
$$S = (\tan(\gamma) \cdot t)^2$$

S reprezentuje nasnímanou plochu čtverce, jak je vizualizováno na obrázku 7.2. Když teď máme nasnímanou plochu jedním pixelem, tak se porovná s maximální plochou, kterou může jeden pixel nasnímat. Pokud je nasnímaná plocha větší, tak trojúhelník nesplní požadavek na požadovanou kvalitu.
Tímto způsobem se vyšlou paprsky ze všech pixelů a u každého paprsku, který zasáhl trojúhelník a byl splněn požadavek na kvalitu snímku, se sčítá nasnímaná plocha konkrétního trojúhelníku. V případě, že nebyla splněna podmínka kvality snímku, tak se poznamená, že tento trojúhelník tento požadavek nesplnil.
Po nasnímání celé scény se sestaví inspekční bod, u kterého se drží informace o poloze, směru kamery a trojúhelníky které jsou vidět celé, částečně a které nesplňují požadovanou kvalitu. Jako trojúhelníky plně viditelné, jsou označeny ty trojúhelníky, které jsou z konkrétního inspekčního bodu nasnímané alespoň z předem definované části. Tato část je další uživatelský parametr programu. Nemůžeme říct, že viditelné jsou pouze trojúhelníky, které mají nasnímáno 100 % své plochy, protože paprsky vyslané z pixelu můžou zasáhnout trojúhelník vedle hrany, takže jako zasažený trojúhelník je označený sousední trojúhelník, ale pixel nasnímá plochu obou trojúhelníků, proto se toto řeší vstupním parametrem minimální nasnímané plochy pro označení trojúhelníku, jako viditelného. Toto číslo je vhodné zvolit mezi 0,9 a 1 (90 % - 100 %), záleží na vstupním modelu, rozlišení a FOV kamery a požadované minimální kvalitě snímku.

Přepočet nasnímané plochy

Výpočet, představený v předchozím odstavci, platí pouze pokud by všechny paprsky dopadaly kolmo na trojúhelník. Jelikož se tak bohužel neděje, jak je vidět na obrázku 7.2, tak se musí výpočty upravit. Ideální by bylo, pronásobit nasnímanou plochu funkcí, která pracuje s úhlem a nabývá hodnot $f(0) = 1$ a $f(90) = Inf$. Nejvhodnější je funkce kosinus, ovšem nasnímaná plocha se nebude touto funkcí násobit, ale dělit. Nyní stačí, kdyby se u každého paprsku nasnímaná plocha vydělila kosinem úhlu mezi dopadajícím paprskem a kolmicí k trojúhelníku v místě dopadu. To je ovšem výpočetně velmi náročné, protože by se musel počítat kosinus u každého paprsku, při zvolení rozlišení například 1280 x 720 to je 921 600 paprsků. Vzhledem k zvoleném modelu, může být provedeno tisíce až desetitisíce skenů scény. To už se dostáváme do miliard vyslaných paprsků, a pokud by se u každého měl počítat kosinus, tak se algoritmus výrazně zpomalí.
Řešením je upravit tuto plochu až po vysílání paprsků. Úhel (nazvěme ho α) se nyní nebude brát mezi paprskem a již zmíněnou kolmicí v bodě dopadu paprsku, ale mezi přímkou, která prochází inspekčním bodem a těžištěm trojúhelníku a kolmicí k trojúhelníku v místě těžiště (vizualizace na obárku 7.3). Vzhledem k tomu, že bereme tento úhel v těžišti trojúhelníku, tak by se měly

vzájemně co nejvíce pokrátit odchylky, které by vznikly tím, že různé paprsky dopadají na trojúhelník pod větším (pixel by zabral více plochy) i menším (pixel by zabral méně plochy) úhlem než, je úhel α. Tudíž se provede aproximace nasnímané plochy trojúhelníků.



Obrázek 7.3: Úhel ovlivňující nasnímanou plochu
Pozn.: IB = inspekční bod, T = těžiště trojúhelníku, úhel α se nachází mezi modrou a červenou přímkou

Při vydělení nasnímané plochy trojúhelníku kosinem úhlu α jsem zjistil, že v případě skenování menšího trojúhelníku dochází ke zkreslení. To znamená, že trojúhelník, který není celý vidět, se po přepočtu nasnímané plochy označí jako viditelný. Další problém je, že se zvětšující se vzdáleností od trojúhelníku se zmenšuje nasnímaná plocha trojúhelníku, a to i v případě, že se všechny jeho vrcholy nachází na snímku a není v zákrytu. Toto bylo patrné hlavně pro menší trojúhelníky (hranice mezi menšími a většími trojúhelníky je definována později v této sekci), viz. tabulka 7.2. Výsledky nasnímaných ploch trojúhelníků jsou v tabulkách 7.1 a 7.2. Nevyplněné buňky značí, že potenciální inspekční bod se už vyskytoval moc blízko k modelu objektu. Hodnoty byly naměřeny s rozlišením kamery 1280 x 720 a vertikálním FOV 30°. Plocha většího trojúhelníku byla 2,57 a menšího 0,12. Oba trojúhelníky byly součástí rovné plochy, kterou nezakrývala žádná překážka. Uvedená data nasnímaných ploch je poměr nasnímané plochy trojúhelníku ku jeho celkové ploše.

α / vzdálenost	4	5	6	7	8	9	10	11
0		0,9864	0,9601	0,9621	0,9547	0,9478	0,9461	0,9427
10		0,9641	0,9554	0,9478	0,9456	0,9397	0,9177	0,9145
20	0,8967	0,891	0,8877	0,8848	0,8855	0,8855	0,8856	0,8811
30	0,8143	0,8089	0,8057	0,8049	0,8039	0,8038	0,8017	0,7991
40	0,7059	0,7003	0,6979	0,6989	0,6948	0,6960	0,6981	0,6985
50			0,5764	0,5764	0,5752	0,5722	0,5724	0,5721
60			0,4395	0,4400	0,4402	0,4399	0,4390	0,4353
70					0,2963	0,2959	0,2951	0,2939

Tabulka 7.1: Nasnímaná plocha většího trojúhelníku pod různými úhly a v různé vzdálenosti od těžiště trojúhelníku

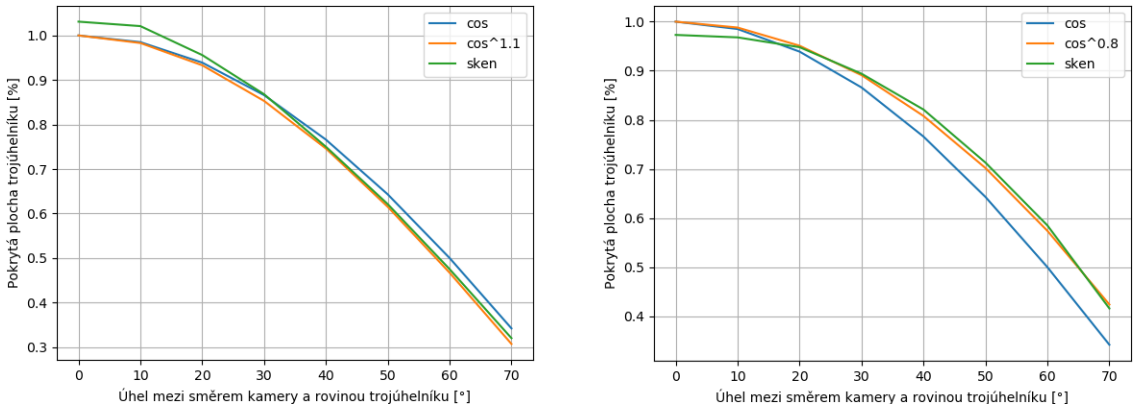
α / vzdálenost	4	5	6	7	8	9	10	11
0		0,9564	0,9233	0,8997	0,9012	0,8982	0,8968	0,8941
10		0,9259	0,9214	0,8995	0,8964	0,8906	0,8892	0,8999
20	0,8967	0,8804	0,8740	0,8710	0,8777	0,8705	0,8711	0,8693
30	0,8143	0,8352	0,8274	0,8319	0,8279	0,8310	0,8278	0,8251
40	0,7059	0,7651	0,7657	0,7603	0,7602	0,7595	0,7546	0,7499
50			0,6705	0,6663	0,6602	0,6643	0,6642	0,6551
60			0,5439	0,5412	0,5424	0,5375	0,5411	0,5354
70					0,3850	0,3854	0,3861	0,3828

Tabulka 7.2: Nasnímaná plocha menšího trojúhelníku pod různými úhly a v různé vzdálenosti od těžiště trojúhelníku

Problém horšího skenu při zvětšování vzdálenosti řeší zvětšení naskenované plochy v závislosti na vzdálenosti inspekčního bodu od těžiště trojúhelníku a měřítku modelu. Měřítko modelu je další (číselný) uživatelský vstup, který říká, kolikrát je model větší, než pokud by byl v měřítku 1:1.

$$S_n = S_o + ((S_o/100) \cdot d/M)$$

Kde M je parametr měřítka modelu, d je vzdálenost inspekčního bodu od těžiště snímaného trojúhelníku, S_o je původní naskenovaná plocha a S_n je zvětšená plocha. Výsledky po přičtení této části lze vidět v tabulkách 1 a 2 v příloze. Pro vyřešení problému s různým úhlem dopadu paprsku na trojúhelník, je potřeba najít takovou funkci používající kosinus úhlu α, aby se hodnoty z tabulek 1 a 2 normalizovaly, jinak řečeno, aby hodnoty po vydělení touto funkcí byly rovny 1. Tato funkce se bude lišit podle toho, jestli se jedná o trojúhelník menší nebo větší. Hledané funkce jsem za pomocí naměřených dat z tabulek aproximoval. Pro větší trojúhelníky se použije funkce $f(x) = \cos(\alpha)^{1,1}$ a pro menší trojúhelníky $f(x) = \cos(\alpha)^{0,8}$. Průběhy funkcí lze vidět grafech na obrázku 7.4.



(a) Větší trojúhelníky

(b) Menší trojúhelníky

Obrázek 7.4: Grafy zobrazující aproximované funkce na skenech ze vzdálenosti 8
Pozn.: modrá = průběh funkce kosinus, zelená = skeny s přičtenou plochou (této funkce se snažíme dosáhnout), oranžová = aproximovaná funkce

Odchylka funkce mezi hodnotami 0° - 30° tolik nevadí, protože při menším úhlu α nedochází k takové odchylce u skenu, takže i bez přepočtu se pohybujeme u výsledného skenu v hodnotách kolem 1. To, že se menší trojúhelníky musí pronásobit jinou funkcí, než větší dává smysl, protože odchylka vzniká u zaznamenání plochy při zásahu paprskem. Větší trojúhelníky zasáhne více paprsků, se zvětšujícím se úhlem α narůstá odchylka u každého zásahu paprskem. Tudíž u většího

trojúhelníku, se musí naskenovaná plocha dělit menším číslem než u menších trojúhelníků. Nyní ještě určit mez, která bude dělit trojúhelníky na větší a menší. Zde opět závisí na rozlišení a FOV kamery, protože tyto parametry, spolu se vzdáleností trojúhelníku od kamery, definují jeho naskenovanou plochu. Proto se u zvolení hranice větších a menších trojúhelníků použije výpočet nasnímané plochy jedním pixelem. Jelikož vzdálenost od trojúhelníku se mění, použije se plocha nasnímaná pixelem v jednotkové vzdálenosti. Hranice byla zvolena jako $nasimana\_plocha\_pixelem \cdot 500000$. Nejde o přesnou hranici, ta se při různém nastavení rozlišení, FOV a různých trojúhelnících pohybovala. Bohužel nikdy nelze najít přesnou hodnotu, protože u skenování záleží, kolik paprsků z dané pozice zasáhne skenovaný trojúhelník. Hodnota byla zvolena na základě pokusů a dopočítávání zasažených ploch pomocí aproximovaných funkcí, u různě velkých trojúhelníků. Výsledná nasnímaná plocha po přepočtu lze vidět v tabulkách 7.3 a 7.4.

α / vzdálenost	4	5	6	7	8	9	10	11
0		1.0357	1.0177	1.0295	1.0311	1.0331	1.0407	1.0464
10		1.0279	1.0299	1.0314	1.0386	1.0417	1.0266	1.0323
20	0.9602	0.9956	1.0076	1.0138	1.0241	1.0335	1.0431	1.0473
30	0.96244	0.9807	1.0005	1.0089	1.0170	1.0263	1.0330	1.0391
40	0.9541	0.9599	0.9918	1.0026	1.0060	1.0171	1.0295	1.0395
50			0.9899	1.0028	1.0101	1.0141	1.0238	1.0326
60			0.9986	1.0092	1.0191	1.0278	1.0351	1.0357
70					1.0416	1.0498	1.0566	1.0619

Tabulka 7.3: Nasnímaná plocha většího trojúhelníku pod různými úhly a v různé vzdálenosti od těžiště trojúhelníku po finálním přepočtu

α / vzdálenost	4	5	6	7	8	9	10	11
0		1.0042	0.9787	0.9627	0.9733	0.9791	0.9865	0.9925
10		0.9872	0.9887	0.9743	0.9800	0.9827	0.9902	1.0112
20	0.9264	0.9837	0.9737	0.979	0.9963	0.9972	1.0071	1.0141
30	0.9217	1.0126	0.9840	0.9987	1.0031	1.0162	1.0216	1.0276
40	0.8807	1.0487	1.0045	1.0069	1.0161	1.0246	1.0273	1.0302
50			1.0121	1.0153	1.0154	1.0311	1.0405	1.0356
60			1.0038	1.0082	1.0199	1.0201	1.0363	1.0347
70					0.9809	0.9910	1.0019	1.0024

Tabulka 7.4: Nasnímaná plocha menšího trojúhelníku pod různými úhly a v různé vzdálenosti od těžiště trojúhelníku po finálním přepočtu

Odstranění trojúhelníků se špatnou kvalitou

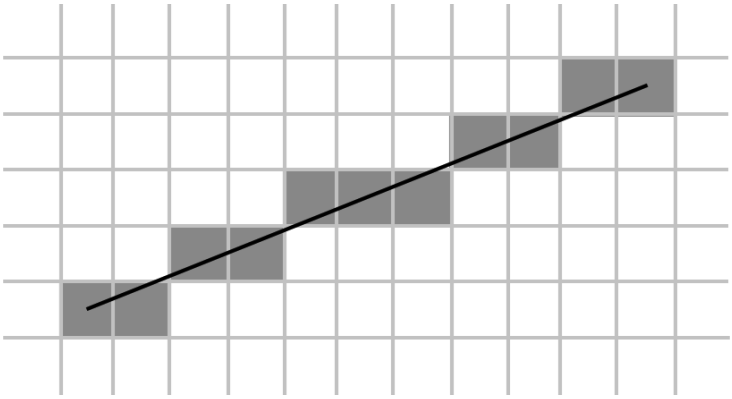
Nyní je ještě potřeba odstranit trojúhelníky, které nesplňují požadovanou kvalitu skenu. Tato část se už řešila během samotného skenování, ale nepočítalo se s úhlem mezi trojúhelníkem a paprskem. Pro každý zaznamenaný trojúhelník u inspekčního bodu proběhne kontrola (kromě těch, co mají záznam o nesplněné kvalitě), zda pixely reprezentované paprsky vyslanými do jeho vrcholů splňují požadovanou kvalitu, pokud ne, tak se trojúhelník přesune do seznamu trojúhelníků nesplňujících kvalitu.

7.3.3 Skenování konkrétních trojúhelníků

Sken celé scény zabírá velké množství času, protože s každým vyslaným paprskem provádí implementace kamery z použitého frameworku dotaz do datové struktury. Pro zrychlení celého

algoritmu se u hledání nových inspekčních bodů pro konkrétní trojúhelníky používá pouze skenování konkrétního trojúhelníku a až nakonec sken celé scény. Ke skenu konkrétního trojúhelníku je potřeba vědět, na jakých pixelech se trojúhelník nachází. Nejdříve se zjistí, na jakých pixelech se nachází vrcholy trojúhelníku tak, že se udělá průmět vrcholů na pixely kamery[4]. Dále se najdou pixely mezi těmito vrcholy, k tomu slouží Bresenhamův algoritmus [24].

Bresenhamovým algoritmem se nazývají algoritmy, které vychází z původního algoritmu představeného J. E. Bresenhamem už v roce 1962. Jde o algoritmus, který zjistí, kterými buňkami v N-dimenzionální tabulce prochází přímka mezi dvěma body. Tento algoritmus se stále používá (v modifikovaných a rozšířených formách) v počítačové grafice dodnes, protože se v něm používají výpočetně levné operace, jako je bitový posun nebo přičítání/odečítání celého čísla. Algoritmus je pouze aproximační, tudíž nemusí označit buňku, kterou přímka prochází pouze z malé části . Ukázka Výsledku Bresenhamova algoritmu je na obrázku 7.5.



Obrázek 7.5: Použití Bresenhamova algoritmu ve 2D[5]

Tímto algoritmem se najdou pixely, které pokrývají hrany trojúhelníku. Nyní se do výsledného seznamu přidají ty pixely, které se nachází mezi těmito třemi hranami. Pozor se musí dát v případě, že se nějaký z vrcholů nachází mimo zorné pole kamery. V takovém případě se přidají jen ty pixely, které spadají do rozlišení kamery. Nyní je k dispozici seznam pixelů pokrývajících trojúhelník. Skenování konkrétního trojúhelníku funguje stejně jako při skenování celé scény s tím rozdílem, že se provádí vyslání omezeného množství paprsků.

7.3.4 Výpočet maximální vzdálenosti inspekčního bodu od trojúhelníku

Po vysvětlení funkce skenování lze vypočítat maximální vzdálenost inspekčního bodu od těžiště trojúhelníku, protože se zde používá princip z evaluace skenu. K výpočtu se použije podobný princip jako u výpočtu nasnímané plochy pixelem, nyní se ale nehledá protilehlá odvěsna k úhlu (což byla hrana z čtvercové plochy, kterou pixel nasnímá), ale snažíme se zjistit délku přilehlé odvěsny k úhlu (což je vzdálenost kamery od snímané plochy kolmo před ní) za použití maximální plochy, kterou může pixel nasnímat, aby se splnil požadavek kvality snímku.
Prvně se vypočte maximální možná vzdálenost kamery od místa dopadu paprsku vyslaného z pixelu [0,0]. To se udělá tak, že maximální požadovaná plocha nasnímaná jedním pixelem se vynásobí kosinem poloviny diagonálního úhlu, protože přesně takový úhel je mezi vyslaným paprskem a rovinou, na kterou paprsek dopadá. Tímto se zjistí plocha, kterou pokrývá rohový pixel. Tato hodnota se odmocní, čímž dostaneme velikost hrany snímané obdélníkové plochy. Nyní se dostáváme k části podobné z předchozího výpočtu. Známe protilehlou odvěsnu k úhlu

[4]Použitý algoritmus z `https://en.wikibooks.org/wiki/Cg_Programming/Vertex_Transformations`

[5]Obrázek stažen z `https://projectprinter2d.wordpress.com/2015/02/12/point-to-point-with-linear-path-bresenhams-algorithm-implementation/`

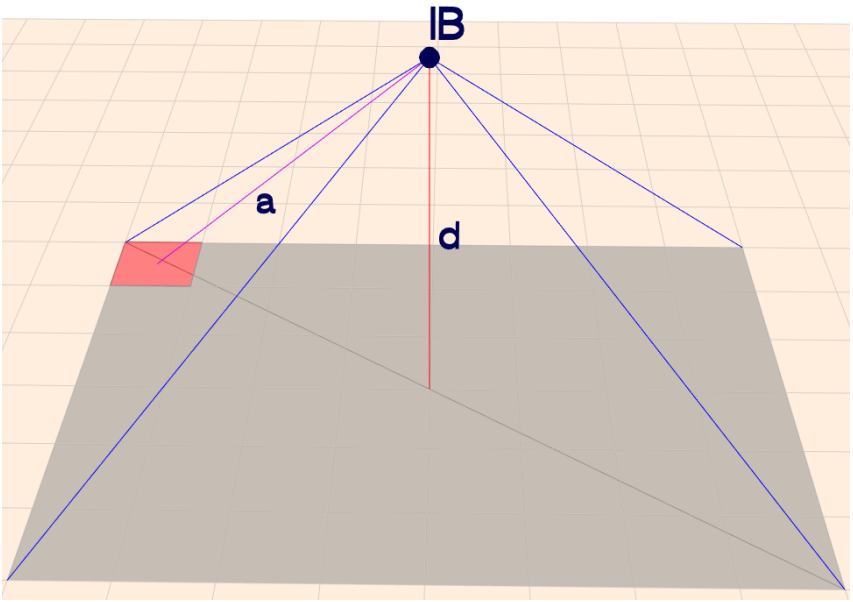
γ, který je stejně, jako v předchozím výpočtu úhel záběru jednoho pixelu. Nyní se za pomocí výpočtů v pravoúhlém trojúhelníku vypočítá přilehlá odvěsna k úhlu γ.

$$\gamma = \frac{FOV_{vert}}{y}$$

$$a = \frac{\sqrt{S_{max} \cdot \cos(\frac{FOV_{diag}}{2})}}{\tan(\gamma)}$$

Kde S_{max} je maximální povolená plocha nasnímaná jedním pixelem a y je počet pixelů na výšku. Proměnná a je naše vypočítaná přilehlá odvěsna a maximální vzdálenost, jenže stále to není naše hledaná maximální vzdálenost, toto je maximální vzdálenost od bodu průniku roviny s paprskem vyslaným z pixelu [0,0] a kamery, my chceme maximální vzdálenost inspekčního bodu ležícího na kolmici k rovině. K tomu se použije opět výpočet v pravoúhlém trojúhelníku. Naše dříve vypočtená vzdálenost je délka přepony v trojúhelníku, polovina diagonálního FOV je náš známý úhel a chceme zjistit délku odvěsny přilehlé k tomuto úhlu. Hledaná maximální vzdálenost d se vypočte následovně:

$$d = \cos(\frac{FOV_{diag}}{2}) \cdot a$$



Obrázek 7.6: Vizualizace výpočtu maximální vzdálenosti
červená plocha = plocha pokrytá pixelem [0,0], IB = inspekční bod

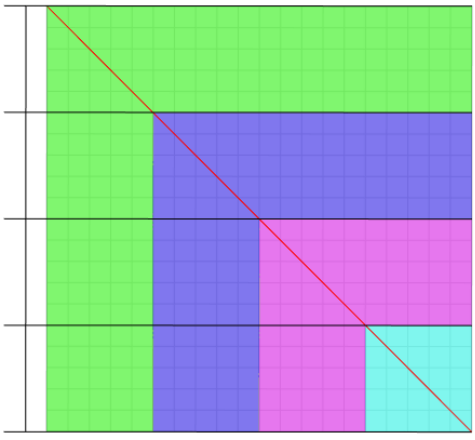
7.4 Prvotní dělení trojúhelníků

Jestli se trojúhelník vejde na snímek se zkouší tak, že se posadí kamera na normálu trojúhelníku do určité vzdálenosti a udělá se průmět vrcholů na pixely kamery. Pokud se vrcholy nachází mimo pixely rozlišení kamery, tak je trojúhelník moc velký a musí se rozdělit. Vzdálenost kamery od trojúhelníku je určena dalším uživatelským parametrem, kterým se vynásobí vypočtená maximální vzdálenost pro inspekční bod. Proto tento parametr musí být menší nebo roven 1, volba tohoto parametru ovlivní počet dělených trojúhelníků. Čím dále je kamera od trojúhelníku, tím menší počet trojúhelníků je na začátku algoritmu rozdělen, ale zároveň hrozí, že tyto trojúhelníky budou moc velké a nevejdou se do zorného pole kamery během vyhledávání inspekčních bodů, které probíhá v menší vzdálenosti kamery od trojúhelníku. Naopak při zvolení kratší vzdálenosti

kamery od trojúhelníku v případě prvotního dělení, vyústí ve vyšší počet trojúhelníků, ke kterým se hledají inspekční body.

7.5 Hledání nejkratších cest mezi inspekčními body

K hledání nejkratších cest mezi inspekčními body se využije voxelizovaný model objektu, kde plné voxely reprezentují překážku pro vyhledávací algoritmus. Nejdřív je potřeba voxelizovaný model zvětšit, aby do něj spadaly všechny nalezené inspekční body. Stačí aby všechny strany byly prodlouženy o maximální vzdálenost inspekčního bodu od trojúhelníku, protože ve větší vzdálenosti se žádné inspekční body nacházet nemohou. Nově vytvořené voxely budou vždy prázdné, s výjimkou těch, jejich souřadnice by byly v kolizi se zemí. Voxely slouží jako vrcholy grafu pro algoritmus, který najde nejkratší cestu z vrcholu A do vrcholu B. Jelikož některé modely mohou být poměrně velké (stovky voxelů v různých směrech), rozhodl jsem se použít algoritmus A* (6.1.2), protože prohledává menší prostor, jak je vidět na obrázku 6.1. Jako heuristickou funkci jsem použil euklidovskou vzdálenost mezi aktuálním a koncovým bodem. Pokud se body nachází ve stejných nebo sousedních voxelech, vezme se jako výsledek jejich euklidovská vzdálenost.
U nalezené výsledné vzdálenosti je možnost uložit si i celou cestu, jako seznam voxelů, kterými je nutno projít. To jsem z důvodu paměťové optimalizace nedělal, protože vybraných inspekčních bodů mohou být tisíce, jedna cesta mezi dvěma inspekčními body může, v závislosti na nastavení parametrů a voxelizovaném modelu, obsahovat stovky voxelů. Výsledkem je obrovské množství dat, které si program musí ukládat do paměti. Zároveň je v paměti uložen každý nalezený adept na inspekční bod (bod, ve kterém se prováděl sken celé scény) i se seznamy trojúhelníků, které pokrývá celé nebo jen z části. I když jsem procesu přidělil přes přepínač "*-Xmx*"6 GB paměti, tak u větších modelů program zahlásil chybu. Konkrétně chybu "*java.lang.OutOfMemoryError: GC overhead limit exceeded*"což znamená, že procesu došla paměť a tráví většinu času uvolňováním paměti. Proto jsem se rozhodl neukládat nalezené cesty, ale pouze jejich délky.



Obrázek 7.7: Ukázka paralelizace vyplňování matice vzdáleností
Pozn.: buňky stejné barvy zpracovává jedno vlákno

Matice vzdáleností je čtvercová matice, jejíž rozměr je definovaný počtem inspekčních bodů. Tato matice je symetrická podle diagonály (pro prvky matice A platí $A_{ij} = A_{ji}$) a na diagonále má nuly. Proto stačí vypočítat pouze hodnoty nad diagonálou a překopírovat je do adekvántího prvku pod diagonálou.
Z důvodu zrychlení algoritmu jsem se rozhodl celý problém hledání nejkratších cest paralelizovat. Řádky matice se rozdělí na k stejně velkých částí. Každou tuto část pak zpracovává samostatné vlákno, jak je vizualizováno na obrázku 7.7. Číslo k je definováno vstupním parametrem, protože zde záleží na procesoru, na kterém algoritmus běží. Tento vícevláknový přístup je bezpečný, protože žádné vlákno nepřistupuje k žádným datům, které by mohlo jiné vlákno změnit.

7.6 Hledání nejkratší cesty pro inspekci dronem

Samotné hledání nejkratší trasy přes inspekční body se provádí za použití algoritmů řešících TSP (6.2). Zde jsem použil Java framework JAMES. Jde o framework napsaný v jazyce Java,

který slouží k řešení problémů diskrétní optimalizace [25]. Jelikož použitý framework provádí pouze optimalizaci výsledku, bylo potřeba mu dodat už nějakou vypočítanou trasu, kterou by mohl dále optimalizovat. K tomuto jsem použil řešení nejbližších sousedů (6.2.2). Frameworku se tento seznam předá spolu s optimalizačním algoritmem a časovým omezením běhu tohoto algoritmu. Toto časové omezení je vstupní parametr programu. Jako optimalizační algoritmus jsem zvolil 2-OPT (6.2.2).
Po dokončení optimalizace je k dispozici seznam inspekčních bodů v pořadí, jakým jimi bude dron prolétat. Nyní se musí najít cesta mezi těmito body. K tomu se použije opět A* algoritmus, jenže v tomto případě je nejkratší cesta vyhledávána jen mezi dvěma za sebou jdoucími body ze seznamu. Předtím, než se spustí A* je ještě přidána do seznamu startovní (která je zároveň konečná) pozice. Tuto pozici jsem zvolil jako krajní prázdný voxel, který se nachází těsně nad zemí. Z tohoto bodu je nalezen nejbližší inspekční bod ze seznamu. Seznam se nyní přeřadí tak, aby první prvek byl ten nalezený nejbližší inspekční bod a poslední prvek byl inspekční bod, který se v seznamu nacházel před oním nejbližším. Pořadí mezi těmito body zůstává takové, jaké vrátil 2-OPT algoritmus. Do tohoto seznamu je na začátek a konec přidána startovní a koncová pozice. Teď je už možné zahájit vyhledávání nejkratších cest mezi inspekčními body. Výsledná cesta mezi dvěma body je seznam za sebou jdoucích voxelů.
Jelikož algoritmus může v některých případech najít cestu, která není přímá, aniž by se snažil vyhnout překážce, tak se cesta musí napřímit. Výhoda napřímení cesty je menší počet výstupních bodů k průletu a zároveň zkrácení cesty. K napřímení cesty se použije opět Bresenhamův algoritmus (7.3.3), ale nyní v úpravě pro 3D. Algoritmus najde všechny voxely, kterými prochází přímka mezi dvěma vybranými voxely z cesty. Napřimování cesty funguje tak, že se zvolí počáteční voxel (na začátku první z výsledné cesty) a postupně se kontroluje přímka mezi ním a následujícími voxely. Pokud žádná přímka neprochází plným voxelem, tak je můžno všechny voxely mezi krajními přeskočit a výsledná cesta bude přímka mezi krajními voxely. Ale v případě, že přímka prochází plným voxelem, tak se přeskočí pouze voxely mezi počátečním voxelem a předchůdcem voxelu, u kterého nastal tento případ (nazvěme ho voxel B). Dále se jako počáteční voxel vezme právě voxel B a pokračuje se od něj dále.
Voxely se po napřímení cesty převedou na body. U krajních voxelů (ve kterých leží inspekční body) se jako souřadnice vezme pozice inspekčního bodu a u voxelů mezi nimi se vezme souřadnice odpovídající středu voxelu. Tímto způsobem se sestaví cesty mezi všemi po sobě jdoucími inspekčními body ze seznamu, jejich spojením přes inspekční body vznikne výsledná cesta pro dron.

Kapitola 8

Experimenty

Vlastnosti algoritmu byly ověřeny na hledání trajektorie pro inspekci 3D objektu bezpilotním prostředkem. Algoritmus se vyzkoušel na různých modelech s různě nastavenými vybranými vstupními parametry. Průběhy a výsledky algoritmu s různými vstupy se vzájemně porovnaly a kontrolovaly, zda odpovídají předpokládaným hodnotám.

8.1 Vstupní parametry s referenčním nastavením

V práci bylo zmíněno mnoho parametrů, které ovlivňují běh algoritmu a jeho výstup. V této kapitole se nachází referenční nastavení všech parametrů. S tímto nastavením byly nalezeny cesty pro inspekci všech 4 modelů a následně vyzkoušeny v simulačním nástroji. Vůči tomuto nastavení se porovnávaly změny v běhu algoritmu a jeho výsledcích v další fázi experimentů. Tyto referenční hodnoty nebyly zvoleny náhodně, ale tak, aby se co nejvíce blížily k hodnotám reálného použití, ale zároveň aby různé části běhu algoritmu netrvaly příliš dlouho. Referenční hodnoty jsou následovné:

- rozlišení kamery - 1280 x 720;
- vertikální FOV - 30°;
- minimální požadovaná kvalita snímku - 0,0001 (1 cm^2/px);
- minimální zasažená plocha trojúhelníku - 0,1 (10 % jeho plochy);
- doba TSP optimalizace - 200 s;
- A* paralelizace - 8;
- pokrytí viditelného trojúhelníku - 0,92 (92 % jeho plochy);
- počet iterací hledání inspekčních bodů - 3;
- minimální pokrytá plocha pro dělení trojúhelníku - 0,2 (20 % jeho plochy);
- vzdálenost pro dělení velkých trojúhelníků - 0,6 (60 % maximální vzdálenosti);
- velikost kroku pro hledání nového inspekčního bodu - 0,4;
- šířka prohledávaného prostoru pro nový inspekční bod - 20;
- redundance snímků - 3;
- minimální vzdálenost inspekčního bodu od trojúhelníku - 2,5;

- minimální vzdálenost od bodu pro splnění redundance snímků - 2;
- minimální úhel mezi vektory směru inspekčních bodů pokrývající stejný trojúhelník pro splnění redundance snímků - 10°;
- plocha příliš malého trojúhelníku - 0,02;
- bezpečná vzdálenost - 2;
- měřítko modelu - 1;

8.2 Testované modely

K vývoji a testování jsem použil modely, které byly zdarma ke stažení z webových stránek TurboSquid[1] a Free3D[2]. K testování byly použity 4 modely, které byly vybrány tak, aby byly co nejvíce odlišné a algoritmus se otestoval co nejkomplexněji. Testované modely jsou:

- Kancelářská budova - poměrně členitý a netriviální 3D model, který obsahuje 35552 trojúhelníků před dělením. Těžším pro inspekci činí model hlavně větší množství oken zapuštěných do budovy s okenicemi a jejich okolím. Model obsahuje trojúhelníky i ve vnitřní části a zároveň schované trojúhelníky pod jinými. Tyto trojúhelníky není možno nasnímat.
- Hangár pro menší letadla - další poměrně členitý a netriviální 3D model, který obsahuje 10984 trojúhelníků před dělením. Model na střeše obsahuje různé komínky a části ventilace, které se musí nasnímat. Dále přístřešky, které musí být nasnímány i ze spodní strany. Některé trojúhelníky jsou zakryté a není je možno nasnímat.
- Obloukový most - jednodušší model s 1068 trojúhelníky před dělením. Problematickou částí tohoto modelu jsou stěny pod obloukem mostu.
- Listy s rotorem z větrné elektrátny - jednodušší model s 13444 trojúhelníky před dělením.

Bohužel ani jeden z modelů nepochází z fotogrammetrie. Z toho důvodu u modelu budovy a hangáru nastáva situace, že se tyto modely skládají z menších částí, které jsou k sobě naskládány tak, že mají společnou stěnu, tudíž některé trojúhelníky jsou nedostupné.

8.3 Výsledky běhů algoritmu s referenčním nastavením

Modely představené v předchozí kapitole byly vyzkoušeny s referenčním nastavením na navrženém algoritmu. Při běhu algoritmu jsem zaznamenával informace o počtu inspekčních bodů, počtu trojúhelníků a době běhu různých fází algoritmu. Výsledky jsou vidět v tabulkách 8.1 a 8.2. Z tabulek je vidět, že u menších modelů, u kterých se nenachází překryv trojúhelníků, se ve 3. iteraci algoritmu pro vyhledávání nových inspekčních bodů už žádný nový nenašel. V případě modelu budovy a hangáru by se trojúhelníky mohly dělit i dále, ale nemá cenu zvolit velký počet iterací, protože může nastat, že nové trojúhelníky po dělení již budou mít plochu menší, než je parametrem daná plocha pro příliš malé trojúhelníky a tyto trojúhelníky by se už nezahrnuly do dalšího kola vyhledávání inspekčních bodů. Dále je z tabulek vidět, že čas strávený hledáním inspekčních bodů, je přímo úměrný počtu trojúhelníků v modelu. Stejně tak výběr množiny nejvhodnějších inspekčních bodů k pokrytí objektu je přímo úměrný počtu nalezených inspekčních bodů.

[1] `https://www.turbosquid.com/`
[2] `https://free3d.com/`

	Most	Větrná elektrárna
Sken z kolmic v max. vzdálenosti	225 s / 866 nových IB	2284 s / 8227 nových IB
1. iterace hledání IB	647 s / 1727 nových IB	34 s / 61 nových IB
2. iterace hledání IB	0 s / 2 nové IB	0 s / 0 IB
3. iterace hledání IB	0 s / 0 IB	0 s / 0 IB
Sken vybraných trojúhelníků z existujících IB	0 s	151 s
Výběr IB	1 s	13 s
Hledání IB pro splnění redundance	82 s / 272 nových IB	6 s / 18 nových IB
Celkem	**965 s / 2867 nalezených IB**	**2508 s / 8306 nalezených IB**
Vybráno IB	**813**	**1070**
A*	291 s	28 s
TSP	200 s	200 s
Celkem	**1456 s**	**2736 s**
Počet trojúhelníků na začátku	1068	13444
Počet trojúhelníků na konci	4530	17554
Viditelných trojúhelníků	3092	15443
Trojúhelníky splňující redundanci	2693	13859
Počet příliš malých trojúhelníků	0	8895

Tabulka 8.1: Data nasbíraná z běhu algoritmu s referenčním nastavením na modelu mostu a větrné elektrárny

IB = inspekční bod

U hledání nejkratších cest mezi inspekčními body algoritmem A* už záleží na více faktorech. Jak je vidět z tabulek, tak model mostu a větrné elektrárny mají vybráno podobný počet inspekčních bodů, které tyto modely pokrývají, ale hledání cest mezi nimi zabralo v případě modelu mostu mnohem delší dobu. Zde záleží na velikost prohledávaného prostoru, který je daný voxelovým modelem. V případě modelu mostu, má voxelový model rozměry 74 x 19 x 13, model větrné elektrárny má rozměry 52 x 3 x 45, model budovy 21 x 24 x 38 a model hangáru 39 x 25 x 12. Proto je u voxelizace vhodné zvolit co nejmenší rozměry voxelizovaného modelu s tím, aby byly stále zachovány rysy původního polygonálního modelu.

To, co nás nejvíce zajímá je celkové pokrytí objektu. Zde se musí brát v potaz, že trojúhelníky na spodní straně modelů nebyly, až na model větrné elektrárny, vůbec skenovány, protože v tomto místě se nachází úroveň země.
U modelu hangáru je pokryto 14578 trojúhelníků z 25328, to je 57,6 %. Trojúhelníků splňujících překryv je 13055, to je 51,5 %. Ale jak bylo psáno u tohoto modelu se nachází velké množství trojúhelníků zakrytých, dále jsou zde trojúhelníky na spodní straně modelu. Tento model navíc obsahuje 6010 moc malých trojúhelníků, u kterých se ani nezkouší vyhledávat inspekční bod. V případě modelu mostu je pokryto 3092 trojúhelníků z celkových 4530, to je 68,3 % trojúhelníků. Nepokryté trojúhelníky se nachází na spodní straně mostu a v oblouku, ve kterém se inspekční body nemohou nacházet, kvůli bezpečné vzdálenosti. Podmínku překryvu splňuje 2693 trojúhelníků, cože je 59,4 %. U modelu větrné elektrárny, což je reálný případ použití algoritmu, je pokryto 15443 trojúhelníků z celkových 17554, to je 88 %. Z toho jich podmínku překryvu splňuje 13859, to je 80%. To už není tak špatné. Model obsahuje 8895 malých trojúhelníků ke kterým se nehledají inspekční body. Tyto trojúhelníky se nachází ve špičkách listů vrtule a kolem rotoru. Další trojúhelníky jsou zakryté v místě kolem rotoru, kde se spojují listy. A u posledního modelu budovy je pokryto pouze 18274 trojúhelníků z celkových 44363 (41 %). Podmínku překryvu splňuje 17648 trojúhelníků (40 %). Tento model ovšem obsahuje velké

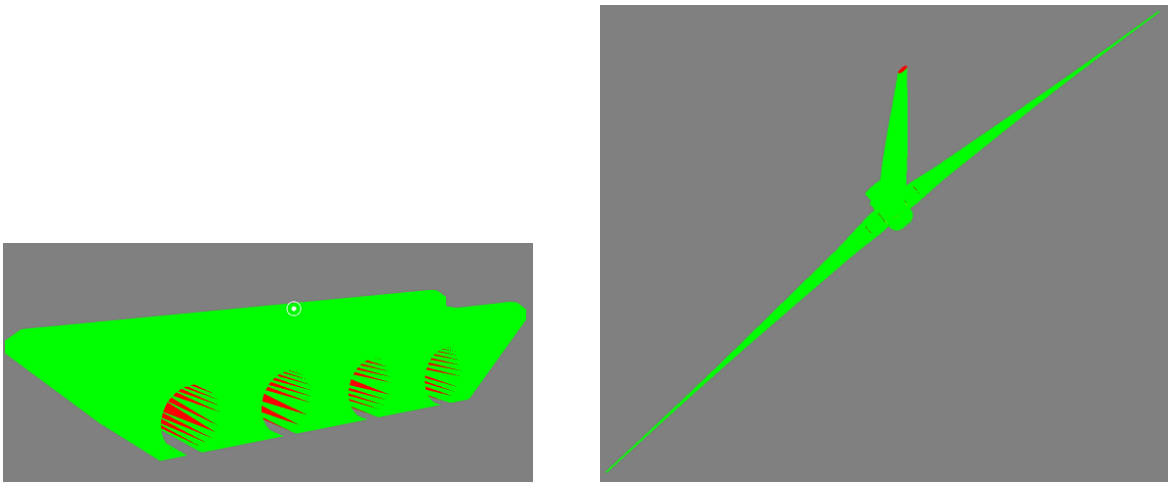
	Budova	Hangár
Sken z kolmic v max. vzdálenosti	2672 s / 10426 nových IB	1585 s / 5261 nových IB
1. iterace hledání IB	3521 s / 8644 nových IB	3209 s / 4412 nových IB
2. iterace hledání IB	1274 s / 2903 nových IB	1079 s / 863 nových IB
3. iterace hledání IB	1327 s / 3433 nových IB	870 s / 920 nových IB
Sken vybraných trojúhelníků z existujících IB	609 s	293 s
Výběr IB	77 s	22 s
Hledání IB pro splnění redundance	23 s / 34 nových IB	182 s / 486 nových IB
Celkem	**9585 s / 25440 nalezených IB**	**5683 s / 11942 nalezených IB**
Vybráno IB	**2320**	**3141**
A*	957 s	718 s
TSP	200 s	200 s
Celkem	**10742 s**	**6601 s**
Počet trojúhelníků na začátku	35552	10984
Počet trojúhelníků na konci	44363	25328
Viditelných trojúhelníků	18274	14578
Trojúhelníky splňující redundanci	17648	13055
Počet příliš malých trojúhelníků	5842	6010

Tabulka 8.2: Data nasbíraná z běhu algoritmu s referenčním nastavením na modelu budovy a hangáru

IB = inspekční bod

množství trojúhelníků schovaných za jinými trojúhelníky, takže se nedají nasnímat.

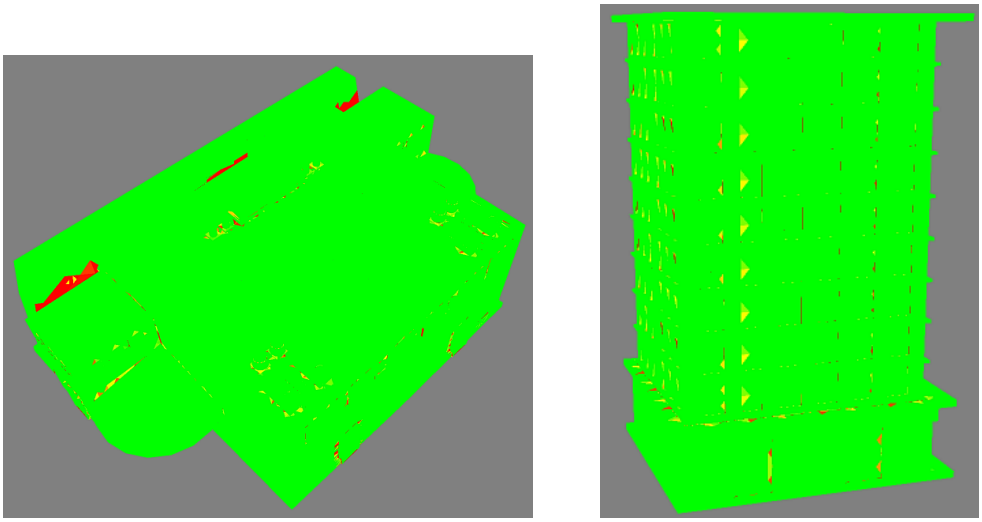
Vizualizace pokrytí objektů je na obrázcích 8.1 a 8.2. Další obrázky s pokrytí objektů jsou k nalezení v příloze (obrázek 1). Zelená barva znázorňuje pokryté trojúhelníky, červená nepokryté a žlutá částečně pokryté.



(a) Most (b) Větrná elektrárna

Obrázek 8.1: Finální pokrytí objektů mostu a větrné elektrárny

červené trojúhelníky = nepokryté, žluto-oranžové trojúhelníky = částečně pokryté, zelené trojúhelníky = pokryté



(a) Hangár z vrchu (b) Budova

Obrázek 8.2: Finální pokrytí objektů hangáru a budovy
červené trojúhelníky = nepokryté, žluto-oranžové trojúhelníky = částečně pokryté, zelené trojúhelníky = pokryté

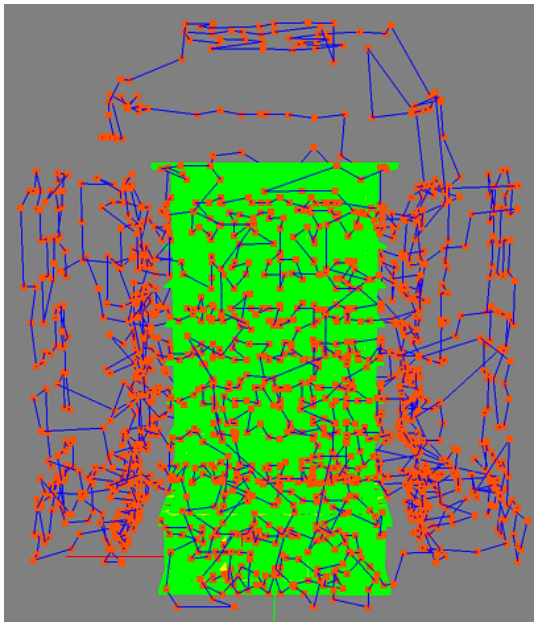
U mostu a větrné elektrárny je vidět, že jsou objekty téměř celé pokryté. Dobrou vizuální úroveň pokrytí mají i modely hangáru a budovy, zde se už ale nachází pouze částečně pokryté trojúhelníky, a to z toho důvodu, že druhá část trojúhelníku je zakryta za jinou částí modelu a není ani možné tento trojúhelník celý nasnímat. V tomto případě má cenu zkusit více iterací vyhledávání inspekčních bodů, protože tyto částečně zakryté trojúhelníky se rozdělí a je pravděpodobné, že po rozdělení bude jeden z dvou nových trojúhelníků možno kompletně pokrýt. Tyto nezakryté části trojúhelníků jsou stále nasnímány, pouze se pro ně nevyhledávají další inspekční body pro splnění podmínky překryvu snímku.
U modelu mostu a hangáru se ale nachází červeně znázorněné nepokryté trojúhelníky. Jde o trojúhelníky, které se nachází v pro dron nedostupných místech (pod obloukem mostu, mezi zdmi s malou mezerou mezi nimi). Počet těchto nepokrytých trojúhelníků lze částečně eliminovat nastavením kamery, ale v některých případech se inspekční bod, který by tyto trojúhelníky pokrýval, nenajde.

Na obrázku s výslednou trajektorií dronu (obrázek 8.3) můžeme rozpoznat různé fáze hledání inspekčních bodů. Inspekční body se nachází ve třech vrstvách kolem objektu. Nejvzdálenější vrstva jsou inspekční body, které byly vytvořeny během vyhledávání základního pokrytí z kolmic k trojúhelníku v maximální vzdálenosti. Dvě bližší vrstvy jsou body vytvořené vyhledávacím algoritmem, hledajícím inspekční body u nepokrytých trojúhelníků po fázi skenování z kolmic. Další obrázky s trajektoriemi lze najít v příloze (obrázky 2, 3, 4).

U výsledné cesty, kterou bude dron prolétat, nás zajímá nejvíce jak dlouhá tato cesta bude, kvůli časové náročnosti samotné inspekce. To je také důvod, proč se ze všech inspekčních bodů vybíraly pouze ty nejvhodnější a také, proč se výsledná cesta optimalizuje. Optimalizační 2-OPT algoritmus běžel ve všech případech 200 sekund. Z tabulky 8.3 je vidět, že optimalizace výsledných cest zmenšila vzdálenosti průměrně o 9,78 %.

	Budova	Hangár	Most	Větrná elektrárna
Metoda nejbližších sousedů	4559	4042	2482	1761
2-OPT	4108	3649	2263	1573

Tabulka 8.3: Vylepšení vzdáleností výsledné cesty po optimalizaci



Obrázek 8.3: Výsledná trajektorie dronu pro model budovy
červené body = inspekncí body, modré úsečky = cesty mezi inspekčními body

8.4 Změny parametrů

V této části otestovalo chování algoritmu změnami vybraných parametrů. Vybranými parametry jsou zákaznické parametry a změna rozlišení kamery. Změny parametrů se testovaly pouze na modelech větrné elektrárny a mostu, protože neobsahují (nebo pouze v malém množství) skryté trojúhelníky, které negativně ovlivňují statistiky pokrytí objektu.

8.4.1 Změna požadované kvality snímku

Referenční hodnota tohoto parametru byla zvolena jako 0,01 x 0,01, takže pokrytá plocha objektu jedním pixelem může být maximálně 0,0001. V tabulkách 8.4 a 8.5 pozorujeme co nastane, když se tato hodnota změní.

Most	**Referenční**	**0,012 x 0,012**	**0,007 x 0,007**
Sken z kolmic v max. vzdálenosti	866 nových IB	932 nových IB	1333 nových IB
1. iterace hledání IB	1727 nových IB	924 nových IB	2340 nových IB
2. iterace hledání IB	2 nové IB	19 nových IB	0
3. iterace hledání IB	0	1 nový IB	0
Sken IB pro splnění redundance	272 nových IB	343 nových IB	678 nových IB
Celkem	2867 IB	2219 IB	4351 IB
Vybráno	813 IB	764 IB	1401 IB
Počet trojúhelníků na začátku	1068	1068	1068
Počet trojúhelníků na konci	4530	3221	7352
Viditelných trojúhelníků	3092	2187	4158
Trojúhelníků splňujících redundanci	2693	2002	3472
Pokryto	68,3 %	67,9 %	56,5 %
Pokryto s redundancí	59,4 %	62,2 %	47,2 %

Tabulka 8.4: Tabulka s výsledky algoritmu po změně požadované kvality snímku u modelu mostu
IB = inspekční bod

Větrná elektrárna	**Referenční**	**0,012 x 0,012**	**0,007 x 0,007**
Sken z kolmic v max. vzdálenosti	8227 nových IB	6837 nových IB	10215 nových IB
1. iterace hledání IB	61 nových IB	67 nových IB	60 nových IB
2. iterace hledání IB	0	1 nový IB	0
3. iterace hledání IB	0	0	0
Sken IB pro splnění redundance	18 nových IB	38 nových IB	97 nových IB
Celkem	8306 IB	6943 IB	10372 IB
Vybráno	1070 IB	955 IB	1362 IB
Počet trojúhelníků na začátku	13444	13444	13444
Počet trojúhelníků na konci	17554	15772	21842
Viditelných trojúhelníků	15443	13953	18530
Trojúhelníků splňujících redundanci	13859	12303	16515
Pokryto	88 %	88,5 %	84,8 %
Pokryto s redundancí	79 %	78 %	75,6 %

Tabulka 8.5: Tabulka s výsledky algoritmu po změně požadované kvality snímku u modelu větrné elektrárny

IB = inspekční bod

Z tabulek je poznat, že změna požadované kvality ovlivní počet výsledných inspekčních bodů. To dává smysl, protože tento parametr ovlivňuje vypočtenou maximální vzdálenost mezi inspekčním bodem a snímaným trojúhelníkem, tím pádem se ovlivní i vzdálenost ze které se testuje, zdali-se trojúhelník vejde do záběru během dělení. Takže při vyšším požadavku na kvalitu (menší maximální plocha nasnímaná pixelem) se vypočtená maximální vzdálenost zmenší a trojúhelníky se dělí více, takže ve výsledku je i víc inspekčních bodů. Jelikož inspekční body v maximální vzdálenosti pokryjí menší plochu, tak musí být i více inspekčních bodů ve výsledné cestě. Pokrytí modelu se razantněji změnilo pouze u mostu. V případě vyššího nároku na kvalitu snímku už kamery nebyla schopna nasnímat vzdálenější trojúhelníky na stěně oblouku.
Dále si lze všimnout, že při snížení požadavku kvality nasnímaného trojúhelníku se mírně snížilo pokrytí u modelu mostu. To je zapříčiněno prodloužením maximální možné vzdálenosti inspekčního bodu od skenovaného trojúhelníku, tato vzdálenost se používá také k dělení trojúhelníků. Svoji roli zde hraje tvar velkých trojúhelníků. Tyto velké trojúhelníky (přes celou délku objektu) se u modelu mostu nachází na jeho stěnách a vrchní části, tyto trojúhelníky se při zvětšení maximální vzdálenosti rozdělí méněkrát. Takže ve výsledku je méně pokrytých trojúhelníků, které zabírají stejnou plochu. Nedostupné trojúhelníky na spodní části mostu se také dělí, ale jsou v porovníní s trojúhelníky na stěnách a vrchní části mostu mnohem menší, takže jich po dělení bude menší počet.

8.4.2 Změna počtu překrývajících se snímků

Referenční hodnota tohoto parametru byla zvolena 3. Takže každý trojúhelník musí být pokryt alespoň ze 3 inspekčních bodů, na které jsou kladeny podmínky minimální vzdálenosti a minimálního úhlu, který musí svírat s trojúhelníkem. V tablukách 8.6 a 8.7 jsou zaznamenány výsledky po změně tohoto parametru.

Most	**Referenční**	**1**	**5**
Sken z kolmic v max. vzdálenosti	866 nových IB	866 nových IB	866 nových IB
1. iterace hledání IB	1727 nových IB	1727 nových IB	1727 nových IB
2. iterace hledání IB	2 nové IB	2 nové IB	2 nové IB
3. iterace hledání IB	0	0	0
Sken IB pro splnění redundance	272 nových IB	0	2246 nových IB
Celkem	2867 IB	2595 IB	4841 IB
Vybráno	813 IB	541 IB	3053 IB
Počet trojúhelníků na začátku	1068	1068	1068
Počet trojúhelníků na konci	4530	4530	4530
Viditelných trojúhelníků	3092	3092	3092
Trojúhelníků splňujících redundanci	2693	3092	2284
Pokryto	68,3 %	68,3 %	68,3 %
Pokryto s redundancí	59,4 %	68,3 %	50,4 %

Tabulka 8.6: Tabulka s výsledky algoritmu po změně počtu překrývajících se snímků u modelu mostu

IB = inspekční bod

Větrná elektrárna	**Referenční**	**1**	**5**
Sken z kolmic v max. vzdálenosti	8227 nových IB	8227 nových IB	8227 nových IB
1. iterace hledání IB	61 nových IB	61 nových IB	61 nových IB
2. iterace hledání IB	0	0	0
3. iterace hledání IB	0	0	0
Sken IB pro splnění redundance	18 nových IB	0	320 nových IB
Celkem	8306 IB	8288 IB	8608 IB
Vybráno	1070 IB	346 IB	2102 IB
Počet trojúhelníků na začátku	13444	13444	13444
Počet trojúhelníků na konci	17554	17554	17554
Viditelných trojúhelníků	15443	15443	15443
Trojúhelníků splňujících redundanci	13859	15443	13198
Pokryto	88 %	88 %	88 %
Pokryto s redundancí	79 %	88 %	75,2 %

Tabulka 8.7: Tabulka s výsledky algoritmu po změně počtu překrývajících se snímků u modelu větrné elektrárny

IB = inspekční bod

Z tabulek je vidět, že při zvolení překryvu snímků z více bodů klesá výsledné pokrytí, které tuto podmínku splňuje. Děje se to hlavně proto, že pro trojúhelníky v hůře dostupných místech už není okolo inspekčního bodu, který tyto trojúhelníky pokryje, místo pro další inspekční body, aby tyto trojúhelníky pokryl a zároveň splnil podmínky, jenž jsou na něj kladeny (viz.: 5.7). Dále je tento parametr přímo úměrný počtu inspekčních bodů obsažených v cestě.

8.4.3 Změna rozlišení kamery

Další parametr, na jehož změnách se vyzkoušelo chování algoritmu je rozlišení. Referenční hodnota byla nastavena na 1280 x 720. V tablukách 8.8 a 8.9 jsou vidět výsledky po změně rozlišení kamery.

Most	Referenční	1920 x 1080	3840 x 2160
Sken z kolmic v max. vzdálenosti	866 nových IB	918 nových IB	681 nových IB
1. iterace hledání IB	1727 nových IB	884 nových IB	688 nových IB
2. iterace hledání IB	2 nové IB	0	28 nových IB
3. iterace hledání IB	0	0	17 nových IB
Sken IB pro splnění redundance	272 nových IB	318 nových IB	99 nových IB
Celkem	2867 IB	2120 IB	1513 IB
Vybráno	813 IB	751 IB	304 IB
Počet trojúhelníků na začátku	1068	1068	1068
Počet trojúhelníků na konci	4530	2500	1546
Viditelných trojúhelníků	3092	2401	1475
Trojúhelníků splňujících redundanci	2693	2144	1427
Pokryto	68,3 %	96 %	95,4 %
Pokryto s redundancí	59,4 %	85,8 %	92,3 %

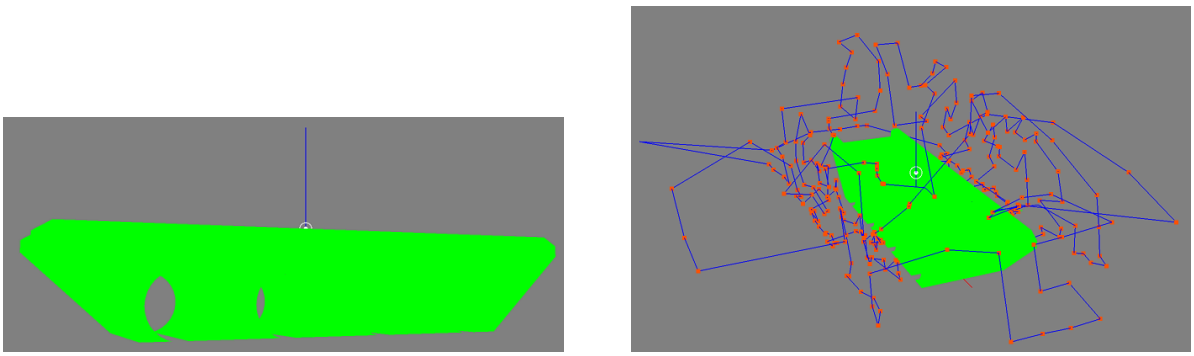
Tabulka 8.8: Tabulka s výsledky algoritmu po změně rozlišení kamery u modelu mostu
IB = inspekční bod

Větrná elektrárna	Referenční	1920 x 1080	3840 x 2160
Sken z kolmic v max. vzdálenosti	8227 nových IB	6701 nových IB	4861 nových IB
1. iterace hledání IB	61 nových IB	32 nových IB	34 nových IB
2. iterace hledání IB	0	0	0
3. iterace hledání IB	0	0	0
Sken IB pro splnění redundance	18 nových IB	6 nových IB	18 nových IB
Celkem	8306 IB	6739 IB	4913 IB
Vybráno	1070 IB	1051 IB	992 IB
Počet trojúhelníků na začátku	13444	13444	13444
Počet trojúhelníků na konci	17554	15631	13603
Viditelných trojúhelníků	15443	14805	12945
Trojúhelníků splňujících redundanci	13859	13715	12167
Pokryto	88 %	94,7 %	95,2 %
Pokryto s redundancí	79 %	87,7 %	89,4 %

Tabulka 8.9: Tabulka s výsledky algoritmu po změně rozlišení kamery u modelu větrné elektrárny
IB = inspekční bod

Z tabulek lze vyčíst, že změna rozlišení má velký vliv na výsledné pokrytí modelu, v případě mostu i na počet inspekčních bodů. Jelikož se zvětšilo rozlišení, tak se tím zvětšila i maximální možná vzdálenost inspekčního bodu od trojúhelníku, takže u mostu nyní kamera mohla pokrýt i trojúhelníky na stěnách oblouku jak je vidět na obrázku 8.4. Zvětšení maximální vzdálenosti také ovlivnilo dělení velkých trojúhelníků, takže po prvotním dělení jich bylo méně, tím pádem i méně inspekčních bodů. Méně inspekčních bodů stačilo také k pokrytí celého objektu, protože body ve větší vzdálenosti nasnímají větší plochu objektu. Zvětšení rozlišení ale negativně ovlivnilo dobu běhu programu. Při každém zvětšení, s testovanými hodnotami rozlišení, se dvojnásobně prodloužila doba výpočtů oproti předchozímu běhu. Opět se u tohoto experimentu vyskytuje snížení celkového pokrytí se zvětšením maximální vzdálenosti (navýšení rozlišení). Vysvětlení je stejné, jako u experimentu se změnou požadované kvality snímku.
Dále lze z tabulek vyčíst, že při zvýšení rozlišení je možné dosáhnout pokrytí 94 - 96 % trojúhelníků. V případě mostu a rozlišení 4K (3840 x 2160) je před prvotním dělením na spodní

straně modelu 30 trojúhelníků, které se ještě mohly dělit na začátku algoritmu. Pokud budeme brát pouze oněch 30 trojúhelníků, tak tím pádem je pokrytých 1475 z 1516 možných trojúhelníků (97,3 %). Pokryto s redundancí je v tomto případě 94,1 % trojúhelníků. Ukázka pokrytí modelu mostu a výsledné trajektorie za použití rozlišení 4K je na obrázku 8.4.



(a) (b)

Obrázek 8.4: Pokrytí modelu mostu a výsledná trajektorie letu při nastaveném rozlišení kamery 4K
zelené trojúhelníky = pokryté, červené body = inspekční body, modré úsečky = cesty mezi inspekčními body

8.5 Simulace

Výsledné trajektorie, získané algoritmem, byly nasimulovány v Gazebo[3] simulátoru. Jde o robotický simulátor sloužící k otestování algoritmů, navrhování robotů, regresnímu testování a trénování systémů umělé inteligence. Simulací prošly výsledky ze všech 4 modelů s referenčním nastavením parametrů algoritmu. Výstupem simulace jsou data o době letu a vzdálenosti, kterou dron během letu urazil. Výsledky jsou vidět v tabulce 8.10. Vizuální ukázky trajektorií, které proběhly simulací jsou v příloze (obrázky 2, 3, 4).
Takové vzdálenosti není možné uletět na jedno nabití baterií, trajektorie by se musely rozdělit na menší.

	Most	**Budova**	**Hangár**	**Vrtule**
Doba letu [min]	44	124	194	75
Délka letu [m]	1556	3908	7641	3317

Tabulka 8.10: Data ze simulace

Výsledná délka letu nekoresponduje s délkami z tabulky 8.3. To proto, že tabulka s ukázkou optimalizace výsledné cesty není v metrech. Jednotka dat v této tabulce je šířka voxelu, která se mění v závislosti na nastavení použitého voxelizátoru.
Simulace proběhla u všech 4 trajektorií bez problému. Ukázka ze simulátoru se nachází v příloze (obrázky 5 a 6).

8.6 Nasazení na reálný bezpilotní prostředek

Zadání práce obsahovalo také nasazení algoritmu na reálný bezpilotní prostředek. K této části bohužel z důvodu vládních nařízení spojených s pandemií COVID-19 nedošlo. Nasazení na reálný

[3] `http://gazebosim.org/`

dron vyžaduje velkou měrou spolupráci s pracovníky AgentFly a proto nebylo žádoucí tento experiment provádět s ohledem na omezení společenských kontaktů. Simulace z předchozí kapitoly plnohodnotně nahrazuje simulaci letu dronu, bohužel neprovádí pořizování snímků v inspekčních bodech.

Kapitola 9

Závěr

V této práci byl představen algoritmus sloužící k nalezení trajektorie pro vizuální inspekci 3D objektů bezpilotním prostředkem.
Na začátku práce jsem se seznámil s problémem a již existujícími řešeními tohoto problému. U těchto přístupů jsem také popsal jejich nedostatky. Dále jsem zadefinoval použité pojmy, řešený problém a výstupu navrženého algoritmu. Poté jsem se seznámil s použitým frameworkem, ve kterém probíhala implementace navrženého algoritmu. V další části práce jsem popsal navržený algoritmus na vyhledávání inspekčních bodů pro ideální pokrytí objektu. Poté jsem popsal způsoby nalezení co nejkratší cesty mezi inspekčními body, ze kterých jsem sestavil výslednou trajektorii pro inspekci bezpilotním prostředkem. Navržený algoritmus jsem naimplementoval a provedl experimenty se změnami nastavení. Nakonec jsem výstup algoritmu otestoval v simulátoru. Zadání bylo tedy splněno.

9.1 Zhodnocení

V práci šlo o co nejlepší pokrytí objektu inspekčními body. Zde velmi záleží na tvaru a členitosti inspektovaného objektu. Jak ukázaly experimenty, tak pokud máme vhodný polygonální model, to znamená, že se některé trojúhelníky navzájem nepřekrývají, je reálně možné pokrýt více než 95 % trojúhelníků, kde nepokryté zůstanou buď velice malé trojúhelníky nebo trojúhelníky o jejichž pokrytí se ani nesnažíme (spodní část modelu, kde dochází ke kontaktu se zemí). To znamená, že celkový pokrytá plocha objektu je ještě větší než 95 %. Pokud se přidá podmínka překryvu snímků, tak lze dosáhnout pokrytí objektu přes 90 %, které bych stále označil za velmi dobré. Ostatní nalezené přístupy většinou neřešily překryv snímků a ani se nesnažily pokrýt hůře dostupná místa objektu.
Samozřejmě že existují objekty, u kterých se některá místa nepokryjí žádným inspekčním bodem. K tomuto může dojít buď z důvodu, že vyhledávací algoritmus nenajde vhodné místo nebo místo není vůbec dostupné a takový inspekční bod, který by toto místo pokryl ani nemůže existovat. Pokud algoritmus pouze nenašel vhodný inspekční bod, tak je možné do výsledné cesty manuálně přidat bod, který by toto místo pokryl. V některých případech stačí pouze zvolit lepší nastavení parametrů algoritmu.
Jak bylo ukázáno v experimentech, tak při hledání inspekčních bodů velmi záleží na zvoleném nastavení kamery. Pokud požadujeme vyšší kvalitu nasnímaného objektu je vhodné zvolit vyšší rozlišení kamery. Obecně, pokud objekt obsahuje hůře dostupná místa pro UAV, tak je vhodné zvolit vyšší rozlišení kamery.

9.2 Problémy zvoleného přístupu

Nejproblematičtější část celého algoritmu je výpočet pokrytí trojúhelníků z inspekčního bodu. I když víme, že se část trojúhelníku nachází v záběru na konkrétním pixelu, tak i v případě že z toho pixelu vyšleme paprsek, nemáme jistotu, že vyslaný paprsek zasáhne chtěný trojúhelník, protože mohl zasáhnout sousední trojúhelník, a tím se nezaznamená jeho nasnímaná plocha. Dále je zde problém výpočtu pokryté plochy pixelem, pokud paprsek dopadne na trojúhelník pod nenulovým úhlem. Proto se v této části práce nachází různé aproximace, přibližně zvolené hodnoty, a i samotný fakt, zda je trojúhelník pokrytý nebo ne se usuzuje na základě toho, zda pokrytá plocha trojúhelníku spadá do určitého intervalu. Tento problém by se dal alespoň z části odstranit tak, že bychom prováděli přepočet nasnímané plochy paprsku dopadajícího na trojúhelník pod jiným než nulovým úhlem u každého vyslaného paprsku zvlášť. To by ale výrazně zhoršilo výpočetní náročnost celého algoritmu. Proto se zvolil přístup aproximace.
Další problém může nastat u vyhledávání inspekčních bodů, kde se zkouší najít lepší bod pouze v 10 směrech od původního bodu, zde může nastat situace, že se ideální inspekční bod vůbec nenajde. Tato chyba se dá ovšem částečně eliminovat vhodným nastavením vstupních parametrů, a to parametry vzdálenosti mezi zkoušenými body a šířce prohledávaného prostoru, stále se ale zkouší hledat pouze v pevně daném počtu směrů.
Další problém je časová a paměťová náročnost algoritmu u modelů s velkým počtem trojúhelníků. Řešením je pustit algoritmus přes noc a přidělit procesu větší množství paměti.
Algoritmus dále neřeší zhoršení světelných podmínek pro nalezení inspekčního bodu, některá místa objektu mohou být ve stínu.

9.3 Možnosti zlepšení práce

Jednou z možností vylepšení práce je místo provedení základního pokrytí objektu inspekčními body ležícími na normálách trojúhelníků v maximální vzdálenosti, vytvořit kolem objektu síť inspekčních bodů v určité vzdálenosti s předem definovanými rozestupy mezi těmito body (aby se snímky částečně překrývaly). Tímto způsobem by se dosáhlo přibližně stejného základního pokrytí objektu, jako v případě této práce. Inspekčních bodů by ovšem ubylo, protože pokud je více malých trojúhelníků vedle sebe, tak algoritmus v této práci vytvoří stejný počet inspekčních bodů, které jsou vedle sebe. Tyto inspekční body pokrývají všechny zmíněné trojúhelníky a zároveň se ani nedají použít ke splnění redundance snímků, protože jsou moc blízko u sebe. Kdyby tyto trojúhelníky pokrýval pouze jediný inspkenčí bod, urychlil by se celý proces hledání a výběru výsledných inspekčních bodů. Pro splnění překryvu snímků by se už hledaly inspekční body jen ke konkrétním trojúhelníkům.
Další způsob vylepšení aktuální implementace je paralelizace vyhledávání inspekčních bodů. Bohužel by šlo o vylepšení časové náročnosti na úkor paměťové náročnosti.

9.4 Navazující práce

Výstup práce je trajektorie pro dron s vyznačenými body pro pořízení snímku. Na toto by mohla navazovat práce, která z těchto snímků, například za použití umělé inteligence, sestaví zpět model s nafocenými texturami.
Další možností by mohla být práce, která ze snímků rozpozná vadu objektu, aniž by musel velké množství snímků procházet odborník, a až snímky s nalezenými vadami předat odborníkovi na posouzení.

Zdroje

[1] I. Corporation, *Intel® Falcon™ 8+ System*, 2020. WWW: https://www.intel.com/content/www/us/en/products/drones/falcon-8.html.

[2] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel a R. Siegwart, "Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics", in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, s. 6423–6430. DOI: 10.1109/ICRA.2015.7140101.

[3] S. Jordan, J. Moore, S. Hovet, J. Box, J. Perry, K. Kirsche, D. Lewis a Z. T. H. Tse, "State-of-the-art technologies for UAV inspections", *IET Radar, Sonar Navigation*, roč. 12, č. 2, s. 151–164, 2018. DOI: 10.1049/iet-rsn.2017.0251.

[4] N. Hallermann a G. Morgenthal, "Visual inspection strategies for large bridges using Unmanned Aerial Vehicles (UAV)", in *Proc. of 7th IABMAS, International Conference on Bridge Maintenance, Safety and Management*, 2014, s. 661–667.

[5] J Fernandez Galarreta, N Kerle a M Gerke, "UAV-based urban structural damage assessment using object-based image analysis and semantic reasoning.", *Natural Hazards & Earth System Sciences*, roč. 15, č. 6, 2015.

[6] T. He, Y. Zeng a Z. Hu, "Research of multi-rotor UAVs detailed autonomous inspection technology of transmission lines based on route planning", *IEEE Access*, roč. 7, s. 114 955–114 965, 2019.

[7] C. Eschmann, C.-M. Kuo, C.-H. Kuo a C. Boller, "Unmanned aircraft systems for remote building inspection and monitoring", in *Proceedings of the 6th European Workshop on Structural Health Monitoring, Dresden, Germany*, sv. 36, 2012, s. 13.

[8] N. Metni a T. Hamel, "A UAV for bridge inspection: Visual servoing control law with orientation limits", *Automation in Construction*, roč. 17, č. 1, s. 3 –10, 2007, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2006.12.010. WWW: http://www.sciencedirect.com/science/article/pii/S0926580507000052.

[9] Z. Shang a Z. Shen, *Indoor Testing and Simulation Platform for Close-distance Visual Inspection of Complex Structures using Micro Quadrotor UAV*, 2019. arXiv: 1904.05271 [cs.RO].

[10] S. Jung, S. Song, S. Kim, J. Park, J. Her, K. Roh a H. Myung, "Toward Autonomous Bridge Inspection: A framework and experimental results", in *2019 16th International Conference on Ubiquitous Robots (UR)*, 2019, s. 208–211. DOI: 10.1109/URAI.2019.8768677.

[11] "Vision-Based Target Finding and Inspection of a Ground Target Using a Multirotor UAV System", *Sensors*, roč. 17, č. 12, s. 2929, 2017, ISSN: 1424-8220. DOI: 10.3390/s17122929. WWW: http://dx.doi.org/10.3390/s17122929.

[12] M. Chloupek, *Plánování trajektorie bezpilotního prostředku pro inspekci 3D objektů*, 2016. WWW: https://dspace.cvut.cz/handle/10467/64848.

[13] P. Novaković, M. Hornak, M. Zachar a N. Joncic, *3D Digital Recording of Archaeological, Architectural and Artistic Heritage*. pros. 2017, ISBN: ISBN 978-961-237-898-1 (pdf). DOI: 10.4312/9789612378981.

[14] *www.scantips.com*. WWW: https://www.scantips.com/lights/fieldofview.html.

[15] P. (www.peckadesign.cz), *Ohnisková vzdálenost*. WWW: https://www.megapixel.cz/ohniskova-vzdalenost.

[16] V. Kaiser, *Efficient Rendering of Earth Surface for Air Traffic Visualization*, 2018. WWW: https://dspace.cvut.cz/handle/10467/73956.

[17] W. Linder, in *Digital photogrammetry: a practical course*. Springer, 2009, 1–16.

[18] M. W. Jones, "The Production of Volume Data from Triangular Meshes Using Voxelisation", *Computer Graphics Forum*, roč. 15, č. 5, s. 311–318, 1996. DOI: https://doi.org/10.1111/1467-8659.1550311. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.1550311. WWW: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.1550311.

[19] H. H. Hoos a T. Stützle, "10 - OTHER COMBINATORIAL PROBLEMS", in *Stochastic Local Search*, ř. The Morgan Kaufmann Series in Artificial Intelligence, H. H. Hoos a T. Stützle, ed., San Francisco: Morgan Kaufmann, 2005, s. 467 –525, ISBN: 978-1-55860-872-6. DOI: https://doi.org/10.1016/B978-155860872-6/50027-5. WWW: http://www.sciencedirect.com/science/article/pii/B9781558608726500275.

[20] J. Demel, *Grafy a jejich aplikace*. Academia, 2002.

[21] D. L. Applegate, *The traveling salesman problem: a computational story*. Princeton Univ. Press, 2007.

[22] J. G. Carbonell, S. Jorg a G. Goos, *Traveling Salesman: Computational Solutions for TSP Applications*. Springer Berlin Heidelberg, 1994.

[23] C. Engels a B. Manthey, "Average-case approximation ratio of the 2-opt algorithm for the TSP", *Operations Research Letters*, roč. 37, č. 2, s. 83 –84, 2009, ISSN: 0167-6377. DOI: https://doi.org/10.1016/j.orl.2008.12.002. WWW: http://www.sciencedirect.com/science/article/pii/S016763770900011X.

[24] J. Bresenham, "Algorithm for Computer Control of a Digital Plotter", *IBM Syst. J.*, roč. 4, s. 25–30, 1965.

[25] H. De Beukelaer, G. F. Davenport, G. De Meyer a V. Fack, "JAMES: An object-oriented Java framework for discrete optimization using local search metaheuristics", *Software: Practice and Experience*, roč. 47, č. 6, s. 921–938, 2017. DOI: https://doi.org/10.1002/spe.2459. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2459. WWW: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2459.

Příloha 1

V této příloze se nachází dodatečné tabulky a obrázky.

1 Výpočet plochy pokryté pixelem

Dodatečné tabulky ve kterých jsou mezivýpočty použité k aproximaci funkce u výpočtu plochy, kterou pokrývá 1 pixel.

α / vzdálenost	4	5	6	7	8	9	10	11
0		1,0357	1,0177	1,0294	1,0310	1,0331	1,0407	1,0463
10		1,0123	1,0127	1,0141	1,0212	1,0242	1,0009	1,0151
20	0,8967	0,9355	0,9409	0,9467	0,9563	0,9652	0,9741	0,9780
30	0,8216	0,8493	0,8540	0,8612	0,8682	0,8761	0,8818	0,8870
40	0,7116	0,7353	0,7397	0,7478	0,7503	0,7586	0,7679	0,7753
50			0,6087	0,6167	0,6212	0,6236	0,6296	0,6350
60			0,4658	0,4708	0,4754	0,4794	0,4829	0,4831
70					0,3200	0,3225	0,3246	0,3262

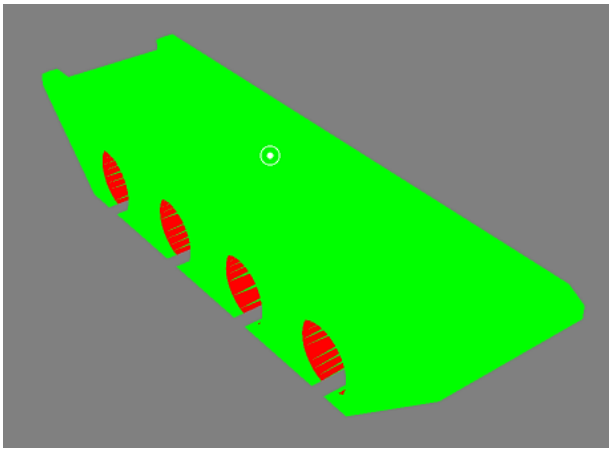
Tabulka 1: Nasnímaná plocha většího trojúhelníku pod různými úhly a v různé vzdálenosti od těžiště trojúhelníku s přičtenou plochou

α / vzdálenost	4	5	6	7	8	9	10	11
0		1,0042	0,9786	0,9626	0,9732	0,9790	0,9864	0,9924
10		0,9721	0,9766	0,9624	0,9681	0,9707	0,9781	0,9988
20	0,8814	0,9244	0,9264	0,9319	0,9479	0,9488	0,9582	0,9649
30	0,8214	0,8769	0,8770	0,8901	0,8941	0,9057	0,9105	0,9158
40	0,7116	0,8033	0,8116	0,8135	0,8210	0,8278	0,8300	0,8323
50			0,7107	0,7129	0,7130	0,7240	0,7306	0,7271
60			0,5765	0,5790	0,5857	0,5858	0,5952	0,5942
70					0,4158	0,4200	0,4247	0,4249

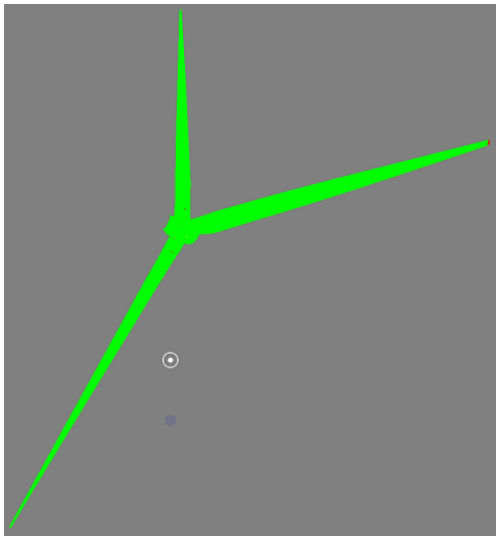
Tabulka 2: Nasnímaná plocha menšího trojúhelníku pod různými úhly a v různé vzdálenosti od těžiště trojúhelníku s přičtenou plochou

2 Výsledné pokrytí

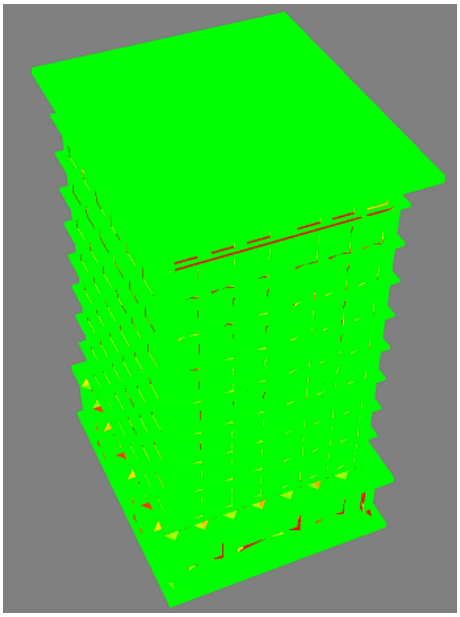
Dodatečné obrázky pokrytí objektu inspekčními body s referenčním nastavením parametrů algoritmu.



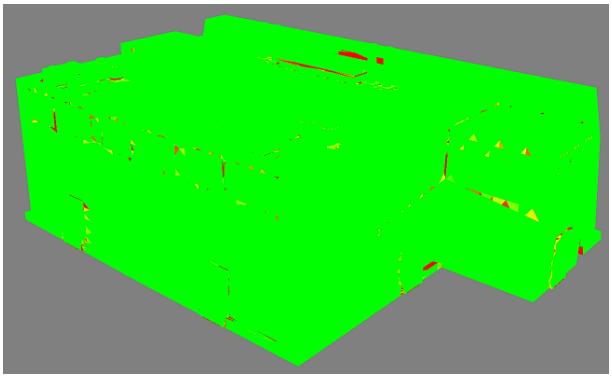
(a) Most



(b) Větrná elektrárna



(c) Budova

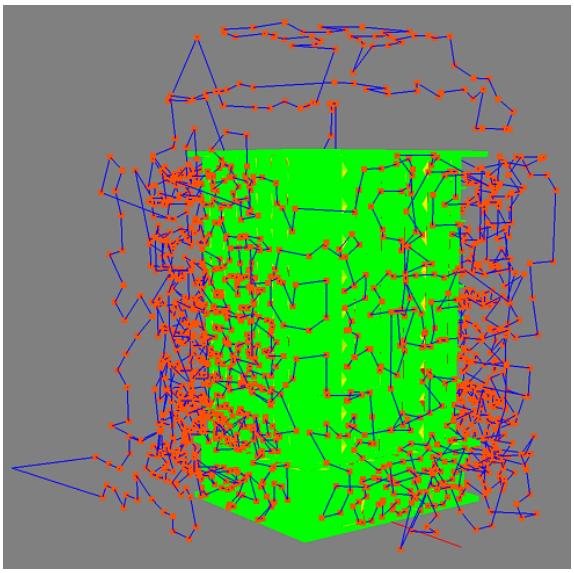


(d) Hangár

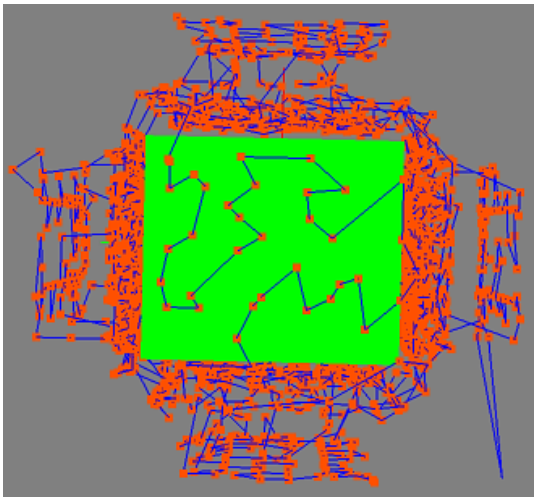
Obrázek 1: Dodatečné obrázky výsledného pokrytí modelů inspekčními body s referenčním nastavením algoritmu
červené trojúhelníky = nepokryté, žluto-oranžové trojúhelníky = částečně pokryté, zelené trojúhelníky = pokryté

3 Výsledné trajektorie

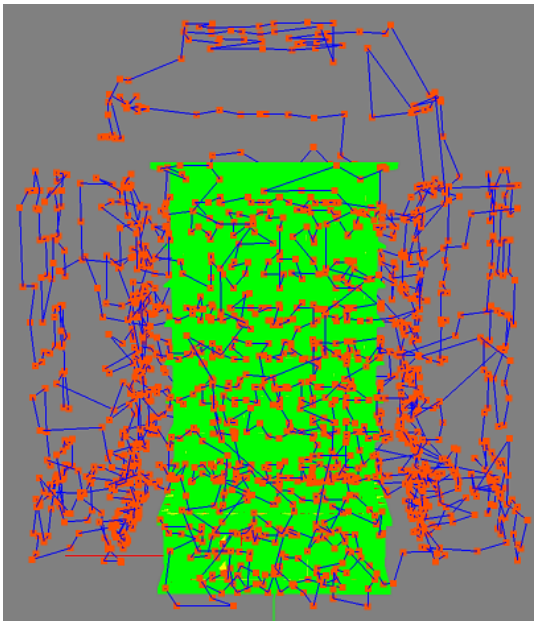
Dodatečné obrázky vizualizace výsledných trajektorií pro testované modely s referenčním nastavením parametrů algoritmu.



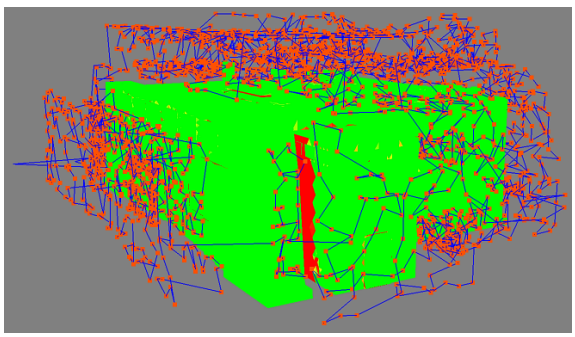
(a) Budova



(b) Budova



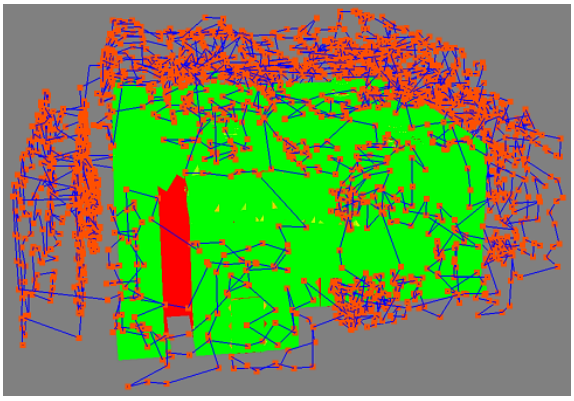
(c) Budova



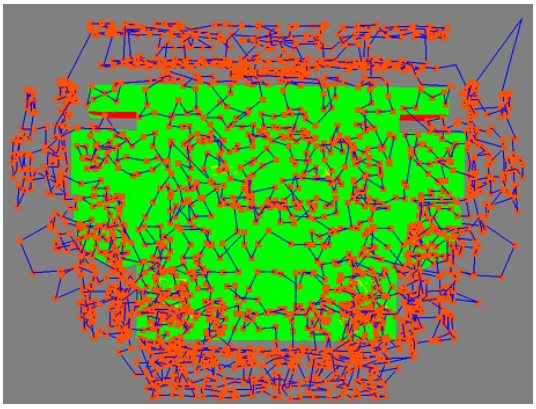
(d) Hangár

Obrázek 2: Dodatečné obrázky výsledných trajektorií pro dron s inspekčními body pro modely s referenčním nastavením algoritmu

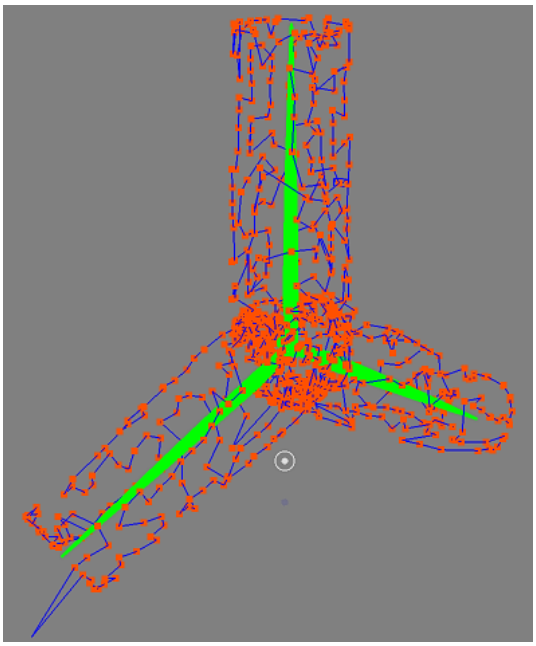
červené body = inspekční body, modré úsečky = cesty mezi body, červené trojúhelníky = nepokryté, žluto-oranžové trojúhelníky = částečně pokryté, zelené trojúhelníky = pokryté



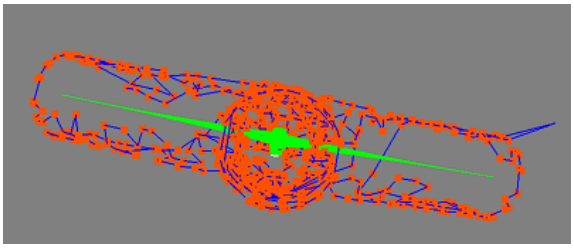
(a) Hangár



(b) Hangár



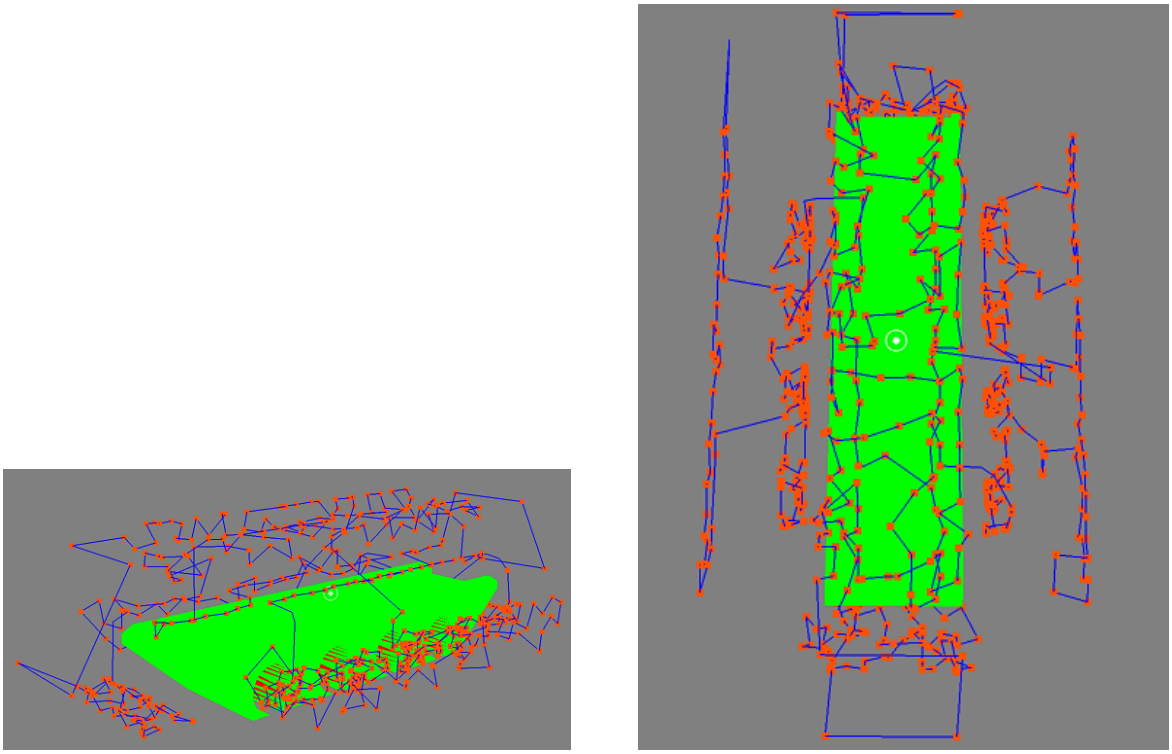
(c) Větrná elektrárna



(d) Větrná elektrárna

Obrázek 3: Dodatečné obrázky výsledných trajektorií pro dron s inspekčními body pro modely s referenčním nastavením algoritmu

červené body = inspekční body, modré úsečky = cesty mezi body, červené trojúhelníky = nepokryté, žluto-oranžové trojúhelníky = částečně pokryté, zelené trojúhelníky = pokryté

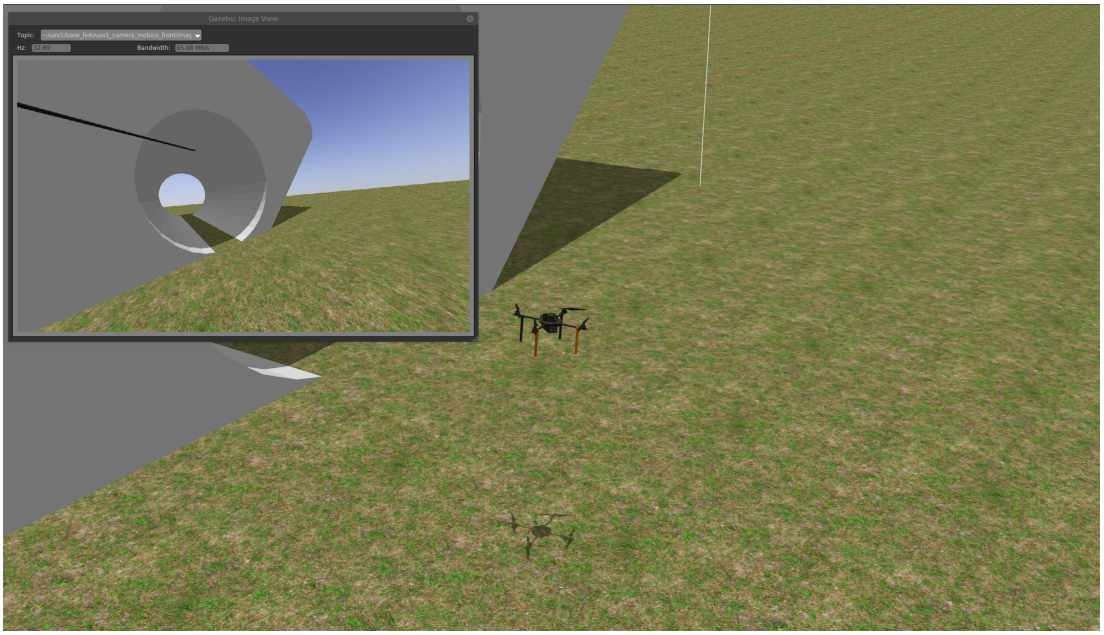


(a) Most (b) Most

Obrázek 4: Dodatečné obrázky výsledných trajektorií pro dron s inspekčními body pro modely s referenčním nastavením algoritmu
červené body = inspekční body, modré úsečky = cesty mezi body, červené trojúhelníky = nepokryté, žluto-oranžové trojúhelníky = částečně pokryté, zelené trojúhelníky = pokryté

4 Simulace

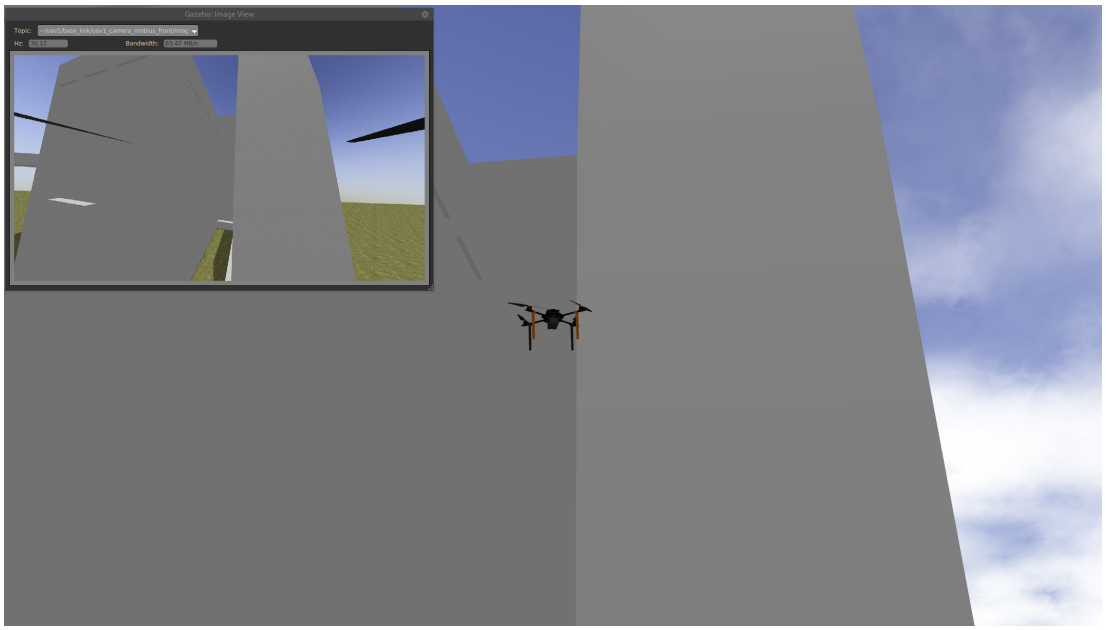
Obrázky ze simulace prolétávání výsledných trajektorií dronem.



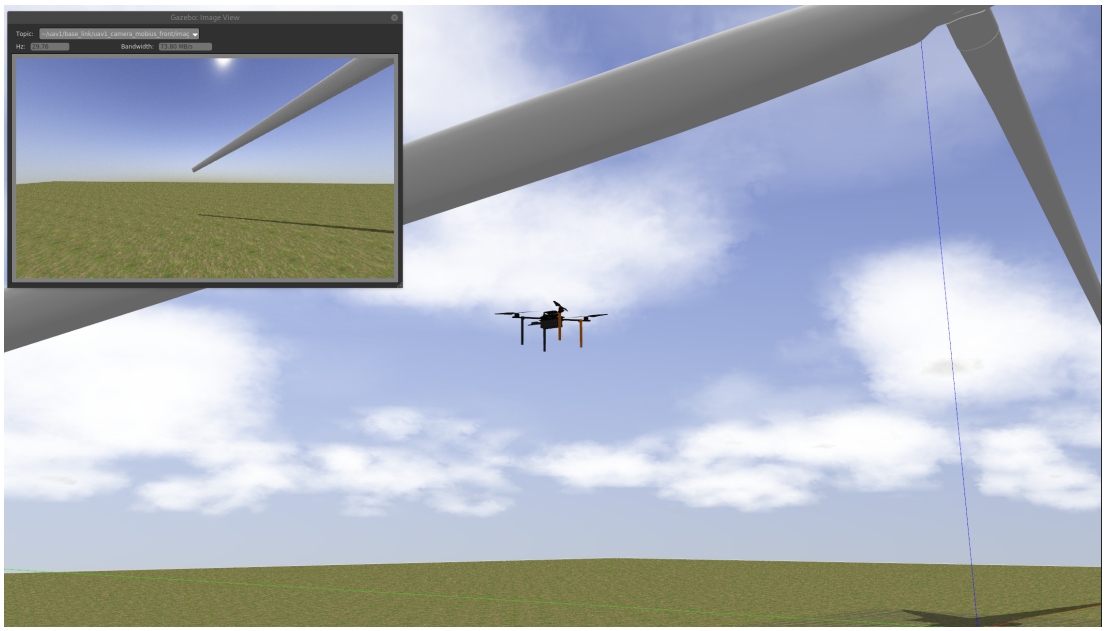
(a)

(b)

Obrázek 5: Ukázka ze simulace letu výslednou trajektorií na modelu mostu a budovy



(a)



(b)

Obrázek 6: Ukázka ze simulace letu výslednou trajektorií na modelu hangáru a větrné elektrárny

Příloha 2

5 Obsah odevzdané přílohy

Odevzdaná příloha obsahuje komprimovaný (.zip) Maven projekt s implementací navrženého algoritmu za použití AgentFly frameworku. Odevzdaný kód je možné naimportovat do vývojového prostředí. Zkompilovat a spustit jde ale pouze v případě, že má uživatel povolený přístup do AgentFly repozitáře s knihovnami. V projektu se nachází i testované modely.