VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PLÁNOVÁNÍ CESTY PRO AUTONOMNÍ ZEMĚDĚLSKÉ STROJE

PATH PLANNING FOR AUTONOMOUS AGRICULTURAL MACHINES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. LUKÁŠ KUCHTA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

142829

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Kuchta Lukáš, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Softwarové inženýrství
Název:	**Plánování cesty pro autonomní zemědělské stroje**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Nastudujte algoritmy pro plánování cesty. Zaměřte se na algoritmy používané pro kompletní pokrytí daného prostoru (tzv. coverage path planning). Konkrétně se pak zaměřte na metody, jak správně naplánovat cestu pro zemědělský stroj s ohledem na sklon pole a vstupní a výstupní body.
2. Navrhněte program, který umožní vytvořit nebo z externího souboru nahrát tvar oblasti a pro ni vytvořit plán cesty. Počítejte s omezeními reálného stroje, tzn. šířka záběru, poloměr otáčení, maximální rychlost při práci a při přejíždění a zrychlení/zpomalení.
3. Trasy navrhněte vzhledem k různým parametrům jako je čas, ujetá vzdálenost, případně spotřeba.
4. Navržený program implementujte a to včetně vizualizace trasy a otestujte jeho funkčnost. Program otestujte i pro více oblastí, mezi kterými bude nutné přejet.

Literatura:

- Howie Choset et al., Principles of Robot Motion, 2005, ISN 0-262-03327-5.
- Rusnák Jakub, Vizualizace algoritmů pro plánování cesty, bakalářská práce, FIT VUT v Brně, 2017.

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	3.11.2022

Abstrakt

So stále rastúcou populáciou ľudí na Zemi je potrebné zabezpečiť, aby rástla aj oblasť poľnohospodárstva, aby bol zabezpečený dostatok potravy pre všetkých ľudí. Jednou z možností, ako to zabezpečiť je zvýšenie automatizácie v tejto oblasti. Keďže sa stále menej ľudí venuje poľnohospodárstvu, tak sú stále častejšie využívané viac alebo menej autonómne zariadenia. Táto práca sa zaoberá zhrnutím algoritmov, ktoré sú používané na plánovanie trasy (nie len) v oblasti poľnohospodárstva, čo je jednou z kľúčových súčastí autonómnych vozidiel. Najskôr je zhrnutá história v tejto oblasti, následne súčasný stav autonómnych vozidiel v poľnohospodárstve. V ďalšej časti sú zhrnuté algoritmy typu point-to-point, za ktorými nasledujú algoritmy typu coverage routing. Následne sa zaoberáme využitím týchto algoritmov v praxi. V ďalšej časti je popísaný návrh aplikácie, ktorá je následne implementovaná. Je vytvorený program, ktorý pre zadané pole a stroj vytvorí mapu pokrytia tohto poľa. Nakoniec sú zhrnuté výsledky práce.

Abstract

With the ever-growing human population on Earth, it is necessary to ensure that the agricultural sector also grows in order to ensure enough food for all people. One way to ensure this is to increase automation in this area. Since fewer and fewer people are engaged in agriculture, more or less autonomous devices are being used more often. This work deals with a summary of algorithms that are used for route planning (not only) in the field of agriculture, which is one of the key components of autonomous vehicles. First, the history in this area is summarized, followed by the current state of autonomous vehicles in agriculture. Next, point-to-point algorithms are summarized, followed by coverage routing algorithms. The next part deals with the use of these algorithms in practice. The next part describes the design of the application, which is implemented in the next part of this work. A program is created that creates a coverage map for the specified field and machine. Finally, the results of the work are summarized.

Kľúčové slová

plánovanie cesty, algoritmy plánovania cesty, autonómne poľnohospodárske stroje, plánovanie trasy pokrytím

Keywords

path planning, path planning algorithms, autonomous agricultural machines, coverage path planning

Citácia

KUCHTA, Lukáš. *Plánování cesty pro autonomní zemědělské stroje*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Plánování cesty pro autonomní zemědělské stroje

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Jaroslava Rozmana, Ing., Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Lukáš Kuchta
15. mája 2023

Poďakovanie

Rád by som využil túto príležitosť na poďakovanie vedúcemu práce pánovi Jaroslavovi Rozmanovi, Ing., Ph.D. za pomoc pri riešení práce a pri hľadaní zdrojov a taktiež by som sa rád poďakoval autorom publikácií, z ktorých som čerpal.

Obsah

Kapitola 1

Úvod

Koncom roku 2022 dosiahla populácia na Zemi hodnotu 8 miliárd ľudí a očakáva sa, že bude ďalej rýchlo rásť. Aj keď sa využívanie pôdy určenej na pestovanie plodín blíži k maximu, je potrebné zvýšiť produkciu plodín, aby boli zabezpečené potraviny pre všetkých ľudí. Je to jedna z najdôležitejších vecí na zabezpečenie pokoja vo svete. Je potrebné skúmať efektívnejšie metódy pestovania plodín, to zahŕňa napríklad využívanie menšieho počtu ľudí, zvýšenie automatizácie pri prácach na poli alebo na farmách, zvýšenie množstva a kvality pestovaných plodín a podobne. Termín precízne poľnohospodárstvo (precition agriculture) označuje skupinu techník na zabezpečenie manažovania polí na zvýšenie výnosu plodín, profit firiem a na udržateľnosť ekosystému. Precízne poľnohospodárstvo už poskytlo množstvo riešení na zabezpečenie týchto cieľov a je považované za kľúčovú stratégiu pre optimalizovanie činností súvisiacich s pestovaním plodín a garantovanie enviromentálnej bezpečnosti.

Vzhľadom na to, že ľudí pracujúcich v poľnohospodárstve po celom svete stále ubúda, je potrebné začať s využívaním autonómnych poľnohospodárskych strojov a to hlavne na veľkých farmách, ktoré majú málo personálu. Takéto stroje dokážu na rozdiel od ľudí pracovať takmer neprestajne, čo veľmi zefektívni prácu a hlavne urýchli dôležité činnosti ako je príprava polí na sejbu, sejba, distribúcia hnojív, ničenie škodcov, alebo zber plodín. Veľmi dôležitou oblasťou je plánovanie cesty pre tieto autonómne poľnohospodárske stroje. Pre čo najväčšiu efektivitu týchto strojov je dôležitá vysoká úroveň automatizácie a čo najmenej zásahov obsluhy. Aby to bolo možné, je potrebné mať spoľahlivú navigáciu a jej hlavnou časťou je plánovanie trasy. To je zodpovedné za vytvorenie cesty medzi zvolenými bodmi vrátanie vyhýbania sa prekážkam. Vozidlo však musí byť pripravené aj na neočakávané udalosti, ktoré sa môžu vyskytnúť v reálnom čase. Polia môžu mať komplexný tvar, obsahovať množstvo rôznych prekážok, mať určitý sklon a všetky tieto faktory je potrebné brať do úvahy pri plánovaní cesty. Metódy využívané v interiéri, alebo v doprave nemusia byť vhodné pre využitie v poľnohospodárstve. Preto je potrebné vymyslieť algoritmus, ktorý sa bude dať použiť pri plánovaní trasy pre autonómne poľnohospodárske stroje a bude brať do úvahy topológiu poľa, prekážky na ňom a taktiež obmedzenia stroja ako jeho polomer otáčania, či šírku pokrytia konkrétnym príslušenstvom.

Kapitola 2

Algoritmy pre plánovanie cesty

Aby sme sa mohli zaoberať algoritmami pre plánovanie cesty, musíme si zadefinovať, čo to je algoritmus a čo znamená. Algoritmus [7][8] je postupnosť presne definovaných inštrukcií na splnenie určitej úlohy. Algoritmus je elementárnym pojmom informatiky – nie je ho možné popísať pomocou ešte elementárnejších pojmov. Algoritmizácia je schopnosť aktívne vytvárať algoritmy určené pre nemysliace zariadenie. Je nevyhnutná pri vytváraní počítačových programov. Program je algoritmus napísaný v programovacom jazyku. Vzhľadom na svoju dlhú históriu predstavuje slovo algoritmus veľmi všeobecný pojem, ktorý nemá jednoznačnú formálnu definíciu. Preto sa jeho súčasná neformálna definícia v informatike obmedzuje vlastnosťami, ktoré musí spĺňať, aby sa dali algoritmy podrobiť vedeckému skúmaniu. Neexistuje jednoznačná zhoda na jedinej správnej formálnej definícií, a preto existuje niekoľko rôznych neformálnych charakterizácií. Niekedy sa tieto definície líšia len zápisom, niektoré vlastnosti sú spojené, inak nazvané, alebo zahrnuté do samotnej definície pojmu algoritmus. Algoritmus môže byť definovaný ako „konečná množina pravidiel, ktorá popisuje postupnosť operácií na riešenie určitého typu problému“. Podľa Donalda Knutha [25] zároveň spĺňa nasledovné vlastnosti:

Konečnosť: Každý algoritmus musí skončiť po vykonaní konečného počtu krokov. Tento počet krokov môže byť ľubovoľne veľký (podľa rozsahu a hodnôt vstupných údajov), ale pre každý jednotlivý vstup musí byť konečný.

Determinovanosť: Každý krok algoritmu musí byť jednoznačne a presne definovaný. V každej situácii musí byť úplne zrejmé, čo a ako sa má vykonať, ako má vykonávanie algoritmu pokračovať. Pretože bežný jazyk zvyčajne neposkytuje úplnú presnosť a jednoznačnosť vyjadrovania, boli pre zápis algoritmov navrhnuté programovacie jazyky, v ktorých má každý príkaz jasne definovaný význam. Vyjadrenie algoritmu v programovacom jazyku sa nazýva program.

Vstup: Algoritmus zvyčajne pracuje s nejakými vstupmi, veličinami, ktoré sú mu odovzdané pred začatím jeho vykonávania, alebo v priebehu jeho činnosti. Vstupy majú definované množiny hodnôt, ktoré môžu nadobúdať.

Výstup: Algoritmus má aspoň jeden výstup, veličinu, ktorá je v požadovanom vzťahu k zadaným vstupom, a tým tvorí odpoveď na problém, ktorý algoritmus rieši.

Efektivita: Aby algoritmus bol efektívny, každá operácia požadovaná algoritmom by mala byť dostatočne jednoduchá na to, aby mohla byť aspoň v princípe prevedená v konečnom čase iba s použitím ceruzky a papiera. Iná definícia vraví, že že efektivita

znamená to, že výpočet sa uskutočňuje v čo najkratšom čase a s využitím čo najmenšieho množstva prostriedkov (časových i pamäťových).

Každý algoritmus začína zo svojho počiatočného stavu a so svojim vstupom, ktorý môže byť aj prázdny. Jednotlivé inštrukcie popisujú postup výpočtu tak, že sa vždy dostaneme do istého definovaného stavu. Týchto stavov je konečný počet. Prechod z jedného stavu nemusí byť vždy deterministický, randomizované algoritmy napríklad počítajú s náhodnými vstupmi. Nakoniec sa každý algoritmus dostane do určitého koncového stavu a dostaneme istý výstup. Algoritmy môžu byť zapísané v rôznej forme, napríklad v prirodzenom jazyku, pseudokóde, vývojovom diagrame, riadiacej tabuľke, programovacom jazyku... Problém vyjadrovania algoritmov v prirodzenom jazyku je ten, že jeho interpretácia nie je jednoznačná a takýto zápis sa takmer nepoužíva pre zápis zložitejších algoritmov. Vyjadrenie algoritmu v pseudokóde alebo pomocou vývojového diagramu je štrukturované a vyhýba sa nejednoznačnostiam, ktoré sa vyskytujú pri použití prirodzeného jazyka. Programovacie jazyky sú určené primárne na vyjadrenie algoritmov vo forme, v ktorej je možné ich vykonať počítačom, ale často sa využívajú aj na dokumentáciu týchto algoritmov. Pri vytváraní algoritmu je nutné dodržiavať postupné kroky:

1. **Návrh**: Prvým krokom je identifikácia problému a jeho pochopenie. V tomto momente je potrebné sa o danom probléme rozprávať s ľuďmi, ktorí mu rozumejú, majú o ňom nejaké informácie, skúsenosti s ním, alebo s jeho časťou, prípadne či pre daný problém alebo jeho časť nepoznajú riešenie. Po získaní všetkých informácií je nutné problém rozdeliť na menšie časti, s ktorými sa bude ľahšie pracovať. V tejto časti návrhu sa často využívajú rôzne nákresy, diagramy a pseudokódy, aby sa čo najskôr odchytili a opravili prípadné chyby. Pri návrhu môžme postupovať:

   Zhora dole: postup riešenia rozkladáme na menšie časti, až sa dostaneme na elementárne operácie,

   Zdola hore: z elementárnych operácie tvoríme väčšie celky, ktorých spojením nakoniec dospejeme k riešeniu problému,

   Kombináciou oboch: obvyklý postup zhora dole doplníme krokmi zdola hore tým, že využijeme funkcie z rôznych knižníc, alebo využijeme programovací jazyk vyššej úrovne.

2. **Analýza**: Ak už máme predstavu, ako bude algoritmus fungovať, je dobré zistiť, aký je efektívny na vyriešenie daného problému. Tento krok je možné vykonať ešte pred naprogramovaním algoritmu, aby sa overilo, či algoritmus spĺňa očakávania, ktoré vyplývajú z fázy návrhu, alebo je možné tento krok vykonať až po nakódovaní algoritmu a jeho spustení. Pri analýze algoritmov sa skúma ich časová a pamäťová zložitosť. Je dôležité skúmať, ako sa tieto zložitosti menia s rastúcimi dátami a dátovými štruktúrami, ktoré sa v nich používajú. Niekedy však môže byť náročné zistiť, kedy náš algoritmus dosiahol maximálnu efektivitu.

3. **Implementácia**: Ďalšou časťou tvorby algoritmu je jeho nakódovanie v určitom programovacom jazyku. Pre každý algoritmus môže byť vhodnejší iný programovací jazyk. Záleží na tom, či má byť vykonaný čo najrýchlejšie, má využívať čo najmenej pamäte, alebo má byt implementovaný čo najrýchlejšie. Dôležité je aj to, aby programátor, ktorý daný algoritmus implementuje, dobre rozumel konkrétnemu programovaciemu jazyku a presne vedel, čo ktorý riadok kódu vykoná. V neposlednom rade je dôležité

kód aj dobre komentovať a dokumentovať, aby sa v ňom autor vyznal aj vtedy, ak by chcel kód po dlhšej dobe refaktorizovať, alebo ak by s týmto kódom v budúcnosti pracoval aj niekto iný.

4. **Testovanie a experimentovanie**: Keď je už algoritmus navrhnutý a naimplementovaný v nejakom programovacom jazyku, mal by sa dobre otestovať, a to vkladaním rôznych dát, či už očakávaných, alebo neočakávaných a takto nájsť prípadné chyby. Pri tomto procese je možné aj testovať kód s rôzne veľkými vstupmi a prípadne prísť na lepší spôsob návrhu alebo implementácie tohto algoritmu, aby bola dosiahnutá čo najlepšia efetkivita.

Pri návrhu algoritmov sa uplatňuje množstvo prístupov, ktoré nám pomáhajú s abstrakciou konkrétnej úlohy. K najužitočnejším metódam patrí:

Divide and conquer (rozdeľ a panuj): Ako názov napovedá, tento prístup rekurzívne rozdelí problém na menšie podproblémy, ktoré sa dajú vyriešiť priamo a následne je pomocou nich možné vyriešiť aj pôvodný problém. Je to najčastejšie používaný prístup pri návrhu algoritmov.

Greedy algorithm (hladný algoritmus): Priamočiary prístup na riešenie určitej triedy optimalizačných úloh. Spracováva sa množina V zložená z n údajov a cieľom je nájsť takú podmnožinu W množiny V, ktorá vyhovuje určitým podmienkam a pritom optimalizuje predpísanú účelovú funkciu[1]. Akákoľvek podmnožina W sa nazýva prípustné riešenie, ak toto riešenie pre účelovú funkciu nadobúda optimálnu hodnotu, riešenie je optimálne.

Dynamic programming (dynamické programovanie): Používa sa, ak je možné nájsť riešenie problému pomocou jednoduchších problémov, ktoré sa v riešení opakujú. Namiesto toho, aby sme problém rozbili na menšie podproblémy, priebežné výsledky sa ukladajú a je možné ich použiť v nasledujúcich operáciách. Oproti metóde Divide and conquer to má výhodu v tom, že je väčšinou rýchlejšie získať predtým vypočítaný výsledok, ako ho znova počítať.

2.1 História a súčasný stav plánovania cesty v poľnohospodárstve

Prvé pokusy o vytvorenie autonómneho traktora sa objavili už v roku 1940, kedy Frank W. Andrew vytvoril svoj vlastný. Bol to veľmi jednoduchý koncept, kde išlo o to, že sa doprostred poľa umiestnil nejaký pevný bod, sud, alebo koleso a okolo neho sa namotal kábel pripevnený k riadeniu v prednej časti traktora. Ďalší pokus o vytvorenie autonómneho traktora sa objavil v 50. rokoch 20. storočia a to firmou Ford, avšak tento stroj sa nikdy nedostal do produkcie, pretože vyžadoval, aby bol v zemi v poli natiahnutý kábel na ovládanie tohto stroja. Potom nastalo na dlhší čas hluché obdobie a to až do roku 1994, keď ľudia zo Silsoe Research Institute vyvinuli systém na analýzu obrazu, ktorý bol využitý na navádzanie malého autonómneho traktora pri pestovaní zeleniny a koreňových plodín, ktorý sa dokázal aj otočiť na konci riadkov. V posledných rokoch sa sa v tejto oblasti využívajú hlavne tieto prístupy:

[1] Pre danú úlohu sa snažíme maximalizovať/minimalizovať hodnotu tejto funkcie

Autonómne vozidlá pod dozorom: Tieto vozidlá využívajú "vehicle-to-vehicle" technológiu. V podstate ide o to, že dva stroje idú za sebou, vykonávajú rovnakú prácu, v prvom stroji sa nachádza obsluha a druhý je s ním bezdrôtovo spojený, takže si môžu vymieňať dáta a tento autonómny stroj potom plní príkazy (napodobňuje jazdu) obsluhy v prvom stroji.
Plne autonómne vozidlá: Väčšina autonómnych vozidiel na poliach využíva lasere, ktoré odrážajú signály od viacerých mobilných transpondérov, ktoré sú umiestnené okolo poľa. Táto technika sa však nepoužíva na veľkých poliach a strojoch, skôr na tých menších. Namiesto obsluhy stroja sa v ňom nachádzajú kontroléry, vďaka ktorým je možné na stroj dozerať vzdialene. To znamená, že obsluha môže dozerať na viacero strojov bez toho, aby sa v nich niekto nachádzal.

Existuje niekoľko hlavných výrobcov, ktorí sa aktívne snažia o výrobu plne autonómnych poľnohospodárskych strojov a v posledných rokoch spravili významné kroky v začatiu masovej výroby týchto strojov. Súčasnými poprednými výrobcami sú John Deere, Autonomous Tractor Corporation, Fendt a Case IH.
John Deere: Táto firma je jedna z najväčších výrobcov poľnohospodárskej techniky a vynakladá veľké úsilie na vývoj v oblasti autonómnych strojov. Začiatkom roku 2008 uviedli na trh produkt s názvom ITEC Pro guidance, čo je systém založený na GPS lokácií, ktorý zabezpečuje autonómne riadenie vozidla a to vrátane vykonávania otočiek na konci riadkov. Namiesto laserov využíva 2 antény, ktoré komunikujú s GPS satelitmi. Na základe signálu zo satelitov stroje využívajúce tento systém dokážu ísť po vopred naprogramovanej ceste pomocou elektronickej mapy. Tieto antény umiestnené na stroji slúžia aj pre operátorov, v prípade, ak sila signálu je slabá z dôvodu hustej vegetácie alebo okolitých budov. Táto firma však vyvíja aj plne autonómny traktor bez kabíny pre vodiča.
Autonomous Tractor Corporation: Táto pomerne mladá spoločnosť bola založená v januári 2012 Terrym Andersonom, ktorý sa rozhodol vrátiť z dôchodku a vyrobiť autonómny traktor. Spočiatku sa snažili o výrobu traktora, ktorý bude nasledovať vedúci traktor s obsluhou, ale časom sa presunuli smerom k vývoju a výrobe autonómnych strojov. Ich produkt sa nazýva SPIRIT a je to autonómny traktor. Táto spoločnosť za dobu svojej krátkej existencie stihla vyhrať rôzne ceny.
Fendt: Tento nemecký výrobca poľnohospodárskej techniky predstavil v roku 2011 na výstave v Hannoveri svoj model traktora bez vodiča s názvom GuideConnect, ktorý je naprogramovaný tak, aby nasledoval iný traktor, v ktorom je operátor, tak ako je to popísané vyššie. Namiesto toho, aby sa zamerali na plne autonomné stroje, spoločnosť vytvorila GuideConnect, aby spolupracoval so strojmi ovládanými obsluhou a komunikoval s ňou pomocou GPS a rádia.
Case IH: Táto spoločnosť vznikla spojením J.I. Case Company a International Harvester. Podobne ako spoločnosť Fendt, aj Case IH má stroje, ktoré autonómne nasledujú stroj vedený obsluhou, tieto stroje sa nazývajú ako "supervised autonomy". V roku 2016 však predstavili koncept traktoru bez kabíny, ktorý by mal byť plne autonómny [9].

2.2 Metodológia

Táto práca sa opiera o štúdiu *A Comprehensive Review of Path Planning for Agricultural Ground Robots* [13], ktorá sa zaoberá vymenovaním a zhodnotením jednotlivých prác v oblasti plánovania trasy pre autonómne poľnohospodárske stroje. Využívala metódu systematického prehľadávania literatúry na skúmanie existujúcich článkov a prác na túto tému s cieľom poskytnúť nové informácie v tejto oblasti. Pomocou vedeckého vyhľadávača Google

Obr. 2.1: Popisované spoločnosti a ich riešenia, postupne zľava John Deere [5], Autonomous Tractor Corporation [1], Fendt [4], Case IH. [2]

Schoolar bol zostavený zoznam najrelevantnejšej literatúry zaoberajúcej sa touto oblasťou. Za posledných pár rokov výrazne stúpol záujem o túto tému. Autori článku zistili, že väčšina z týchto článkov/štúdií sa však zaoberalo vzdušnými dopravnými prostriedkami. Taktiež sa tam často objavovali články zaoberajúce sa plánovaním pohybu (motion planning), ktoré sú často zmieňované v literatúre, pokiaľ ide o automatizované činnosti, ktoré súvisia s robotikou, ako napríklad problémy koordinácie objektov, kinematické obmedzenia a rôzne ďalšie faktory. Rovnako tak tam boli aj články na tému plánovaním trasy (route planning), ktoré sa zaoberali hlavne problémom najoptimálnejšieho prechodu grafu. Na druhej strane, články na tému plánovania trasy (path planning), ktoré nás v tejto práci zaujímajú sa zaoberajú nájdením trasy z počiatočného bodu do cieľového bodu vrátane vyhýbania sa prekážkam. Autori článku vybrali niekoľko zaujímavých článkov z rôznych oblasti a skúmali nasledujúce otázky:

- Aká poľnohospodárska činnosť sa vykonáva?
- Aká technika plánovania cesty sa používa?
- Je vyhodnocovanie trasy on-line?
- Dynamické alebo statické vyhodnocovanie
- Je cesta optimálna?

- Charakteristiky geometrie
- Kritéria optimalizácie
- Obmedzenia stroja
- Limitácie
- Výpočtová náročnosť a doba spracovania
- Bolo vykonané testovanie v teréne?

Existujú dva spôsoby, na základe ktorých sa implementuje plánovanie trasy: lokálny polohovací systém (local positioning system), ktorý je založený na senzoroch na danom vozidle, alebo globálny polohovací systém (global positioning system - GPS), ktorý je založený na komunikácií so satelitmi. Od 70. rokov 20. storočia sa využíval hlavne lokálny polohovací systém, nebol až tak nákladný, čo sa týka investície, bolo možné ho využívať hlavne v interiéri, ale nie je vhodné ho použiť vo vonkajšom prostredí. S vylepšením satelitnej komunikácie sa začalo prechádzať na GPS a prinieslo veľa výhod, ako napríklad menej práce pre obsluhu stroja, ktorý sa nemusí až tak sústrediť na presné navádzanie, vylepšila sa presnosť v rôznych činnostiach, ako je orba, sejba, distribúcia hnojív, alebo zber plodín. Taktiež to pomohlo pri práci pri zlej viditeľnosti, alebo v noci. V ďalšej časti budú rozoberané dva hlavné prístupy na plánovanie trasy: point-to-point routing a coverage routing.

Kapitola 3

Point-to-point routing

Cieľom plánovania cesty použitím prístupu point-to-point je nájdenie cesty bez kolízií z počiatočného bodu do cieľového bodu a to pri minimalizácii času, vzdialenosti a spotreby energie. Tento prístup je popísaný v článku Spatial Planning: A Configuration Space Approach [27] od autora Tomása Lozano-Péreza, ktorý definuje stroj alebo robota za jeden bod, pričom cieľový bod ho priťahuje, zatiaľ čo rôzne prekážky ho odpudzujú. Poľnohospodárske polia sú v tomto prístupe rozdelené na malé oblasti, ktoré sa nazývajú bunky a to pomocou metódy nazývanej metóda bunkovej dekompozície (cell breakdown method). Princíp tejto metódy je založený na vytvorení bodu, ktorý bude reprezentovať pozíciu a orientáciu objektu v konfiguračnom priestore, kde každá súradnica tohoto bodu reprezentuje jeho polohu, respektíve orientáciu. Článok prezentuje algoritmy na výpočet prekážok v konfiguračnom priestore, kde všetky objekty sú polygóny alebo mnohosteny. Hlavným problémom, ktorým sa autor zaoberal bol ten, ako umiestniť objekt A do nejakej oblasti R tak, aby nekolidoval s ostatnými objektmi B_j, ktoré sú tam už umiestnené. Tento problém sa nazýva **Findspace problem**. Podobným problémom je určiť, ako naplánovať pohyb objektu A z jedného miesta na druhé bez toho, aby nastala kolízia s akýmkoľvek objektom B_j. Tento problém sa nazýva **Findpath problem**. Tieto dva problémy môžu byť definované formálne:

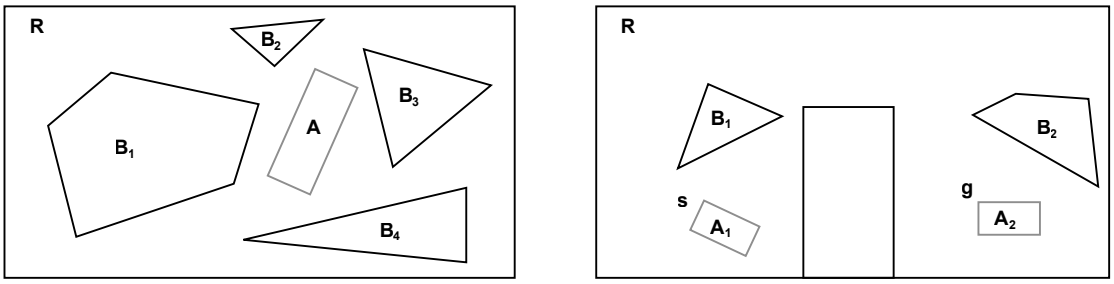
Definícia 1 *Nech R je objekt, ktorý obsahuje k_B iných, prípadne sa pretínajúcich objektov B_j.*

1) Findspace: Nájdenie pozície pre objekt A vo vnútri objektu R tak, aby pre všetky B_j platilo: $A \cap B_j = \emptyset$. Taká pozícia sa nazýva bezpečná pozícia (safe position).

2) Findpath: Nájdenie cesty pre objekt A z miesta s do miesta g takej, že objekt A je vždy vo vnútri objektu R a všetky pozície objektu A na ceste sú bezpečné. Taká cesta sa nazýva bezpečná cesta (safe path).

Obrázok 3.1 ilustruje definíciu Findspace a Findpath problémov.

V algoritmoch, ktoré sú v článku popísané, sa využívajú objekty nazývané *prekážky v konfiguračnom priestore (configuration space obstacles)*, ktoré predstavujú všetky pozície objektu A také, ktoré by mohli spôsobiť kolíziu s objektmi B_j. Vďaka tomuto prístupu je riešením týchto problémov len nájdenie bodu, resp. sekvencie bodov, ktoré sa nachádzajú mimo prekážok v konfiguračnom priestore. Výhoda je v tom, že riešiť pretínanie bodu s objektami je ľahšie, ako riešiť pretínanie jednotlivých objektov. Pri tvorbe prekážok v konfiguračnom priestore však treba rátať s tým, že objekty sa môžu rôzne pohybovať a otáčať.



Obr. 3.1: R, B_j a A pre Findspace a Findpath problémy v dvojdimenzionálnom priestore [27].

Konfiguráciu objektu A je možné vyjadriť ako bod $s \in \mathbb{R}^n$, kde $\mathbb{R}^n$ je n-dimenzionálny Euclidovský priestor. Táto konfigurácia sa nazýva $Cspace_A$. Nie všetky konfigurácie v $Cspace_A$ sú povolené, napríklad konfigurácia $A \cap B_j \neq \emptyset$ nie je povolená, pretože by spôsobila kolíziu.

Definícia 2 *$Cspace_A$ obstacle (prekážka) vzhľadom na B, označované ako $CO_A(B)$ je definované:*

$$CO_A(B) \equiv \{x \in Cspace_A \mid (A)_x \cap B \neq \emptyset\}$$

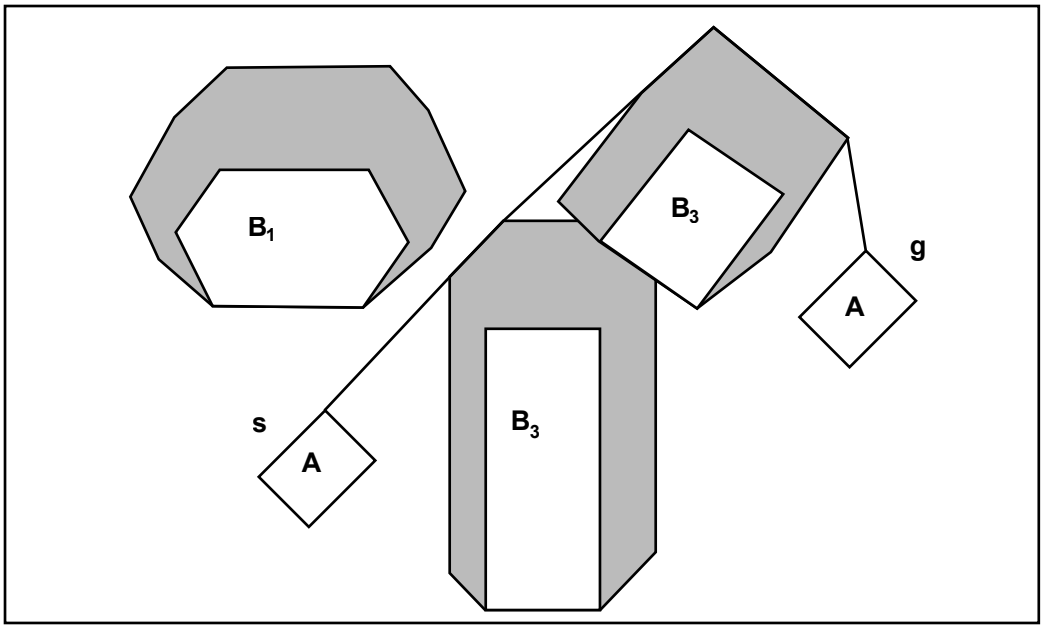
To znamená, že ak $x \in CO_A(B)$, potom sa $(A)_x$ pretína s B a x nie je "bezpečné" a naopak, ak pre akúkoľvek konfiguráciu x platí, že $x \notin CO_A(B_j)$ (pre všetky B_j), potom je x bezpečné.

Definícia 3 *$Cspace_A$ interior (vnútro) objektu B, označované ako $CI_A(B)$ je definované:*

$$CI_A(B) \equiv \{x \in Cspace_A \mid (A)_x \subseteq B\}$$

Je zjavné, že $CI_A(B) \subseteq CO_A(B)$. Použitím definícií $Cspace_A$ *obstacle* a $Cspace_A$ *interior* môžu byť problémy Findspace a Findpath vyjadrené ako ekvivalentné problémy, ktoré zahŕňajú umiestnenie jedného bodu, konfigurácie A, vzhľadom na objekty $CO_A(B_j)$ a $CI_A(R)$. Vo všeobecnosti je jedno, či hľadáme jeden bod konfigurácie A, alebo sekvenciu konfigurácií A mimo $CO_A(B_j)$ a zároveň vo vnútri $CI_A(R)$.

Ak je objekt A polygón a všetky objekty B_j tiež a ak je orientácia objektu A fixná, potom $CO_A^{xy}(B_j)$ sú tiež polygóny, kde xy označuje, že sa jedná o 2D priestor. V takom prípade je najkratšia bezpečná cesta pre objekt A lineárna cesta spájajúca počiatočný a cieľový bod cez vrcholy polygónov $CO_A^{xy}(B_j)$ ako na obrázku 3.2. Preto môže byť tento Findpath problém vyjadrený ako problém prehľadávania grafu. Graf je zostrojený spojením všetkých párov vrcholov $CO_A^{xy}(B_j)$, vrátane počiatočného a cieľového bodu, ktoré na seba "vidia", čiže môžu byť prepojené priamkou bez toho, aby táto priamka pretínala akékoľvek iné prekážky. Najkratšia cesta v grafe z počiatočného bodu do cieľového bodu je zároveň najkratšia cesta objektu A medzi týmito bodmi. Tento algoritmus rieši Findpath problém v 2D priestore, kedy je orientácia objektu A fixná, ale nájdená cesta môže byť chybná, ak sú nejaké nepresnosti v modeli, pretože tieto cesty sa dotýkajú prekážok v $Cspace_A$, takže ak by bol model úplne presný, objekt A by sa pri prechode z počiatočného bodu do cieľového bodu dotýkal prekážok B_j. Keďže sa táto práca venuje algoritmom typu coverage routing, tak sa tu nebudú riešiť konkrétne algoritmy pre point-to-point routing, je však možné si



Obr. 3.2: Findpath problém a jeho formulácia pomocou $CO_A^{xy}(B_j)$ [27].

ich pozrieť v článku [27], kde sa autor zaoberá jednak algoritmami pre riešenie Findspace a Findpath problémov pre objekty s fixnou orientáciou, tak aj voľnou orientáciou.

Prístup popísaný vyššie využívajú autori článku [30], kde predstavili prístup k riešeniu problému obmedzenia pohybu mobilných robotov na poliach. Princíp je založený na deterministickom vyhľadávaní v špeciálne diskretizovanom stavovom priestore. Vypočítava sa tam množina elementárnych pohybov, ktoré spájajú každú diskrétnu hodnotu stavu s množinou ich susedných stavov, ktoré je možné dosiahnuť pomocou realizovateľných pohybov. Táto množina pohybov teda vytvára súvislý vyhľadávací graf. Každý pohyb je navrhnutý tak, aby končil v diskrétnom stave, v ktorom spĺňa určité vlastnosti, ako napríklad poloha, rýchlosť, smer. Diskrétne stavy a pohyby sa opakujú a tým sa vytvára akási mriežka. Výsledná stavová mriežka umožňuje rýchle vyhodnocovanie akcií v konfiguračnom priestore a taktiež detekciu kolízií. Experimentálne výsledky demonštrujú, že tento spôsob plánovania umožňuje využiť celý rozsah manévrovateľnosti vozidla v nerovnom teréne a to aj v reálnom čase.

Pri vývoji algoritmov typu point-to-point sa postupom času využívali rôzne prístupy, ako napríklad Goto a kolegovia v ich článku [19], kde navrhli efektívny algoritmus plánovania cesty na výber optimálnej, alebo takmer optimálnej cesty v grafoch obrovských rozmerov. Základnou myšlienkou je vopred hľadať takmer optimálnu heuristiku. Ak táto heuristika je neprípustná, algoritmus sa zmení na klasický algoritmus A, ktorý vyberie takmer optimálnu cestu do cieľa. Pri praktickom použití algoritmus popísaný autormi extrémne šetrí čas, ale kvalita nájdenej cesty je nemenná.

Castillo a kolegovia [12] zase využili genetický algoritmus na riešenie problému plánovania cesty. Pomocou simulácie ukázali, že konvenčné genetické algoritmy aj viacúčelové

genetické algoritmy vybavené základným opravným mechanizmom pre neplatné cesty môžu vyriešiť tento problém.

3.1 Rapidly exploring random tree

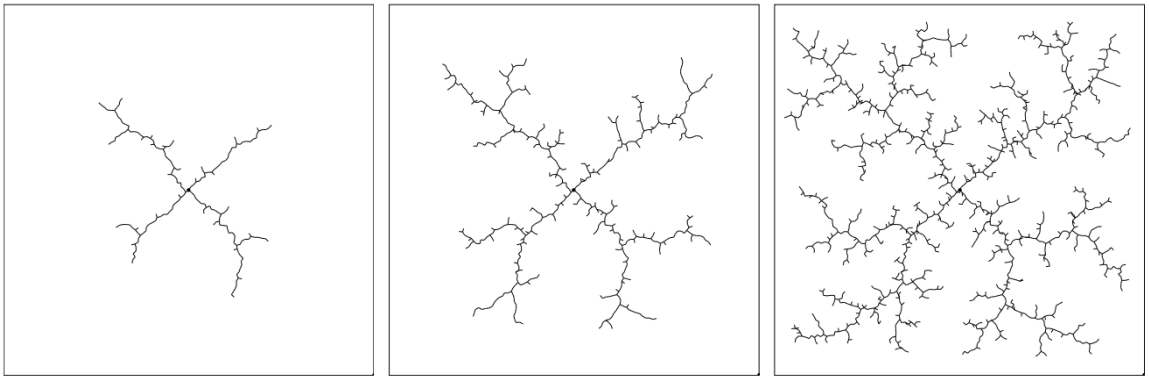
Ďalším prístupom môže byť využitie RRT (rapidly exploring random tree). Jeho autorom je Steven M. LaValle [26]. RRT je algoritmus navrhnutý na efektívne prehľadávanie nekonvexných, viacdimenzionálnych priestorov vytváraním stromu, ktorý zapĺňa voľný priestor. Strom je postupne vytváraný zo vzoriek, ktoré sú náhodne vyberané a je tvorený tak, aby rástol smerom k nepreskúmaným oblastiam a veľmi dobre sa vie vysporiadať s prekážkami. RRT vytvára strom, ktorý začína v počiatočnom bode, na ktorý nadväzujú náhodné vzorky umiestnené v prehľadávanom priestore. Každá novo vytvorená vzorka sa spojí so vzorkou stromu, ktorá je k nej najbližšie. Ak toto spojenie spĺňa všetky podmienky, teda dodržiava všetky obmedzenia a neprechádza žiadnou prekážkou, je pridané ako nový stav do vytváraného stromu. Dĺžka spojenia medzi vzorkou a stromom je často obmedzená na určitú maximálnu veľkosť, aby neboli vytvárané spojenia so vzorkami, ktoré sú ďaleko od stromu. Ak je nová vzorka vytvorená od najbližšej časti stromu ďalej ako je maximálna povolená hodnota, nový stav stromu je vytvorený na priamke spájajúcej túto vzorku s najbližším stavom stromu a to v maximálnej povolenej vzdialenosti. Výber vzoriek nemusí byť náhodný, ale môže byť ovplyvnený tak, že sa budú vzorky vyberať z určitej oblasti s väčšou pravdepodobnosťou. Väčšina praktických implementácií RRT to využíva na usmernenie hľadania smerom k cieľovému bodu prehľadávania.
Vstup algoritmu 1: Počiatočná konfigurácia q_{init}, počet vrcholov v RRT K, maximálna vzdialenosť Δ d.
Výstup algoritmu 1: RRT graf G

Algoritmus 1 RRT algoritmus

$G.init(q_{init})$
for $k = 1$ **to** K **do**
  $x_{rand} \leftarrow RAND\_STATE()$
  $x_{near} \leftarrow NEAREST\_NEIGHBOUR(q_{rand}, G)$
  $u \leftarrow SELECT\_INPUT(x_{rand}, x_{near})$
  $x_{new} \leftarrow NEW\_STATE(x_{near}, u, \Delta d)$
  $G.add\_vertex(q_{new})$
  $G.add\_edge(q_{near}, q_{new}, u)$
end for
return G

RRT graf sa najskôr inicializuje počiatočným vrcholom q_{init}. Následne sa v každom cykle vyberie náhodný vrchol pomocou funkcie $RAND\_STATE()$. Pre tento nový vrchol sa pomocou funkcie $NEAREST\_NEIGHBOUR(x_{rand}, G)$ nájde najbližší vrchol grafu. V ďalšom kroku sa do u uloží nový vstup, ktorý minimalizuje vzdialenosť z x_{near} do x_{rand}. Funkcia $NEW\_STATE(x_{near}, u, \Delta d)$ sa volá pre každý vstup u, aby sa vyhodnotil potenciálny nový stav. Následne sú nový stav x_{new} a vrchol z x_{new} do x_{near} pridaný do grafu G. Vytváranie RRT grafu je možné vidieť na obrázku 3.3.

Aj keď RRT algoritmus dokáže pomerne rýchlo nájsť riešenie problému hľadania cesty k cieľu, má tendenciu konvergovať k riešeniu ktoré nie je ani zďaleka optimálne. Preto Sertac Karaman a Emilio Frazzoli [24] vymysleli lepšiu verziu tohto algoritmu a to konkrétne



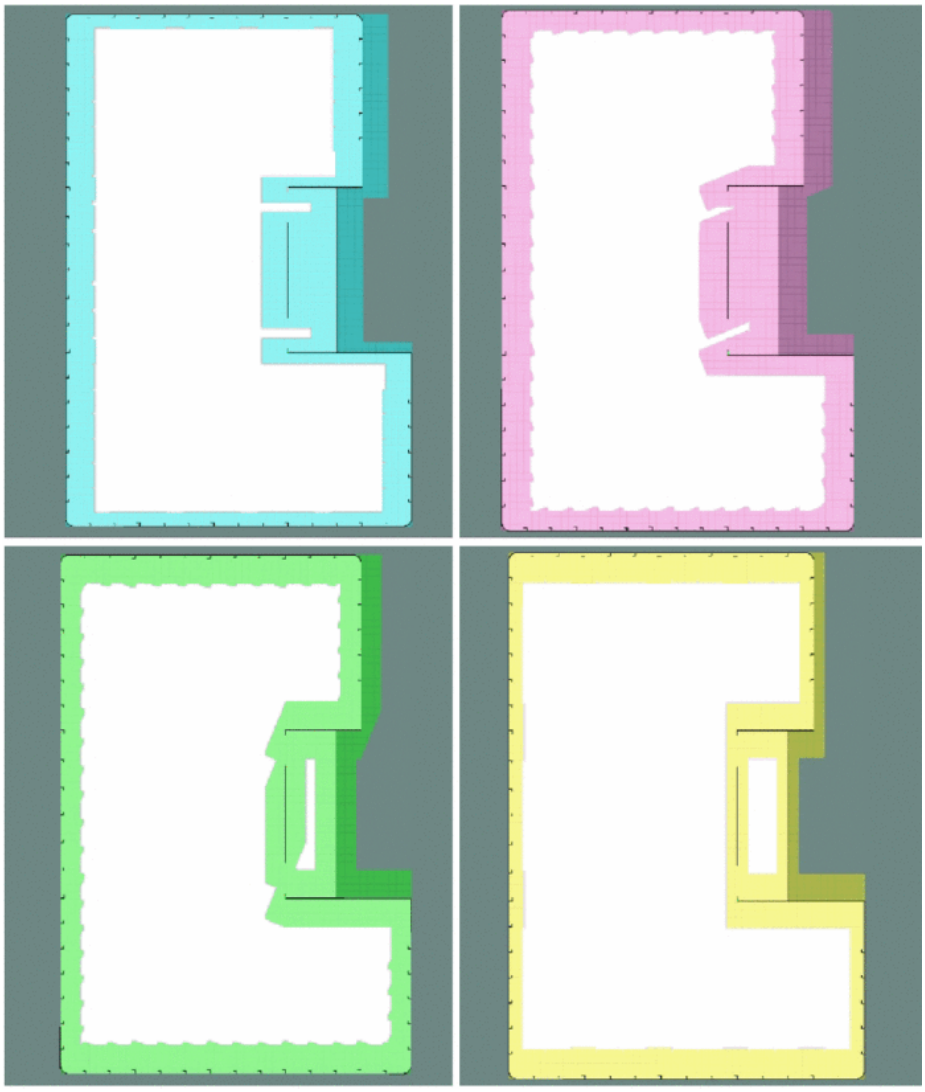
Obr. 3.3: Princíp činnosti algoritmu RRT [6].

RRT*. Ak sa počet vrcholov grafu blíži k nekonečnu, RRT* dokáže nájsť najkratšiu možnú cestu do cieľa a tým pádom nájsť takmer optimálne riešenie. Základný princíp je rovnaký ako pri RRT, ale sú k nemu pridané dva výrazné vylepšenia. Prvým je to, že každý vrchol grafu si zaznamenáva vzdialenosť od počiatočného bodu. To znamená, že rovnako ako pri RRT sa vytvorí nový bod, nájde sa najbližší vrchol a pripojí sa k nemu. Rozdiel je však v tom, že novovytvorený bod preskúma svoje okolie a ak v ňom nájde vrchol, ktorý má menšiu vzdialenosť od počiatočného bodu (cena cesty) ako vrchol, ku ktorému sa pripojil, tak toto spojenie zruší a pripojí sa k vrcholu s menšou vzdialenosťou k počiatočnému bodu. Druhým vylepšením je to, že po pripojení uzlu k najlepšiemu susedovi sa opäť preskúma okolie tohto vrcholu a kontroluje sa, či jeho susedné vrcholy nebudú mať po prepojení s ním lepšiu cenu cesty, ako majú teraz. Ak áno, zrušia svoje pôvodné spojenie a pripoja sa na tento novovytvorený vrchol. Týmto sa dosiahne to, že výsledná cesta je kratšia a oveľa hladšia. Autori algoritmu porovnali klasický RRT a ich RRT* pomocou simulácií založených na metóde Monte Carlo, ale aj pomocou experimentu v reálnom prostredí pri plánovaní trasy vysokozdvižného vozíka.

Článok [17] predstavil rozšírenie algoritmu A*, kde sa okrem 2D polohy vozidla brala do úvahy aj jeho orientácia, čoho výsledkom je vytvorenie hladšej cesty. Aby sa predišlo prílišnej konzervatívnosti pri tvorení cesty, alebo aby sa nevytvárali nerealizovateľné cesty, zohľadňovala sa skutočná stopa vozidla. Aby sa zlepšila kontrola kolízií a zredukoval sa stavový priestor, bolo vytvorených 16 vrstiev orientácií na základe orientácie vozidla, ktoré v smere vozidla ako keby "nafúkli" prekážky a týmto sa zjednodušila kontrola kolízií, ako je to možné vidieť na obrázku 3.4. Pri tvorbe cesty sa však skúmajú len susedné vrstvy, čo uľahčuje výpočet. Výsledkom je algoritmus schopný generovania hladkej cesty pre vozidlo akéhokoľvek tvaru, berúc do úvahy obmedzenia orientácie s cieľom vytvoriť cestu v blízkosti prekážok.

3.2 Algoritmy inšpirované prírodou

V téme plánovania cesty metódou point-to-point sa začalo venovať viac pozornosti algoritmom inšpirovanými prírodou. V literatúre sa často objavujú genetické algoritmy (genetic algorithm - GA), optimalizácia hejnom častíc (particle swarm optimization - PSO), alebo optimalizácia kolóniou mravcov (ant colony optimalization - ACO), ktoré ukazujú sľubné výsledky pri plánovaní trasy. Niektoré z týchto algoritmov sú popísané v článku A Survey of Path Planning Algorithms for Mobile Robots [15], ktorý poskytuje základný prehľad



Obr. 3.4: Nafúknutie prekážok pre rôzne orientácie [17].

algoritmov pre plánovanie trasy, zaoberá sa napríklad algoritmami A*, D*, Dijkstrovým algoritmom, už spomínaným RRT, alebo genetickými a ACO algoritmami.

Algoritmy plánovania trasy založené na mriežke popísané vyššie vyžadujú značný výkon procesora a veľké množstvo pamäte, preto bude ďalej predstavený genetický algoritmus, ktorý pomáha prekonať tieto obmedzenia. Jeho výhodou je to, že môže byť použitý na pokrytie veľkého prehľadávacieho priestoru a využíva pri tom minimálne zdroje procesora a pamäte. Menšou nevýhodou môže byť, že nájdené riešenie nemusí byť vždy optimálne, teda nie je nájdená úplne najkratšia cesta. Zaujímavou aplikáciou genetického algoritmu je použitie v malých humanoidných robotoch, ktorí sa snažia hrať futbal. Cieľom robota je dostať loptu do súperovej brány, zatiaľ čo opačný tým predstavuje prekážky. V tomto prípade je to teda pre robota dynamické plánovanie trasy. Keďže ide o dynamické plánovanie, optimálne riešenie, teda nájdenie najkratšej cesty do cieľa bez kolízie, je potrebné aktualizovať podľa zmeny okolia. V tomto prípade evolučné algoritmy konvergujú k optimálnemu riešeniu. Všetky možné riešenia sú reprezentované ako jednotlivci populácie, ktorí

sú tvorení súborom génov a každý z týchto génov predstavuje istý parameter. Nová generácia sa vytvára výberom najlepších jedincov z rodičovskej generácie, na ktorých sú použité rôzne genetické operácie, ako napríklad kríženie alebo mutácia. Každý potomok je následne testovaný pomocou takzvanej fitness funkcie, ktorá je jedinečná pre každý riešený problém. Táto funkcia musí brať do úvahy nielen najkratšiu trasu, ale aj jej hladkosť a voľný priestor pre dynamické plánovanie.

$$eval(p) = w_d dist(p) + w_s smooth(p) + w_c clear(p) \tag{3.1}$$

kde w_d, w_s a w_c reprezentujú váhy jednotlivých zložiek a $dist(p)$, $smooth(p)$ a $clear(p)$ sú definované následovne:

$$dist(p) = \sum_{i=1}^{n-1} d(s_i)$$

kde $d(s_i)$ je vzdialenosť medzi dvoma susednými uzlami,

$$smooth(p) = \sum_{i=2}^{n-1} e^{a(\theta_i - \alpha)}$$

kde θ_i je uhol medzi predĺžením dvoch úsečiek spájajúcich i-tého bodu, α je požadovaný uhol a a je koeficient,

$$clear(p) = \sum_{i=1}^{n-2} e^{a}(g_i - \gamma)$$

kde g_i je najmenšia vzdialenosť z i-tého segmentu ku všetkým prekážkam a γ je požadovaná voľná vzdialenosť (odstup).

Genetické operátory sa používajú na vývoj ciest pre konkrétnych vybraných rodičov. Operátor výberu vyberie najvhodnejších jedincov a nechá ich odovzdať svoje gény ďalšej generácii. Operátor kríženia náhodne vyberie bod kríženia z génov oboch rodičov, ktorí sa majú spáriť. Mutačný operátor následne vykoná preklopenie niektorých bitov v génovom reťazci, aby sa zachovala diverzita. V článku je spomenutá aj implementácia pre plánovanie trasy v nedynamickom prostredí, konkrétne v statickom poli, kde bola táto metóda účinná. Dôležitou charakteristikou, vďaka ktorej sú genetické algoritmy vhodné pre riešenie takýchto úloh je schopnosť paralelného vyhľadávania a schopnosť hľadať optimálnu cestu v danom prostredí.

Algoritmus optimalizácie kolónie mravcov (ACO), ktorý je založený na heuristickom prístupe inšpirovanom kolektívnym správaním mravcov, ktorí zanechávajú stopy pri hľadaní najkratšej cesty bez kolízií je ďalším z príkladov, kedy sa vedci inšpirovali prírodou pri navrhovaní algoritmov optimalizácie plánovania ciest. Tento algoritmus navrhol Marco Dorigo vo svojej práci [16], aby simuloval hľadanie potravy mravcov. Keď mravce hľadajú potravu, tak po ceste uvoľňujú chemické látky - feromóny. Nasledujúce mravce si potom vyberú vhodnú cestu na základe koncentrácie feromónov, ktoré zanechali predchádzajúce mravce.

Keď je koncentrácia feromónov vyššia, zvyšuje sa aj pravdepodobnosť, že si mravce danú cestu vyberú. Koncentrácia feromónov sa však časom vďaka odparovaniu znižuje. Keď sa počet hľadaní ciest zvyšuje, kratšie cesty budú mať vyššiu koncentráciu feromónov, pretože viac mravcov tieto cesty navštívi a vďaka odparovaniu feromónov budú iné cesty vyberané s menšou pravdepodobnosťou, čím sa nakoniec docieli to, že sa nájde kratšia cesta do cieľa. Algoritmus je založený na feromónových depozitoch, ktoré určujú pravdepodobnosť prechodu umelého mravca z jedného uzla do druhého na ceste. Pravdepodobnosť prechodu z uzla i do j je daná vzťahom:

$$p_{ij}^k = \begin{cases} \frac{\left(\tau_{ij}^k\right)^\alpha \left(\eta_{ij}^k\right)^\beta}{\sum_{l \in N_i^k} \left(\tau_{il}^k\right)^\alpha \left(\eta_{il}^k\right)^\beta} & \text{ak } j \in N_i^k \\ 0 & \text{ak } j \notin N_i^k \end{cases} \tag{3.2}$$

kde τ_{ij}^k je hladina feromónov označujúca počet mravcov, ktorí si vybrali rovnakú hranu ako predtým, η_{ij}^k je heuristika označujúca blízkosť pozície cieľu a α a β sú váhy určujúce dôležitosť feromónov a heuristiky. Vzhľadom na aplikáciu, hodnoty α, β a η sú vyberané tak, aby bolo možné určiť susedné uzly, ktoré sa vyberú na ceste k cieľu. Tradičný ACO algoritmus má pomalú konvergenciu kvôli obmedzeniam heuristiky. Pri jeho využití v aplikáciách plánovania cesty je ľahké v počiatočnej fáze optimalizácie nájsť len lokálne optimálne riešenie, čo vedie k stagnácii. Preto bolo vymyslených viacero vylepšení, vďaka ktorým je konvergencia k optimálnej ceste oveľa rýchlejšia, napríklad The Enhanced Ant Colony Algorithm, The Improved Ant Colony Optimalization, alebo The Adaptive Ant Colony algorithm, ktorý má dobrú konvergenciu lepšiu optimalizačnú výkonnosť v porovnaní s iným variantami algoritmu ACO.

Kapitola 4

Coverage Routing

Vytvorenie trasy, ktorá vedie všetkými bodmi určitej oblasti (pokrýva celú plochu) a pritom sa vyhýba prekážkam sa nazýva coverage path planning (CPP). Táto úloha je neoddeliteľnou súčasťou mnohých aplikácií v robotike, ako napríklad automatizované kombajny (všeobecne zberače akejkoľvek úrody), automatické vysávače, roboty na maľovanie, alebo umývanie okien, ale aj odmínovacie roboty. V jednej z prvých prác [11] na tému CPP boli definované požiadavky, ktoré musí robot splniť, aby mohol vykonať požadovanú operáciu. Hoci cieľovou aplikáciou v uvedenom článku je mobilný robot pohybujúci sa v plochom dvojrozmernom prostredí, rovnaké kritéria platia aj pre iné scenáre pri plánovaní trasy pokrytím. Požiadavky sú nasledovné:

1. Robot musí byť schopný pokryť celý priestor
2. Robot musí úplne pokryť plochu bez akéhokoľvek prekrývania
3. Procesy musia byť nepretržité a sekvenčné, pričom sa nesmú opakovať žiadne cesty
4. Robot sa musí vyhýbať všetkým prekážkam
5. Využívajú sa len základné trajektórie pohybu
6. Za daných okolností sa hľadá optimálne riešenie

V niektorých prípadoch však nie je možné splniť všetky z týchto požiadaviek. Napríklad pri orbe, aplikovaní hnojív alebo postrekov proti škodcom je nežiadúce, aby sa cesty stroja prekrývali, ale pri zbere plodín, ak bude stroj prechádzať cez už spracovanú časť poľa, to zas až tak vadiť nebude. Výsledkom toho je priradiť jednotlivým požiadavkom určitú prioritu a to na základe vykonávanej činnosti.

Problém CPP súvisí s takzvaným "covering salesman problem", ktorý je variantou dobre známeho "traveling salesman problem" s tým rozdielom, že sa nenavštevuje každé mesto, ale agent musí navštíviť všetky štvrte v danom meste. Rozdiel CPP je v tom, že tu sa musia prejsť všetky body v danej oblasti. Keďže problém traveling salesman je NP-ťažký, čas potrebný na vyriešenie tohto problému drasticky rastie s rastúcim problémom. Problém známy ako "lawnmower problem", kde je cieľom nájsť cestu pre kosačku tak, aby pokosila všetku trávu na nejakej ploche je tiež NP-ťažký. V tomto probléme sa dokonca neberú do úvahy prekážky. Dva ďalšie podobné problémy ako CPP sú "art gallery problem" a "watchman route problem". Art gallery problem hľadá najmenší počet strážnikov umiestených v galérii tak, aby každé miesto bolo videné aspoň jedným strážnikom. Watchman route

problem hovorí o tom, ako nájsť cestu z nejakého bodu späť do tohto bodu tak, aby sme po ceste videli všetky ostatné body v priestore. Oba tieto problémy sú vo všeobecnosti taktiež NP-ťažké.

Algoritmy na pokrytie priestoru možno klasifikovať ako heuristické, alebo úplné v závislosti od toho, či preukázateľne zaručujú pokrytie daného priestoru. Môžu byť taktiež klasifikované ako off-line, kde sa predpokladá, že prostredie je vopred známe a spoliehame sa na to, alebo on-line algoritmy, kde sa nepredpokladá znalosť prostredia, ktorá sa má pokryť a využívajú sa tak rôzne senzory, ktoré v reálnom čase určujú polohu vozidla a jeho okolie.

Jednou z možností, ako riešiť problém CPP je pomocou náhodného výberu. Je to prístup, na ktorom sú založené napríklad robotické vysávače, alebo obecne čističe podláh. Tento prístup má určité výhody, hlavnou je, že robot nepotrebuje žiadne zložité senzory na lokalizáciu ani nemusí byť vykonaný zložitý výpočet cesty. Tento prístup sa spolieha na to, že ak bude robot dostatočne dlho náhodne jazdiť po danej ploche, tak ju celú pokryje. Tento prístup sa však nedá využiť pri strojoch vykonávajúcich prácu na poliach a to z očividných dôvodov. Taktiež sa tento prístup nedá reálne využiť na činnosti vykonávané v 3D priestore, keďže náklady na prevádzku vozidla, či už časové alebo ekonomické, by boli veľmi vysoké. Tento prístup je vhodné použiť na malých plochách, kde nám nevadí, ak bude robot jazdiť viackrát po rovnakej ploche [18].

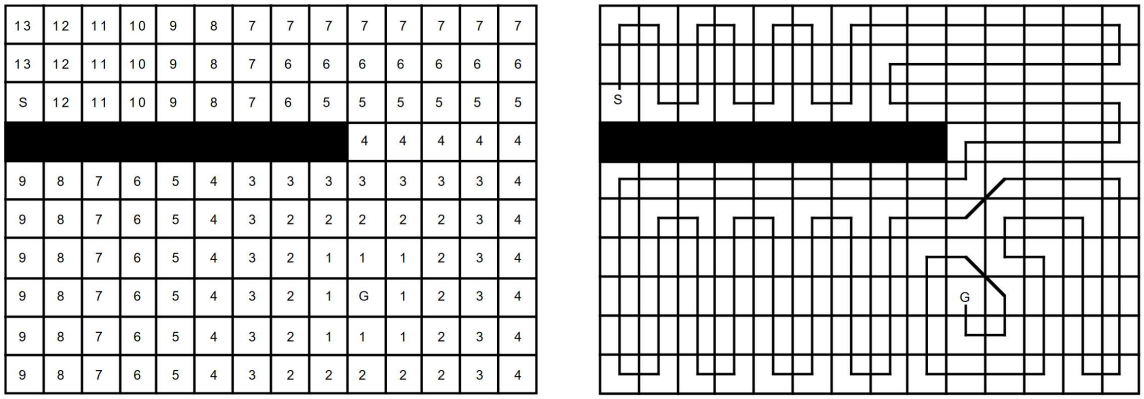
Ďalšou z možností, ako riešiť problém CPP je pomocou bunkovej dekompozície, kde sa mapa rozdelí na mriežku buniek a trasa sa navrhne tak, aby prechádzala každou bunkou. V ďalšej časti sa budeme zaoberať tromi typmi dekompozície: približnou (approximate), polo-približnou (semi-approximate) a presnou (exact).

4.1 Približná bunková dekompozícia

Tento typ bunkovej dekompozície je založený na jemnej mriežke, ktorá reprezentuje plochu, ktorú je treba pokryť. Všetky bunky sú rovnakej veľkosti a tvaru, ale ich zjednotenie predstavuje len približný tvar danej plochy. V tomto prístupe sa predpokladá, že akonáhle robot vojde do určitej bunky, v tom momente ju pokryje. Bunky sú typicky také veľké, ako je plocha robota. Keď robot navštívi všetky bunky, tak je pokrytie dokončené. Na nájdenie trasy je tu použitý takzvaný "wavefront" algoritmus, ktorý priradí cieľovej bunke hodnotu 0 a následne priradí hodnotu 1 všetkým bunkám v okolí cieľovej bunky. Všetkým bunkám, ktoré sú v okolí buniek s hodnotu 1 priradí hodnotu 2 a to sa opakuje, až kým sa nedostaneme k zdrojovej bunke. Následne je použitý gradientný zostup na nájdenie cesty. Tento algoritmus však nájde len najlepšiu cestu zo zdrojovej bunky do cieľovej, ale nepokryje celý priestor. Preto bol tento algoritmus vylepšený tak, že šírenie hodnôt sa neskončí, kým nemá každá bunka priradená svoju hodnotu. Následne sa vykoná takzvaný "pseudogradientný vzostup", čo znamená, že sa robot posunie k bunke, ktorá má najväčšiu hodnotu susediacu s aktuálnou bunkou a ešte nebola navštívená. Jedinečnou vlastnosťou tohto algoritmu je to, že sa nenájde len cesta, ktorá pokryje celý priestor, ale môže sa špecifikovať aj jej začiatok a cieľ. Taktiež sa môžu použiť ďalšie metódy, aby sa docielila trasa s menším počtom zákrut, napríklad tak, že sa berie do úvahy vzdialenosť každej bunky k najbližšej prekážke.

V článku sa spomína aj Algoritmus nazývaný Spanning Tree Covering od autorov Gabriely a Rimon, ktorý dokáže nájsť optimálnu cestu v mriežkovom zobrazení priestoru. Oblasť sa rozdelí na disjinktné bunky a následne sa prechádza graf (spanning tree graph) tak, že každá bunka sa navštívi len raz. Sú vytvorené tri verzie tohto algoritmu. Prvou je off-line

verzia, kde má robot presné informácie o priestore. Vďaka tomu je možné vypočítať optimálnu cestu v lineárnom čase $O(N)$, kde N je počet buniek. Druhá verzia algoritmu je on-line, kde robot využíva vlastné senzory na detekciu prekážok a vytvára graf zatiaľ čo pokrýva danú plochu. Algoritmus nájde optimálnu trasu v lineárnom čase $O(N)$, ale vyžaduje rovnako $O(N)$ pamäte na výpočet. Posledná verzia algoritmu je založená na niečom ako ACO. Tu robot tiež nemá znalosti o danom priestore, ale môže nechávať feromónové značky počas pokrývania priestoru. Tento algoritmus má časuvú zložitosť lineárnu $O(N)$, ale pamäťovú zložitosť konštantnú $O(1)$.

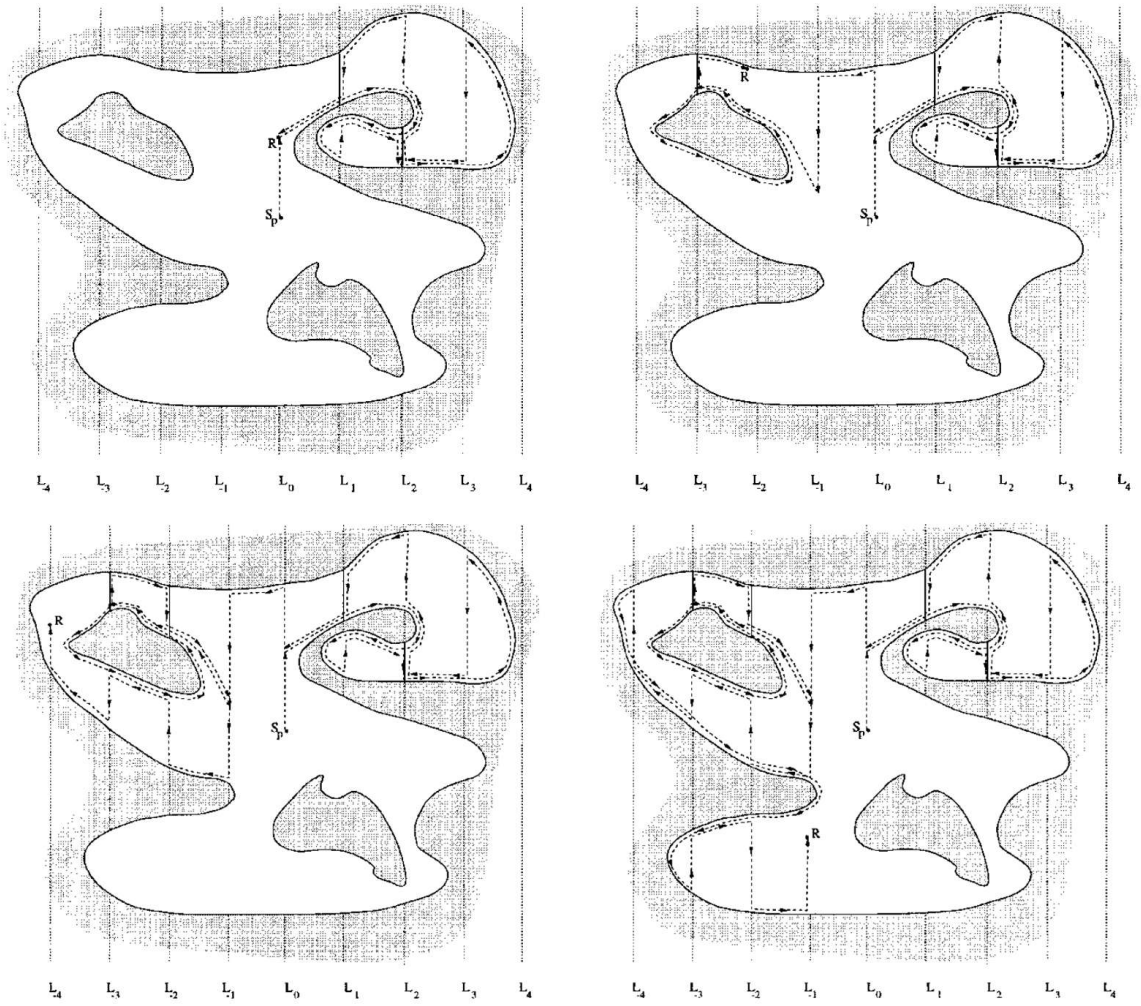


Obr. 4.1: Približná bunková dekompozícia [34].

4.2 Polo-približná bunková dekompozícia

Autori [22][28] prezentujú algoritmus pokrytia, ktorý je založený na čiastočnej diskretizácií priestoru, kde bunku majú pevnú šírku, ale na výšku nie sú nejako obmedzené. Ich algoritmus prekrytia terénu sa dá použiť na jednoduché, ale aj zložitejšie prepojené prostredia. Jednoduchosť algoritmu spočíva v rekurzívnosti. Robot môže začať v akomkoľvek mieste a bude sa kľukatiť pozdĺž rovnobežných čiar mriežky tak, aby pokryl danú oblasť. Oblasti, ktoré nie sú pokryté, alebo by boli pokryté viackrát sú detekované a následne pokryté rovnakým spôsobom. Rekurzívny postup zaručí, že sa tieto oblasti prekryjú v poradí "depth-first". Robot si však musí pamätať všetky vstupy a výstupy z daných oblastí (inlets), pretože by mohlo prekryť túto plochu viackrát. Preto musia byť vytvorené špeciálne vstupy (diversion inlets), do ktorých robot vstupuje pozdĺž ich hranice. Keď danú oblasť pokryje, tak z nej vyjde a ďalej pokračuje, ako keby tam ani nebola. Keď plocha, ktorá sa má pokryť obsahuje ostrovčeky aj tieto špeciálne vstupy, postupuje sa veľmi podobne, len s drobnými úpravami tak, aby boli tieto ostrovčeky pokryté zo všetkých strán. Tento postup je lepšie pochopiteľný z obrázku, viz. obrázok 4.2.

Je dokázané, že algoritmus pracuje správne a jeho zložitosť sa meria dvoma rôznymi spôsobmi: z hľadiska vzdialenosti, ktorú robot prejde a z hľadiska pamäte, ktorá je potrebná na uloženie vstupných informácií. V rovinnom prostredí je dĺžka trasy v najhoršom prípade lineárna vzhľadom na dĺžku vonkajších hraníc, hraníc ostrovčekov a dĺžku segmentov mriežky, ktorá rozdeľuje danú plochu. Ďalšou výhodou tohto algoritmu je to, že je možné ho použiť on-line, teda nie je potrebné mať vopred informácie o ploche. V nerovinnom dvojrozmernom prostredí sa horná hranica dĺžky cesty nelíši od dĺžky cesty v rovinnom prostredí o viac ako multiplikatívnu konštantu, ktorá závisí na sklone poľa.



Obr. 4.2: Polo-približná bunková dekompozícia [14].

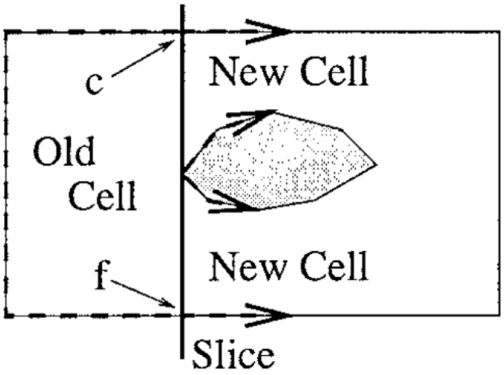
4.3 Presná bunková dekompozícia

Presná bunková dekompozícia je súbor nepretínajúcich sa oblastí, ktoré sa nazývajú bunky a ich zjednotenie vypĺňa celú cieľovú plochu. Robot môže pokryť tieto bunky len použitím jednoduchých pohybov tam a späť, takže plánovanie cesty na pokrytie celej plochy je obmedzené len na plánovanie prechodov z jednej bunky do druhej. Jenou z populárnych techník, ktorá môže nájsť plné pokrytie plochy sa nazýva lichobežníkový rozklad (trapezoidal decomposition), pri ktorej je plocha rozdelená do buniek lichobežníkového tvaru. Plné pokrytie plochy je docielené navštívením každého uzlu grafu vytvoreného z jednotlivých buniek.

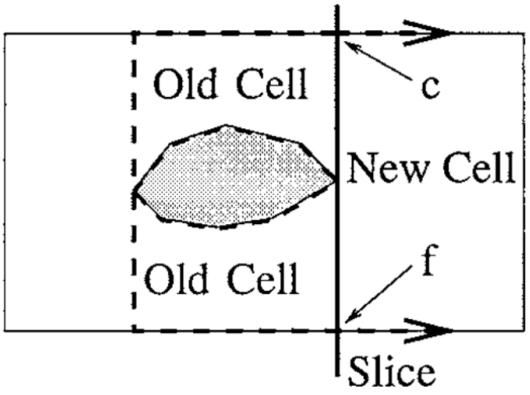
"Boustrophedon" dekompozícia

Množstvo buniek v lichobežníkovom rozklade môže byť zlúčených tak, že robot môže pokračovať v pohybe tam a späť tak, že pokryje celú bunku a nepretína prekážky. Preto bol vyvinutý nový rozklad nazývaný "boustrophedon decomposition" na vyriešenie problému

so zhlukovanými bunkami. Pri tejto metóde sa prechádza prostredím pomocou úsečky. Keď dôjde k prerušeniu úsečky, vytvorí sa nová bunka. To nastáva vtedy, ak sa úsečka dotkne prekážky. Keď to nastane, vytvoria sa dve nové bunky, ako je to znázornené na obrázku 4.3. Naopak, ak sa úsečky opäť spoja do pôvodnej úsečky, dve bunky sa zlúčia do jednej, ako je to znázornené na obrázku 4.4. Ak je dokončená dekompozícia a je vytvorený graf prepojenia buniek, je použitý algoritmus na prehľadávanie grafov a tak sa určí cesta cez každú bunku a keďže v každej bunke sa vykonávajú len pohyby tam a späť, dôjde k pokrytiu celej plochy.



Obr. 4.3: Presná bunková dekompozícia - vytvorenie nových buniek [14].

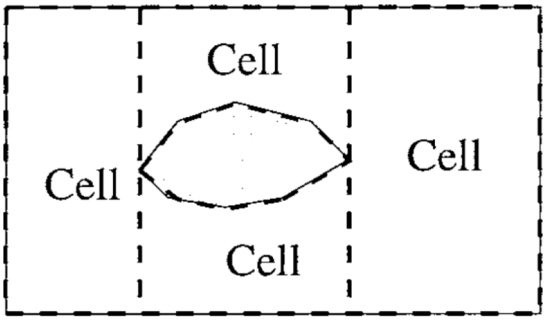


Obr. 4.4: Presná bunková dekompozícia - spojenie buniek [14].

Optimálna dekompozícia

Huang [23] prispôsobil prístup "boustrophedon" na dosiahnutie optimálneho pokrytia. Namiesto minimalizovania dĺžky cesty sa pri optimalizácii zameral na celkový počet odbočiek potrebných na pokrytie všetkých buniek, pretože zákruty sú časovo náročné, robot musí spomaliť, vykonať otočku a znova zrýchliť a náklady na presun medzi bunkami sú zanedbateľné. Počet odbočiek je viac-menej úmerný šírke oblasti, ktorá vznikla rozdelením plochy pomocou prístupu "boustrophedon". Problém potom nastáva vybrať si smer, ktorým sa pohybuje úsečka v tomto prístupe. Huang ukázal, že optimálna dekompozícia musí byť vykonaná pomocou čiar, ktoré sú paralelné s okrajmi hraníc alebo prekážok [14].

Veľmi dôležitou aplikáciou algoritmov na plánovanie trasy je pri odmínovaní alebo odstraňovaní nevybuchnutej munície. Tejto téme sa venovali autori článku [10]. Vo všeobec-



Obr. 4.5: Výsledok presnej bunkovej dekompozície (Boustrophedon decomposition) [14].

nosti sú informácie o mínových poliach minimálne a taktiež sú to neštrukturované prostredia, kde sa môžu často vyskytnúť chybné dáta zo senzorov. Využitie robotov pri tejto činnosti minimalizuje riziko a zvyšuje efektivitu týchto úloh. Prvým cieľom pri odmínovaní je lokalizácia všetkých cieľov. To si vyžaduje plánovač, ktorý generuje cestu, po ktorej detektor prejde všetkými bodmi a pokryje tak celú plochu, teda plánovač je kompletný. Predtým, ako autori zverejnili túto prácu, bol pohyb robota pri týchto úlohách náhodný, alebo boli využité len jednoduché heuristiky. Tieto metódy však nemajú záruku úplného pokrytia priestoru, čo je nevyhnutné pre nájdenie všetkých mín. Autori článku najskôr zhrnú ich algoritmus úplného pokrytia založený na kompletnej bunkovej dekompozícií a porovnajú ho s randomizovanými prístupmi. Robot vytvára túto dekompozíciu zatiaľ čo pokrýva priestor pohybmi tam a späť. Demonštrujú, že kompletné pokrytie je rýchlejšie ako pokrytie náhodným prehľadávaním. Taktiež je ukázané, že pri prístupe kompletného pokrytia je možné použiť filter, ktorý dokáže detekovať nesprávne údaje prijaté senzormi, čo je nevyhnutné pre úspešné nasadenie robota pri týchto činnostiach. Autori overili výsledky vykonaním experimentov v neštrukturovanom vnútornom prostredí. Pre scenáre, kde sú k dispozícií nejaké informácie o mínovom poli, zaviedli pravdepodobnostnú metódu, aby proces odmínovania urýchlili a robot nemusel vykonávať kompletné pokrytie.

Yang a Luo [33] navrhli metódu postavenú na neurónovej sieti na hľadanie cesty v dynamickom prostredí v reálnom čase pre čistiace roboty. V neurónovej sieti sa nachádzajú iba lokálne prepojenia medzi neurónmi. V prípade, že je pracovné prostredie obdĺžnikového tvaru, je počet neurónov $M = N_x \times N_y$, kde N_x a N_y sú diskretizované rozmery daného priestoru. Každý neurón má najviac 8 prepojení, takže celkový počet prepojení nie je viac ako $8M$. Ak je prostredie veľkosti $N \times N$, potom existuje N^2 neurónov a teda najviac $8N^2$ prepojení, čiže výpočtová zložitosť algoritmu je $O(N^2)$. Algoritmus je výpočtovo nenáročný, cesta robota je generovaná prostredníctvom dynamickej neurónovej aktivity a predchádzajúceho umiestnenia robota bez akejkoľvek optimalizácie globálnej "cost" funkcie (vykonáva sa len lokálne vyhľadávanie nasledujúcej polohy robota medzi jeho ôsmimi susednými neurónmi) a to bez akejkoľvek predchádzajúcej znalosti dynamického prostredia. Vyžaduje sa len znalosť súčasného stavu prostredia, ktorá sa zisťuje pomocou senzorov. Model neuviazne v deadlocku, ak sa robot ocitne na mieste, ktorého susedné miesta sú buď preskúmané, alebo sú to prekážky, pretože miesta, ktoré ešte neboli pokryté dokážu priťahovať robota z ktoréhokoľvek miesta v prostredí pomocou propagácie neurónovej aktivity. Model sa taktiež vie vyrovnať so zmenami okolia, ako napríklad pridávanie alebo odstraňovanie prekážok.

Ďalší z algoritmov bol taktiež vyvinutý pre pohyb čistiacich robotov. Štúdia [32] popisuje nový prístup na riešenie problému plánovania trasy pokrytím priestoru. V tomto prístupe

je plocha opäť rozdelená do malých štvorcov, ktorých veľkosť uhlopriečky sa rovná veľkosti robota. Sú definované štyri základné druhy pohybu: pohyb rovno, odbočenie doprava, odbočenie doľava a otočka (U-turn). Vyhodnocovacou funkciou je celková cena cesty, ktorá je výsledkom súčtu cien všetkých pohybov. Táto funkcia dokáže nájsť lepšiu cestu, aj keď sú na výber dve cesty s rovnakým počtom opakovaných návštev niektorých navštevovaných štvorcov, berie teda do úvahy spotrebu energie. Na to bol využitý genetický algoritmus. Vďaka kríženiu, výberu a mutácii genetického algoritmu sa trasa vylepšuje, kým sa z nej nestane optimálne, alebo takmer optimálne riešenie.

V tejto kapitole boli zhrnuté niektoré algoritmy z oblasti plánovania trasy pokrytím priestoru. Tieto algoritmy boli zoradené od tých najstarších po tie najnovšie a boli popísané ich základné princípy. Aj keď sú všetky tieto riešenia zaujímavé, nie všetky sa dajú použiť pre plánovanie trasy pre poľnohospodárske stroje pri práci na poli. Väčšina týchto algoritmov sa zaoberá pokrytím menších, vopred známych plôch s využitím menších strojov, ako sú vysávače, alebo záhradné kosačky. Niektoré z týchto algoritmov sa však dajú využiť pre splnenie cieľov tejto práce, avšak bude potrebné vykonať isté úpravy a pridať nové princípy.

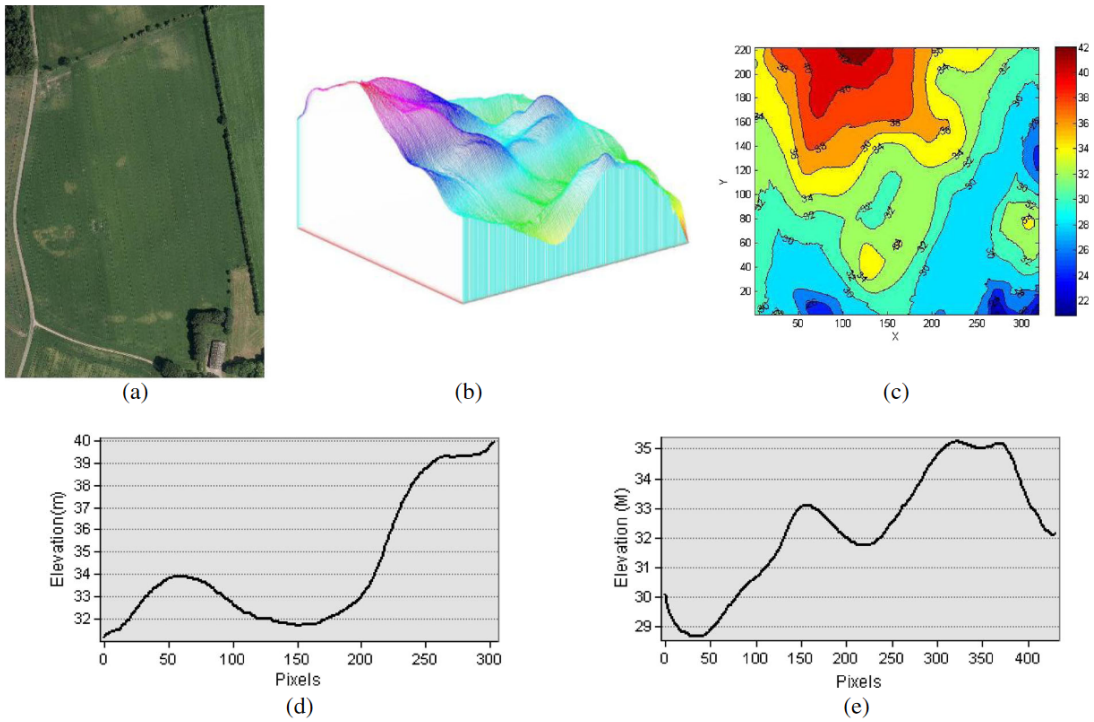
Kapitola 5

Aplikácia plánovania trasy v poľnohospodárstve

V tejto časti sa zameriame na niektoré implementácie "coverage path planning". V literatúre sú rôzne metódy na riešenie tohto problému. V článku [29] autori popisujú 2 nové algoritmy, oba sú to "greedy" algoritmy. Cieľom prvého algoritmu je rozdeliť pole na menšie časti, s ktorými sa bude lepšie pracovať. Tento algoritmus využíva algoritmus lichobežníkového rozkladu a taktiež vyhľadáva najlepší smer jazdy a čo najlepšie rozdelenie poľa na menšie časti. Druhý algoritmus, podobne ako prvý, je inkrementálny, ale cesta je plánovaná na základe aktuálneho stavu stroja a namiesto ďalšieho podpoľa sa vyhľadáva ďalší riadok. Oba algoritmy majú výhody, ale aj nevýhody a ani jeden z nich nie je optimálny pri plánovaní trasy. Prvý algoritmus je typu offline, zatiaľ čo ten druhý je typu online. Môže sa stať, že tieto algoritmy nenájdu rozumnú cestu, ale dokážu pracovať s poľami komplexných tvarov, s rôznymi zákrutami a prekážkami. Algoritmy neboli testované v reálnom prostredí a sú NP-ťažké.

Autor článku [21] sa zameral na minimalizovanie času presunov po poli a minimalizovanie spotreby pohonných hmôt. Pri plánovaní trasy berie do úvahy 3D modely terénu, viz. obrázok 5.1, ktoré pomáhajú vytvoriť plán cesty tak, aby sa zohľadňoval sklon terénu a tým pádom aj spotrebu paliva, čo priamo ovplyvní množstvo CO_2 a NO, ktoré sú jednými z hlavných dôvodov znečistenia ovzdušia a skleníkového efektu. Na základe výsledkov z dvoch oblastí štúdie sa ukázalo, že zníženie energetickej náročnosti pri optimalizácii uhla jazdy bolo priemerne 6,5% v porovnaní s prípadom, keď bol uhol jazdy optimalizovaný za predpokladu rovnomerného terénu. Taktiež sa ukázalo, že úspora času môže byť 10-15%. Autor využil genetický algoritmus, ktorý bol použitý v statickom prostredí a je typu offline. Môže byť použitý v komplexných a rôzne tvarovaných poliach. Podobne ako predošlé algoritmy, aj tento je NP-ťažký. Môže sa však stať, že algoritmus nájde cestu, ktorá bude mať dlhší prevádzkový čas stroja a tým aj vyššie náklady.

V inej štúdií [20] sa opäť rieši plánovanie trasy v 3D teréne. Autori popisujú, že algoritmy plánovania trasy pokrytia v 3D teréne, ktoré sa využívali v čase písania tejto štúdie, boli jednoducho algoritmy, ktoré pracovali v 2D teréne a boli len premietnuté do 3D terénu. Pri tomto premietaní sa však skutočná vzdialenosť medzi susednými cestami na topografickom povrchu buď zväčšuje, alebo zmenšuje, čo môže spôsobiť to, že sa isté časti poľa vynechajú, alebo iné budú prekryté viackrát. To môže viesť k neefektívnemu využívaniu pôdy a zdrojov. Autori štúdie navrhli numerický prístup na odhad vynechaných alebo viackrát pokrytých oblastí a využili ho na určenie optimálneho uhla jazdy, ktorý minimalizuje



Obr. 5.1: Experimentálne pole: a) satelitná snímka, b) 3D model poľa, c) vrstevnicový pohľad, d) nadmorská výška zo západu na východ, e) nadmorská výška zo severu na juh [21].

tento vplyv. Taktiež bol vyvinutý nový prístup plánovania trasy pokrytia 3D priestoru, takzvaný "side-by-side", ktorý predchádza vynechávaniu alebo viacnásobnému prekrytiu oblastí bez ohľadu na typografickú povahu terénu poľa. Prístup poukázal na skutočnosť, že je potrebný vylepšený algoritmus 3D pokrytia schopný eliminovať typografický vplyv na priestory medzi riadkami. Tieto prístupy boli testované na hypotetickom testovacom poli a taktiež na skutočnom experimentálnom poli s nerovným terénom. Prístupy ilustrovali, že pri použití vhodného uhla jazdy by sa dalo ušetriť významné percento nepokrytej plochy a to v rozsahu 2-14%. Algoritmus je typu offline, pracuje so statickým prostredím rôzneho typu, ale nedokáže určiť rozdiel medzi nepokrytou a viackrát pokrytou plochou.

Kapitola 6

Návrh programu

Prvým krokom pri tvorbe programu je jeho návrh. Keďže bude program slúžiť na plánovanie trasy pre poľnohospodárske stroje na poliach, bude potrebné získať informácie o týchto poliach. Konkrétne sa tým myslí získanie GPS súradníc všetkých okrajových bodov poľa. Tieto informácie sú verejne dostupné na internete. Keďže má program brať do úvahy aj sklon poľa, je potrebné zistiť aj nadmorskú výšku v rôznych častiach poľa, ideálne získať celkový 3D model poľa. To však nebude možné, pretože neexistuje 3D model pre každé jedno pole, takže bude potrebné získať len nadmorskú výšku určitých bodov poľa a z týchto informácií vytvoriť jeho približný model. Aj keď sa dajú využiť rôzne služby, ktoré pri zadaní GPS súradníc vrátia nadmorskú výšku v danom bode, je otázne, ako spoľahlivé sú tieto informácie. Keď už budeme mať informácie o mape poľa, bude ju potrebné vykresliť a určiť počiatočný a cieľový bod pre stroj pracujúci na danom poli. Taktiež je potrebné, aby mal program rozhranie pre pridávanie, úpravu a odstraňovanie strojov a ich príslušenstva. Takto budú zaznamenané informácie o strojoch, ako napríklad typ stroja, značka, popis, výkon motora, rýchlosť, polomer otáčania, možnosti, aké príslušenstvo je so strojom kompatibilné a taktiež informácie o príslušenstve, ako jeho typ, značka, šírka záberu, kompatibilné stroje, minimálny výkon stroja pre dané príslušenstvo a podobne.

Základnou časťou programu bude algoritmus na plánovanie trasy. Ako bolo zistené pri štúdovaní súčasného stavu a používaných algoritmov v tejto oblasti, neexistuje nejaký všeobecný algoritmus pre plánovanie trasy pre poľnohospodárske stroje, vždy sa algoritmy zameriavajú na nejakú konkrétnu činnosť. Tento program však pri plánovaní trasy musí brať do úvahy obmedzenia reálnych strojov, ako je šírka záberu, polomer otáčania, maximálna rýchlosť ale hlavne to, o aký stroj ide a akú činnosť vykonáva. Napríklad trasa pre prípravu poľa na sejbu bude iná, ako trasa pre sejbu a tá bude zase iná, ako trasa pre kombajny pri zbere úrody. Všetky tieto činnosti majú rôzne obmedzenia, napríklad je nežiadúce, aby stroj pri aplikácií hnojív prechádzal viackrát po rovnakej oblasti, ale pri zbere úrody nevadí, ak kombajn prejde po oblasti, ktorá bola už spracovaná. Algoritmus musí pri plánovaní trasy brať do úvahy aj parametre ako sú čas, prejdená vzdialenosť, alebo spotreba paliva, čo súvisí so správnym plánovaním vzhľadom na sklon poľa. Po naplánovaní trasy ju bude potrebné zobraziť a prípadne uložiť.

Pri návrhu programu sa počíta s využitím nasledujúcich technológií:

6.1 Python

Na tvorbu práce bude použitý jazyk Python 3 a to z nasledujúcich dôvodov: Je to objektovo orientovaný programovací jazyk, ktorý spája vlastnosti flexibility a použiteľnosti skriptovacích jazykov a výkonného návrhu tradičných programovacích jazykov. Práca s ním je pomerne jednoduchá a produktívna, programy sa ľahko implementujú. Taktiež sa programy vyvíjajú rýchlejšie a to hlavne vďaka tomu, že je objektovo-orientovaný. To prispieva k jednoduchšej čitateľnosti kódu a hlavne sa program v budúcnosti ľahšie udržuje. Výhodou Pythonu je veľká komunita vývojárov, ktorá poskytuje silnú základňu zdieľaných vedomostí a podpory. Taktiež existuje nespočet voľne dostupných knižníc na rôzne použitia. Ďalšou výhodou je, že na spustenie programov napísaných v Pythone nie je potrebný prekladač, stačí mať na zariadení nainštalovaný interpret jazyka Python alebo prípadne vytvoriť .exe súbor. V minulosti bolo menšou nevýhodou interpretovaných programovacích jazykov to, že ich výsledné programy boli pomalšie ako programy kompilovaných programovacích jazykov, ale táto nevýhoda sa pri dnešných možnostiach hardvéru a softvéru už stráca. Samozrejme, že pri veľmi náročných aplikáciách sú rýchlejšie programy napísané v programovacích jazykoch nižšej úrovne, ale pre jednoduché programy to nezohráva úlohu. Python je taktiež považovaný za jeden z najlepších jazykov pre tvorbu grafického užívateľského rozhrania (GUI). Má zabudované možnosti vývoja GUI, alebo je možné použiť rôzne voľne dostupné frameworky. Je ľahko prenositeľný a beží na akejkoľvek platforme: Windows, Unix, Linux.... Python komunikuje bez problémov takmer s hocičím. Ak zistíte, že sa dá nejaká časť programu spraviť efektívnejšie v inom jazyku, nie je s tým problém. Python je v tejto práci využitý ako na tvorbu frontendu, čiže zobrazenia aplikácie, tak aj na backend a to jednak na prácu so vstupnými súbormi s poľami a strojmi, tak aj na plánovanie trasy a jej zobrazenie.

6.2 PyQt

Na jednoduchšie vytváranie programov a ich zobrazenia je výhodné Python rozšíriť o knižnice, ktoré nám zjednodušia tieto programy vytvárať rýchlejšie a efektívnejšie. V tomto prípade ide o použitie knižnice Qt, konkrétne verziu pre Python - PyQt, ktorá slúži na vytvorenie rozhrania programu. Qt je jedna z najpopulárnejších multiplatformových knižníc pre vytváranie programov s grafickým užívateľským rozhraním. Je napísaná v programovacom jazyku C++, ale dá sa použiť aj v jazyku Python, C, Java.... Je taktiež multiplatformná. Má veľmi dobre spracovanú dokumentáciu a vývojové programy Qt Creator alebo Qt Designer. Aplikácie vytvorené pre grafické užívateľské prostredie používajú natívny vzhľad operačného systému, takže vytvorené programy sa vždy prispôsobia vzhľadu používateľského prostredia.

6.3 Mplleaflet

Python je jeden z najuniverzálnejších programovacích jazykov, nie len že poskytuje moduly a knižnice na prácu s objemnými dátami, je taktiež nápomocný v mnohých doménach, ako napríklad v mapových službách. Ak chceme zobraziť niečo na mape na základe zemepisnej šírky a dĺžky, existuje na to viac spôsobov, jedným z nich je využiť knižnicu Mplleaflet. Táto knižnica dokáže konvertovať matplotlib zobrazenie na webovú stránku, ktorá obsahuje

zoomovateľnú mapu. Jej cieľom je využiť Python a matplotlib na vizualizáciu geografických dát na prehľadnej mape bez potreby písania HTML alebo Javascriptového kódu.

6.4 Zdroj dát

Na plánovanie trasy na poli samozrejme potrebujeme zistiť hranice jednotlivých polí, ale aj možné prekážky, ktoré sa na nich nachádzajú. Bolo skúmaných množstvo zdrojov dát, či už register s pozemkami, alebo rôzne mapy, kde bol však ten problém, že každé pole bolo rozdelené na jednotlivé parcely, kde každú parcelu mal iný vlastník, alebo sa na poliach nenachádzali informácie o prekážkach. Nakoniec bol využitý portál eagri.cz [3]. Hlavnou ideou tohoto webového portálu bolo vytvoriť jeden centrálny prístupový bod k informačným zdrojom Ministerstva poľnohospodárstva a jeho podriadených organizácií. Jadro portálu bolo vytvorené zlúčením stránok pozemkových úradov, portálu farmára a začlenením portálu siete pre obce. Portál zjednocuje webovú prezentáciu dátových zdrojov v ktorejkoľvek časti portálu. Medzi jeho prínosy patrí:

- Centrálne vyhľadávanie informácií vo všetkých podriadených weboch.
- Rovnakú logiku prezentácie informácií a ovládacích prvkov.
- Trvalo aktualizovaná databáza právnych predpisov v konsolidovaných zneniach.
- Zoskupenie všetkých formulárov na jednom mieste.
- Sprehľadnenie prezentovaných informácií rozdelením portálu na tematické sekcie.

Jednou z týchto sekcií je verejný export dát LPIS, čiže register poľnohospodárskej pôdy. Je tu možné vyexportovať si dáta z každého katastrálneho územia. Exportované dáta sú vo formáte XML s nasledujúcou štruktúrou:

```
<ns2:DATZMENYDPB></ns2:DATZMENYDPB> - datum poslednej zmeny
        <ns2:DPB> - jedno pole
                <ns2:IDDPB></ns2:IDDPB>
                <ns2:CTVEREC></ns2:CTVEREC>
                <ns2:ZKOD></ns2:ZKOD> - oznacenie pola
                <ns2:STAV></ns2:STAV>
                <ns2:STAVID></ns2:STAVID>
                <ns2:UCINNOST_DLE_ZAKONA></ns2:UCINNOST_DLE_ZAKONA>
                <ns2:GEOMETRIE></ns2:GEOMETRIE> - hranice pola a prekazok
                <ns2:PLATNOSTOD></ns2:PLATNOSTOD>
                <ns2:VYMERA></ns2:VYMERA>
                <ns2:VYMERAOPV></ns2:VYMERAOPV>
                <ns2:KULTURA></ns2:KULTURA>
                <ns2:KULTURAID></ns2:KULTURAID>
                <ns2:KULTURANAZEV></ns2:KULTURANAZEV>
                <ns2:KULTURAOD></ns2:KULTURAOD>
                <ns2:UZIVATEL>
                        <ns2:IDUZIVATELE></ns2:IDUZIVATELE>
                        <ns2:OBCHODNIJMENO></ns2:OBCHODNIJMENO>
                        <ns2:IC></ns2:IC>
```

```
                <ns2:PRAVNIFORMA></ns2:PRAVNIFORMA>
            </ns2:UZIVATEL>
    ...
</<ns2:DPB>
```

V tejto štruktúre boli vynechané niektoré položky. Dôležité položky sú pre nás <ns2:DPB>, čo oddeľuje jednotlivé polia, <ns2:ZKOD>, čo je číselná identifikácia poľa v súbore a <ns2:GEOMETRIE>, kde sú jednotlivé body hraníc poľa a prekážok v ňom. Tento súbor však neobsahuje informácie o mieste, ktoré sa využíva ako vstup na pole a taktiež na ňom nie sú uvedené žiadne informácie o nadmorskej výške, preto je vhodné doplniť do tejto štruktúry GPS súradnice vstupu na pole a taktiež vektor, ktorý bude slúžiť ako smer, v ktorom sa bude stroj po poli pohybovať. Detailnejšie sa tomuto budeme venovať v ďalších častiach práce, ale v skratke ide o to, aby mal užívateľ možnosť zadať uhol, v ktorom sa bude stroj po poli pohybovať. Je to z toho dôvodu, ak by bol smer pohybu stroja vypočítaný algoritmom neoptimálny a to či už z nepravidelného tvaru poľa, alebo z dôvodu prechádzania stroja poľom po vrstevniciach, pretože nebolo možné získať dáta o nadmorskej výške poľa. Bolo vykonaných viacero pokusov o získanie dát o nadmorskej výške polí a jednotlivých bodov v nich, žiaľ neúspešne. Niektoré zdroje dát boli nepresné, niektoré nespoľahlivé, niektoré nefungovali pre Českú republiku, alebo získavanie dát prebiehal príliš dlho. Preto je najlepšie do mapy pridať informáciu o optimálnom smere jazdy pri členitých poliach manuálne, pretože ľudia, ktorí spravujú tieto polia najlepšie vedia, akým smerom je po nich najoptimálnejšie jazdiť. XML súbor je preto rozšírený o tieto informácie:

```
<ns2:VSTUP></ns2:VSTUP>
<ns2:UHOL></ns2:UHOL>
```

Vstup bude obsahovať dva čísla oddelené medzerou a to konkrétne zemepisnú šírku, resp. dĺžku. Uhol bude taktiež reprezentovaný dvoma číslami, vektorom, ktorý udáva smer pohybu stroja po poli.

Zdrojom dát pre stroje a ich príslušenstvo bude súbor, ktorý sa musí volať `equipment.xml` a musí byť v rovnakej zložke, ako je program. Pri spustení programu bude tento súbor automaticky spracovaný. Je to z toho dôvodu, že s programom bude pracovať užívateľ, ktorý používa spravidla svoje stroje, alebo stroje zo spoločnosti, v ktorej pracuje, preto nebude nutné pri každom spustení aplikácie nutné vyberať súbor so strojmi. Formát xml súboru so strojmi vyzerá nasledovne:

```
<tractors>
            <tractor>
                    <name></name>
                    <type></type>
                    <hp></hp>
                    <speed></speed>
                    <radius></radius>
            </tractor>
</tractors>
<harvesters>
            <harvester>
                    <name></name>
                    <type></type>
                    <hp></hp>
                    <speed></speed>
```

```
                <radius></radius>
            </harvester>
</harvesters>
<headers>
            <header>
                <name></name>
                <type></type>
                <compatible></compatible>
                <speed></speed>
                <width></width>
            </header>
</headers>
<plows>
            <plow>
                <name></name>
                <type></type>
                <hp></hp>
                <speed></speed>
                <width></width>
            </plow>
</plows>
<cultivators>
    ...
</cultivators>
<seeders>
    ...
</seeders>
<sprayers>
    ...
</sprayers>
```

Kapitola 7

Implmentácia

Implementácia programu sa skladá z viacerých častí, konkrétne spracovaním súboru so strojmi a ich príslušenstvom, spracovanie súboru s poľami, vytvorenie rozhrania, pomocou ktorého bude možné tieto stroje spracovávať a vyberať si medzi jednotlivými poľami a najviac podstatná časť - samotné plánovanie trasy.

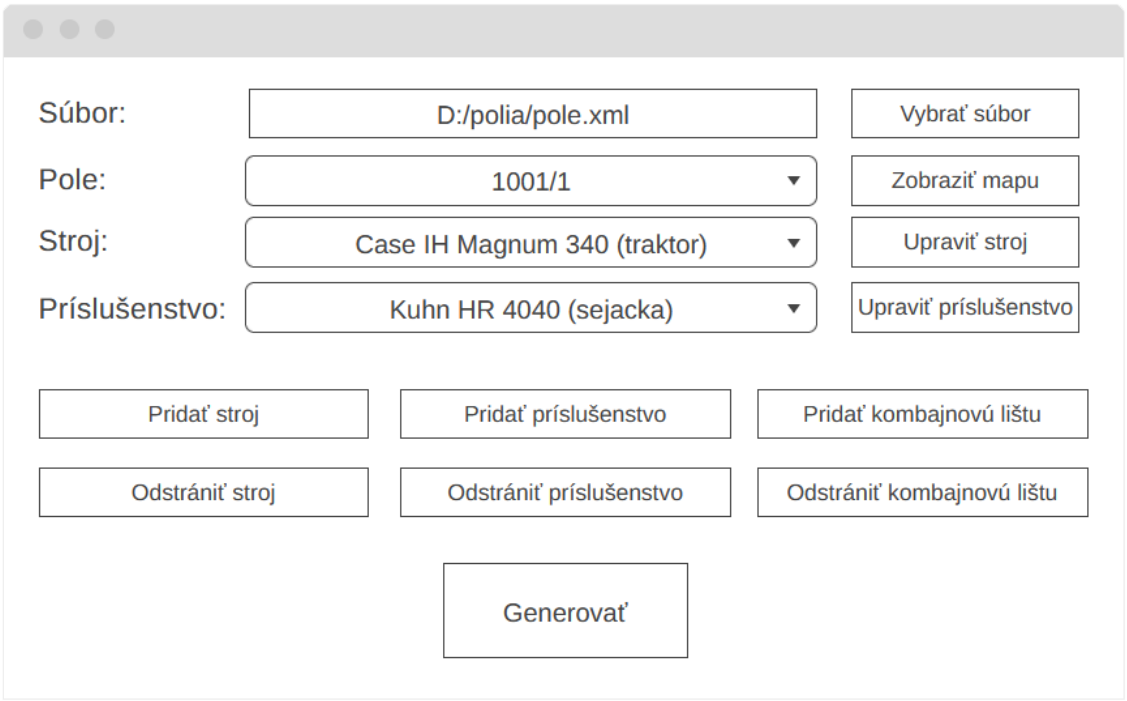
7.1 Frontend

Táto časť programu sa venuje vizuálnej stránke aplikácie. Pred samotným programovaním je vhodné vytvoriť grafický návrh aplikácie, určiť, aké prvky sa v nej budú vyskytovať, aký budú mať vzhľad a rozloženie, aby bol program čo najviac užívateľsky prívetivý. Tu sa naskytuje otázka, pre koho bude aplikácia určená. Tento program na plánovanie trasy pre autonómne poľnohospodárske stroje bude určený predovšetkým pre pracovníkov v tejto oblasti, čiže farmárov, pracovníkov v oblasti poľnohospodárstva, majiteľov poľnohospodárskych pozemkov, alebo majiteľov spoločností poskytujúcich služby v oblasti poľnohospodárstva. Aj keď už dnes práca s počítačom bežná vec a myslíme si, že s nimi vie pracovať každý, musíme predpokladať, že s touto aplikáciou môžu v budúcnosti pracovať aj ľudia, ktorí nemajú veľké zručnosti čo sa týka oblasti informačných technológií a práce s počítačmi, alebo nie sú technicky veľmi zruční. Z tohto dôvodu musí byť aplikácia navrhnutá tak, aby bola dostatočne prehľadná pre každého užívateľa, odhliadnuc od jeho technických zručností. Na počiatočný návrh rozloženia prvkov aplikácie je vhodné využiť pero a papier a načrtnúť si ich rozloženie, prípadne využiť jeden z mnohých softvérov, ktorý takúto možnosť ponúka.

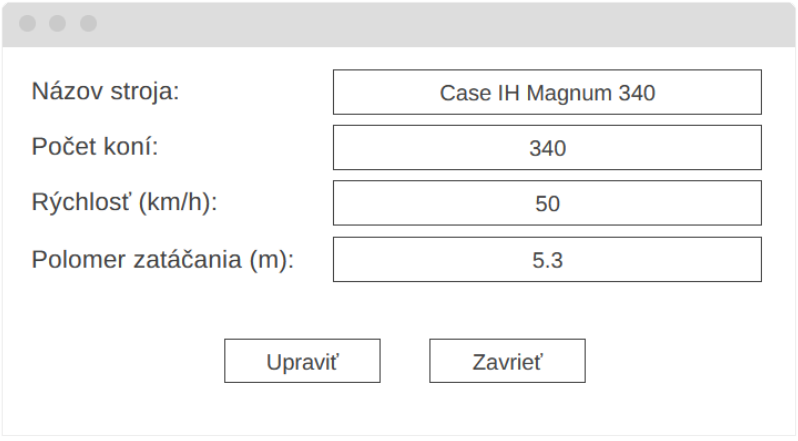
Zo zadania práce vyplýva, že sa tam bude pracovať so strojmi a s poľami. Čo sa týka polí, tak tie sa budú načítavať z nejakého súboru, preto je potrebné mať v aplikácii možnosť výberu tohoto súboru, a zobrazenie jeho názvu, prípadne cesty k nemu. Keďže v danom súbore bude viac polí, užívateľ by mal byť schopný si z týchto polí vybrať to, na ktorom plánuje vykonávať určitú prácu. Na to je ideálne využiť rolovací zoznam, v ktorom budú vypísané názvy alebo označenie polí. Môže však nastať situácia, kedy užívateľ nepozná názvy alebo označenie týchto polí, alebo je tých polí viac a majú podobné označenie. V tejto situácii je vhodné, aby bolo možné si v aplikácii vykresliť mapu so všetkými poľami a užívateľ si bude môcť vizuálne vyhľadať konkrétne pole. Na to je vhodné do aplikácie pridať nejaké tlačidlo umiestnené v sekcii pre správu polí, po ktorého stlačení sa objaví mapa so všetkými poľami v súbore. To sú všetky operácie, ktoré by mohol užívateľ vykonávať s poľami.

Predpokladá sa, že užívateľ, či už je to farmár, alebo iný zamestnanec, bude pracovať stále s rovnakými strojmi, resp. so svojimi strojmi. Preto v aplikácii nebude možnosť výberu súboru, z ktorého sa budú stroje načítavať, ale tento súbor bude pevne daný a uložený v adresári aplikácie. Tým pádom ho nebude nutné po spustení aplikácie vyberať, ale hneď sa zobrazia dostupné stroje a príslušenstvo k nim. Pod sekciou na prácu s poľami sa teda bude nachádzať sekcia na správu strojov a k nim patriacemu príslušenstvu. Keďže sa vopred nevie, koľko sa ich bude využívať, opäť je vhodné na ich výber využiť rolovacie zoznamy. Prvý rolovací zoznam bude obsahovať dostupné stroje a druhý zoznam bude obsahovať príslušenstvo, ktoré je kompatibilné s vybraným strojom. Bude vhodné, ak bude každá položka v zozname označená menom, ale aj typom stroja a to pre lepšiu prehľadnosť a rýchlejšie vyhľadávanie. Samozrejme je potrebné, aby sa tieto stroje dali pridávať, upravovať a odstraňovať. Na tieto účely budú pridané samostatné tlačidlá. Keďže sa rozlišujú stroje a ich príslušenstvo, je pre každý tento typ potrebné vlastné zobrazenie. Samozrejme to platí aj pre kombajnové lišty. Budú teda vytvorené tlačidlá na pridávanie, úpravu a odstraňovanie strojov, následne pridávanie, úpravu a odstraňovanie príslušenstva a v neposlednom rade pridávanie a odstraňovanie kombajnových líšt. Pri stlačení tlačidla na pridávanie strojov sa zobrazí okno, v ktorom bude potrebné zadať typ stroja (bude na výber z možností v rolovacom zozname), jeho názov, počet konských síl, maximálnu rýchlosť a polomer zatáčania. Pri úprave stroja sa zobrazí podobné okno, ale nebude tam možnosť na zmenu typu stroja, všetky ostatné položky budú rovnaké ako pri pridávaní stroja, ale budú vyplnené na základe stroja, ktorý je momentálne vybraný v rolovacom zozname výberu strojov. V prípade odstraňovania strojov sa zobrazí okno, v ktorom sa bude nachádzať rolovací zoznam so všetkými strojmi. Po stlačení tlačidla odstrániť sa zmaže aktuálne vybraný stroj. Pridávanie príslušenstva bude obdobné ako pridávanie strojov, vyberie sa typ príslušenstva z rolovacieho zoznamu a vyplní sa jeho názov, minimálny počet konských síl stroja, na ktorom sa dá dané príslušenstvo používať, maximálna rýchlosť pri používaní a šírka záberu stroja. Odstraňovanie bude fungovať rovnako ako pri strojoch. Líšiť sa však bude úprava príslušenstva, kde sa bude rozlišovať príslušenstvo za traktor a kombajnové lišty. Pri stlačení tlačidla na úpravu príslušenstva sa zobrazí rolovací zoznam, kde sa vyberie konkrétne príslušenstvo alebo kombajnová lišta. Po výbere príslušenstva za traktor sa objaví okno s možnosťou zmeny jeho atribútov rovnako ako pri strojoch. Pri výbere kombajnovej lišty sa však objaví okno, v ktorom sa okrem atribútov dá určiť, ktoré kombajny sú s danou lištou kompatibilné. Na to budú slúžiť dva rolovacie zoznamy, v jednom budú kompatibilné kombajny s možnosťou ich odstránenia, v druhom budú kombajny, ktoré nie sú kompatibilné, ale bude možnosť ich medzi kompatibilné pridať. Ostávajú už len možnosti na pridanie kombajnovej lišty, ktorá vyzerá obdobne ako pri pridávaní strojov alebo príslušenstva, len s tým, že sa tam zadá jej názov, šírka a maximálna rýchlosť pri práci. Zvyšné tlačidla slúžia na odstránenie príslušenstva a kombajnových líšt. Posledným prvkom bude tlačidlo na generovanie trasy na základe vybraného poľa, stroja a príslušenstva, po ktorého stlačení sa vygeneruje trasa a zobrazí sa na mape. Na obrázkoch 7.1 až 7.5 je možné vidieť grafický návrh rozloženia prvkov a niektorých okien.

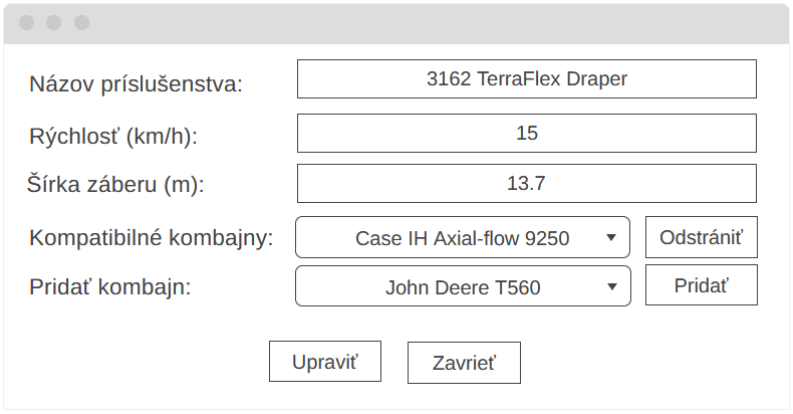
Ako bolo spomenuté vyššie, frontendová časť je vytvorená pomocou PyQt a je uložená v súbore `app.py`. Na začiatku súboru sa nachádza importovanie všetkých potrebných knižníc. Nasleduje základná trieda `MainWindow`, ktorá obsahuje premenné `vehicles` - zoznam strojov, `equipments` - zoznam príslušenstva, `fields` - zoznam polí, `inputFile` - názov vstupného súboru s poľami, `equipmentFile` - názov súboru so strojmi a `equipmentTree` - spracovaný súbor so strojmi v podobe stromovej štruktúry. Následne je zavolaná funkcia `initUI`, ktorá vytvorí základné okno aplikácie so všetkými nápismi, rolovacími zoznamami a tlačid-



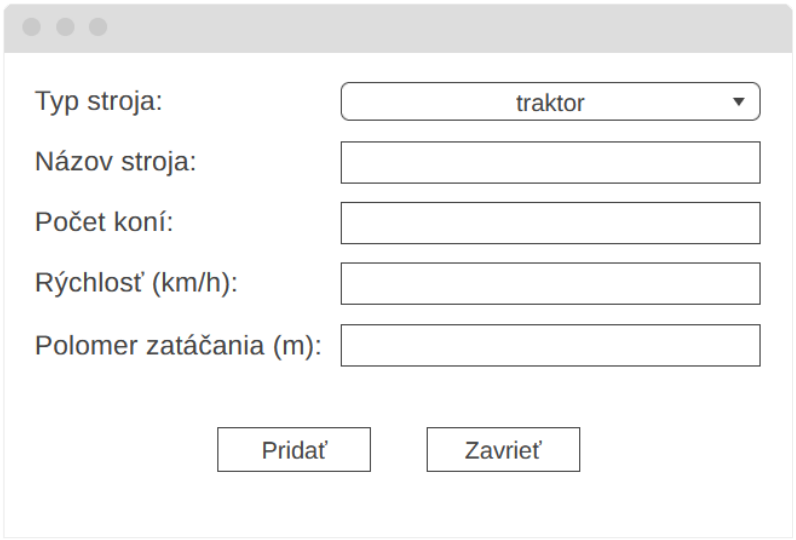
Obr. 7.1: Grafický návrh základného zobrazenia aplikácie



Obr. 7.2: Grafický návrh okna na úpravu strojov



Obr. 7.3: Grafický návrh okna na upravovanie kombajnových líšt



Obr. 7.4: Grafický návrh okna na pridávanie strojov

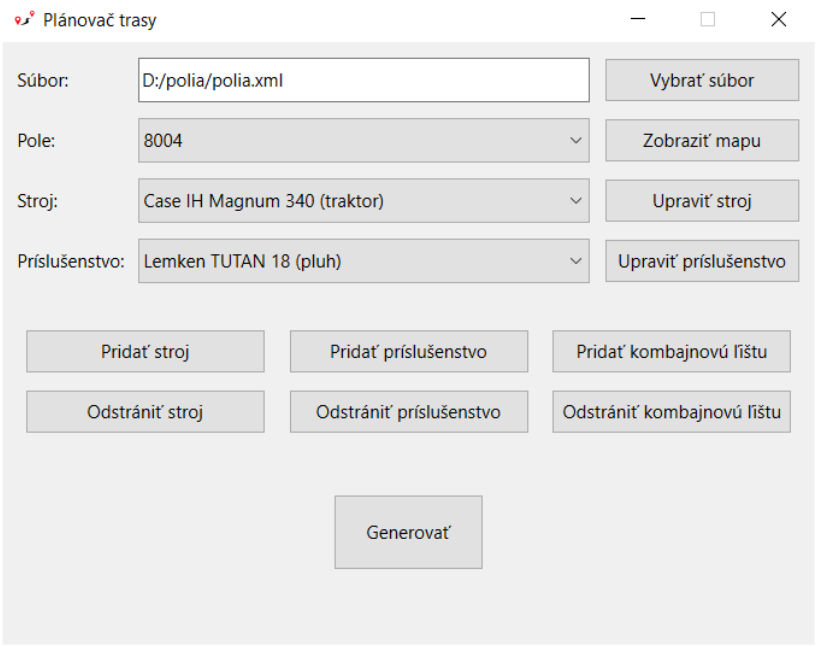
Obr. 7.5: Grafický návrh okna na odstraňovanie strojov

lami, ktorým sú priradené konkrétne funkcie. Ďalej budú vymenované a popísané jednotlivé metódy, mnohé z nich pracujú aj s backendom, konkrétne so spracovávaním súboru so strojmi.

- **disableButton**: Zablokovanie tlačidla na odstránenie stroja v prípade, že boli všetky stroje odstránené.
- **editVehicleDialog**: Vytvorenie okna na editáciu strojov. Najskôr sa vytvorí okno, následne sa získajú dáta o vybranom stroji a pomocou týchto dát sa vyplnia vytvorené elementy v okne. Nakoniec sú vytvorené tlačidlá na úpravu stroja a zatvorenie okna. Pri kliknutí na tlačidlo Upraviť sa spustí funkcia na overenie správnosti vstupu.
- **editVehicle**: Funkcia na overenie správnosti vstupu. Kontroluje sa, či sú zadané číselné hodnoty naozaj čísla. Ak sa vstupné dáta nepodarí overiť (napr. pre rýchlosť vozidla je vložený text a nie číslo), je na to užívateľ upozornený. V prípade úspešného overenia vstupu je zavolaná jedna z funkcií backendu, ktorým sa budeme venovať v ďalšej časti.
- **chooseEditEquipmentDialog**: Vytvorenie okna na výber príslušenstva alebo kombajnových líšt, ktoré sa majú upraviť. Po stlačení tlačidla Upraviť sa zobrazí okno na úpravu vybraného príslušenstva.
- **editEquipmentDialog**: Funkcia, ktorá sa spustí pomocou predchádzajúcej funkcie. Je vytvorené okno na úpravu príslušenstva, alebo kombajnovej lišty. Vzhľad okna je vytvorený na základe typu zvoleného príslušenstva. V prípade kombajnovej lišty je možné upraviť jej atribúty, ale aj kompatibilné kombajny s danou lištou. Najskôr sú získané kompatibilné a nekompatibilné kombajny pre danú lištu, následne sú vytvorené dva rolovacie zoznamy, jeden pre kompatibilné kombajny s možnosťou zvolený kombajn z kompatibilných odstrániť a podobne aj druhý pre nekompatibilné kombajny s možnosťou zvolený pridať medzi kompatibilné. V prípade ostatného príslušenstva sa upravujú len ich atribúty.
- **deleteCompatible**: Funkcia na odstránenie kombajnu z kompatibilných kombajnov pre danú lištu.
- **addCompatible**: Funkcia na pridanie kombajnu ku kompatibilným kombajnom pre danú lištu.
- **editEquipment**: Funkcia na overenie vstupu z funkcie na editáciu príslušenstva, opäť je overený vstup, v prípade úspechu je príslušenstvo upravené, v opačnom prípade je užívateľ upozornený na nesprávny vstup.
- **editHeader**: Funkcia rovnaká ako pre editáciu príslušenstva, len sa upravuje kombajnová lišta.
- **addVehicleDialog**: Funkcia na vytvorenie okna na pridanie nového stroja.
- **addVehicle**: Funkcia na overenie vstupu pri pridávaní nového stroja, upozornenie užívateľa v prípade neúspechu overenia, v opačnom prípade pridanie nového stroja.

- **removeVehicleDialog**: Vytvorenie okna na odstránenie stroja. V okne je vytvorený rolovací zoznam, ktorý obsahuje všetky stroje a tlačidlo na odstránenie práve zvoleného stroja v zozname. Ak je rolovací zoznam prázdny, je tlačidlo na odstránenie zablokované.
- **removeVehicles**: Funkica na odstránenie stroja a obnovenie rolovacích zoznamov so strojmi a príslušenstvom.
- **addEquipmentDialog**: Funkcia vytvorí okno na pridanie nového príslušenstva. Z rolovacieho zoznamu je možné vybrať typ príslušenstva, následne sú vyplnené jeho atribúty.
- **addEquipment**: Overenie vstupu z predchádzajúcej funkcie, v prípade úspechu pridanie nového príslušenstva
- **removeEquipmentDialog**: Vytvorenie okna na odstránenie príslušenstva. Funkcia podobná ako pri odstraňovaní strojov, je vytvorený rolovací zoznam so všetkým príslušenstvom, po stlačení tlačidla Odstrániť sa spustí nasledujúca funkcia. Ak je rolovací zoznam prázdny, je tlačidlo na odstránenie zablokované.
- **removeEquipments**: Funkcia na odstránenie príslušenstva a obnovenie rolovacích zoznamov.
- **addHeaderDialog**: Vytvorenie okna na pridanie kombajnovej lišty.
- **addHeader**: Funkcia na overenie vstupu z predchádzajúcej funkcie a v prípade úspechu pridanie novej lišty.
- **removeHeaderDialog**: Vytvorenie okna na odstránenie kombajnovej lišty, funkcia podobná ako pri odstraňovaní strojov, či príslušenstva. Vytvorenie rolovacieho zoznamu, ktorý obsahuje všetky kombajnové lišty a možnosť ich odstránenia.
- **removeHeaders**: Funkcia na odstránenie kombajnovej lišty a obnovenie rolovacích zoznamov.
- **closeDialog**: Funkcia na zatvorenie vyskakovacích okien (dialógov).
- **getAttachment**: Funkcia, ktorá získa kompatibilné príslušenstvo pre stroj, ktorý je aktuálne zvolený v rolovacom zozname so strojmi. Najskôr je vyprázdnený zoznam s príslušenstvom a aj rolovací zoznam s príslušenstvom. Následne sa zistí, či je aktuálny stroj traktor, alebo kombajn a podľa toho je získané príslušenstvo, alebo kombajnové lišty. Následne je naplnený zoznam s príslušenstvom a taktiež aj rolovací zoznam s príslušenstvom.
- **showDialog**: Funkcia, ktorá vytvorí okno, v ktorom sa dá vybrať súbor s poľami. Je nastavené obmedzenie na výber súboru s príponou .xml. Po výbere súboru sa jeho absolútna cesta vypíše v okne aplikácie a je spustená funkcia na spracovanie polí a ich uloženie v premennej fields. Nakoniec sa rolovací zoznam s poľami naplní ich názvami.
- **showMap**: Funkcia zobrazí mapu so všetkými poľami v súbore. Konkrétne sú vykreslené vonkajšie hranice týchto polí bez prekážok, ktoré sa nachádzajú v ich vnútri. Okrem toho je vypočítaný stred každého poľa a na toto miesto je pridaný názov alebo označenie konkrétneho poľa. Mapa je zobrazená pomocou knižnice `matplotlib`, pretože `mplleaflet` nepodporuje vkladanie textu.

- **checkNumber**: Funkcia, ktorá kontroluje, či je vstup číslo, alebo nie a podľa toho vracia True alebo False.
- **generate**: Hlavná funkcia na generovanie samotnej trasy stroja pre zadané pole. Najskôr sú získané súradnice poľa, kde je jeho vstup a prípadný vektor, v ktorom sa má stroj pohybovať. Je taktiež získaný stroj a z neho získaný jeho polomer zatáčania a taktiež príslušenstvo, ktoré je k nemu pripojené a jeho šírka záberu. Následne sú vykreslené hranice poľa a to vrátane prekážok, ktoré sa v ňom nachádzajú. Ďalej sú všetky súradnice poľa prevedené do formátu, ktorý je požadovaný pre ďalšie spracovanie. Polomer zatáčania a šírka záberu sú vynásobené konštantou 0.00001393, ktorá bola získaná iteratívnym výpočtom tak, že bolo vytvorené pole o veľkosti 100x200 metrov a na ňom bola plánovaná trasa pre stroj s príslušenstvom o šírke 10 metrov. Konštanta bola upravovaná dovtedy, kým vzdialenosť dvoch vedľa seba ležiacich bodov nebola presne 10 metrov. Nakoniec je zavolaná funkcia na plánovanie trasy so zadanými parametrami pre konkrétne pole. Ak sa plánovanie podarilo, je vykreslená mapa poľa aj s trasou pomocou spomínanej knižnice `mplleaflet`, ak sa to nepodarí, je o tom užívateľ informovaný.



Obr. 7.6: Základné zobrazenie programu

7.2 Backend

Ako bolo spomenuté skôr, funkcie frontendu sú prepojené s funkciami backendu. Ten má viacero častí - spracovanie máp, spracovanie strojov a plánovanie trasy. Funkcie na spracovanie máp a strojov sa nachádzajú v súbore `parser.py`. Tento súbor obsahuje nasledujúce funkcie:

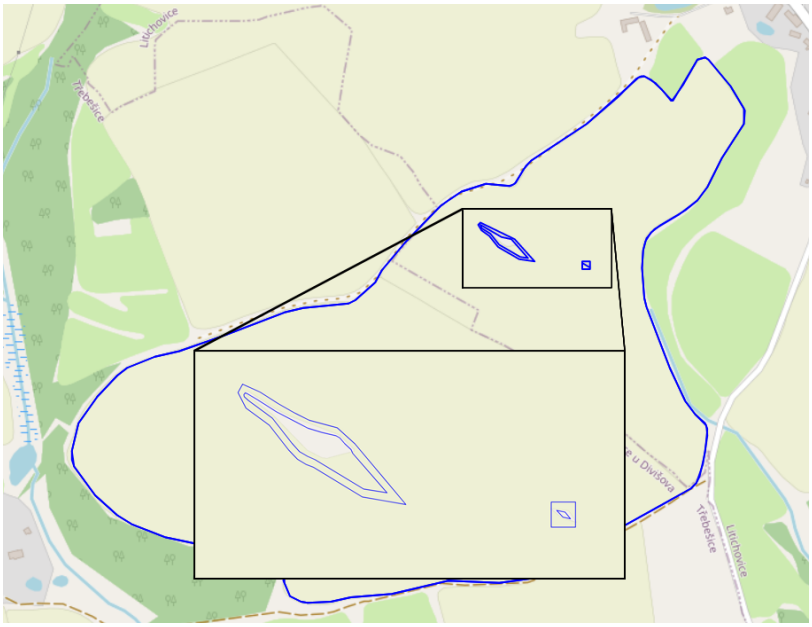
- **parseFields**: Funkcia, ktorá spracuje vstupný súbor s poľami predaný ako parameter a vráti zoznam všetkých polí v súbore so všetkými informáciami o nich.
- **getCoords**: Funkcia na získanie súradníc okrajov polí a prekážok v nich. Najskôr sa získa pole dané indexom, ktorý bol predaný ako parameter a získajú sa súradnice, ktoré však nie sú upravené, preto je ich potrebné rozdeliť, odstrániť zátvorky a uložiť pre ďalšie spracovanie. Tieto súradnice sú však vo formáte S-JTSK, takže je ich potrebné transformovať do formátu GPS. To je dosiahnuté použitím knižnice `Transformer` a funkcie `from_crs` s argumentmi 'EPSG:5514', 'EPSG:4326'. Výsledkom funkcie je teda zoznam GPS súradníc okrajov poľa a prekážok na ňom.
- **getEnter**: Ako bolo spomenuté v časti o zdroji dát, je možné do nich pridať GPS súradnice vstupu na pole. Táto funkcia slúži na získanie týchto dát pre konkrétne pole zadané jeho indexom.
- **getSweepVec**: Podobne ako predchádzajúca funkcia, ale táto vracia vektor, v ktorom sa bude stroj pohybovať po poli. Tento vektor nemusí byť zadaný, môže však pomôcť pri plánovaní optimálnejšej trasy vzhľadom na topológiu poľa.
- **parseVehicles**: Funkcia na spracovanie strojov. Vracia zoznam obsahujúci všetky stroje - teda traktory a kombajny.
- **parseEquipments**: Funkcia vracia príslušenstvo, ktoré je možné pripojiť za traktor, konkrétne pluhy, kultivátory, sejačky a postrekovače.
- **parseHeaders**: Funkcia vracia kombajnové lišty.
- **remove**: Funkcia slúži na odstránenie stroja, príslušenstva, alebo kombajnovej lišty na základe zadaného názvu. Funkcia prejde celý XML strom, ak sa jej podarí nájsť konkrétnu položku, tak ju zmaže a nakoniec sa aktualizuje vstupný XML súbor.
- **addVehicle**: Funkcia na pridanie stroja. Zistí sa, či ide o traktor alebo kombajn a na základe toho sa vytvorí nový element. K tomuto elementu sú vytvorené jednotlivé atribúty - meno, typ stroja, počet konských síl, rýchlosť a polomer zatáčania. Tieto vlastnosti sú priradené k elementu a ten je pridaný do XML stromu a následne je aktualizovaný aj celý súbor.
- **addEquipment**: Funkcia pracuje rovnako ako predchádzajúca, len vykonáva pridanie príslušenstva.
- **addHeader**: Opäť obdobná funkcia ako predchádzajúce dve, len sa pridávajú kombajnové lišty.
- **getVehicle**: Získanie mena, počtu konských síl, rýchlosti a polomeru zatáčania pre zadaný stroj - traktor alebo kombajn.
- **editVehicle**: Pre zadaný traktor alebo kombajn sa aktualizujú údaje. Najskôr sa na základe pôvodného mena stroja tento stroj vyhľadá a následne sa aktualizujú jeho atribúty. Ak je to kombajn, musia sa dodatočne prejsť všetky kombajnové lišty a ak sa daný kombajn nachádza medzi kompatibilnými, je aktualizovaný jeho názov. Nakoniec je aktuálna verzia zapísaná do súboru.

- **getEquipment**: Funkcia vracia atribúty zadaného príslušenstva. Na základe jeho typu sa vracajú špecifické dáta - pri kombajnovej lište sú to typ, názov, kompatibilné kombajny, maximálna pracovná rýchlosť a šírka záberu, pri zvyšnom príslušenstve sú to typ, meno, minimálny počet konských síl traktora, rýchlosť a šírka záberu.
- **editEquipment**: Funkcia na úpravu príslušenstva. Funguje podobne ako úprava strojov, na základe typu stroja a jeho názvu sa príslušenstvo vyhľadá a upravia sa jeho atribúty.
- **editHeader**: Opäť obdobná funkcia ako predchádzajúca, len pre kombajnové lišty.
- **getCompatibleHarvesters**: Funkcia, ktorá pre zadaný názov kombajnovej lišty vráti jej kompatibilné a nekompatibilné kombajny. Najskôr sa lišta vyhľadá a uložia sa kompatibilné kombajny. Následne sa prechádzajú kombajny a tie, ktoré sa nenachádzajú v zozname kompatibilných sa pridajú do zoznamu nekompatibilných.
- **deleteCompatible**: Funkcia na odstránenie kombajnu zo zoznamu kompatibilných kombajnov pre danú lištu.
- **addCompatible**: Funkcia na pridanie kombajnu do zoznamu kompatibilných kombajnov pre danú lištu.

Tieto funkcie zabezpečujú všetky operácie, ktoré by bolo potrebné vykonávať so strojmi a ich príslušenstvom a to v prehľadnom a jednoduchom zobrazení, s ktorého ovládaním nebude mať problém žiaden užívateľ. Druhou a hlavnou časťou backendu je samotné plánovanie trasy. Na to slúži funkcia `path_planning`, ktorá je zavolaná z funkcie `generate` a sú jej predané parametre súradníc polí a prekážok v nich, GPS súradnice vjazdu na pole, vektor, v ktorom bude stroj po poli prechádzať, šírka záberu príslušenstva a polomer zatáčania stroja. Táto funkcia sa nachádza v súbore `planner.py`. Hneď na začiatku je zavolaná funkcia `planning` s rovnakými argumentmi. V nej sa na začiatku vypočíta hodnota premennej `rotates`, ktorá označuje, koľko prejazdov musí stroj s konkrétnym príslušenstvom vykonať na konci riadkov, aby prekryl nepokrytú plochu, ktorá vzniká pri otáčaní stroja na konci riadku. Následne sa zavolá funkcia `find_sweep_direction_and_start_position`, ktorá nájde dva najvzdialenejšie body poľa a na základe nich vypočíta najoptimálnejší uhol jazdy stroja tak, aby sa minimalizoval počet otočení na konci riadkov. Funkcia teda vracia výsledný vektor a počiatočnú pozíciu. Ďalej sa overí, či bol zadaný vjazd na pole spomínaný v časti o zdroji dát, konkrétne hodnota VSTUP v XML súbore s poľami. Ak bola hodnota zadaná, tak je prepísaná hodnota vrátená funkciou `find_sweep_direction_and_start_position`. Rovnako je skontrolované, či nebol zadaný vektor jazdy stroja v hodnote UHOL v XML súbore s poľami. Ak bola hodnota zadaná, je použitá táto, ak nie, je použitá hodnota vrátená funkciou `find_sweep_direction_and_start_position`.

V ďalšej časti je zavolaná funkcia `enlarge`, ktorá sa nachádza v súbore `shrink.py`. Slúži na zväčšenie prekážok na poli, konkrétne tých, ktoré sú príliš malé na to, aby boli detekované pri ďalšom spracovávaní a vytváraní mapy poľa. Sú to napríklad rôzne stĺpy elektrického vedenia, šachty, alebo rôzne iné prekážky. Keďže chceme zväčšiť len prekážky na poli, nie samotné pole, tak sú súradnice v prvom poli (zozname) len skopírované, upravované sú až tie ďalšie. Pre každú prekážku je vypočítaná jej výška a šírka. Ak je jedna z týchto hodnôt menšia ako šírka záberu príslušenstva, tak je prekážka rozšírená vo všetkých smeroch o hodnotu polovice pokrytia príslušenstva, teda ak ju rozšírime do každej strany, rozšíri ju to presne o hodnotu pokrytia. Pre prípad, že je prekážka väčšia, sa využije taký prístup, že sa

jednotlivé úsečky ktoré ohraničujú pole rozšíria paralelne smerom von o hodnotu polovice pokrytia príslušenstvom. Je to z toho dôvodu, aby sa pri plánovaní nestalo, že by stroj prechádzal po hranici týchto prekážok, alebo aj cez ne napríklad pri vykonávaní otáčania. Konkrétne je to implementované tak, že sa vždy vezmú dve po sebe idúce úsečky, pre obe sa zavolá funkcia `parallel`, ktorá vráti priamky paralelné na pôvodné úsečky vo vzdialenosti polovice pokrytia. Pre tieto priamky sa následne nájde ich priesečník pomocou funkcie `line_intersection`, ktorý slúži ako nový bod hranice poľa. Toto sa vykoná pre všetky priamky vyjadrujúce hranice prekážky a vznikne tak nová, rozšírená hranica prekážky. Na záver funkcie `enlarge` sa odstránia prebytočné body, ktoré sú veľmi blízko pri sebe. Výsledok tejto operácie je možné vidieť na obrázku 7.7. Nové súradnice sú vrátené späť do funkcie `planning`.

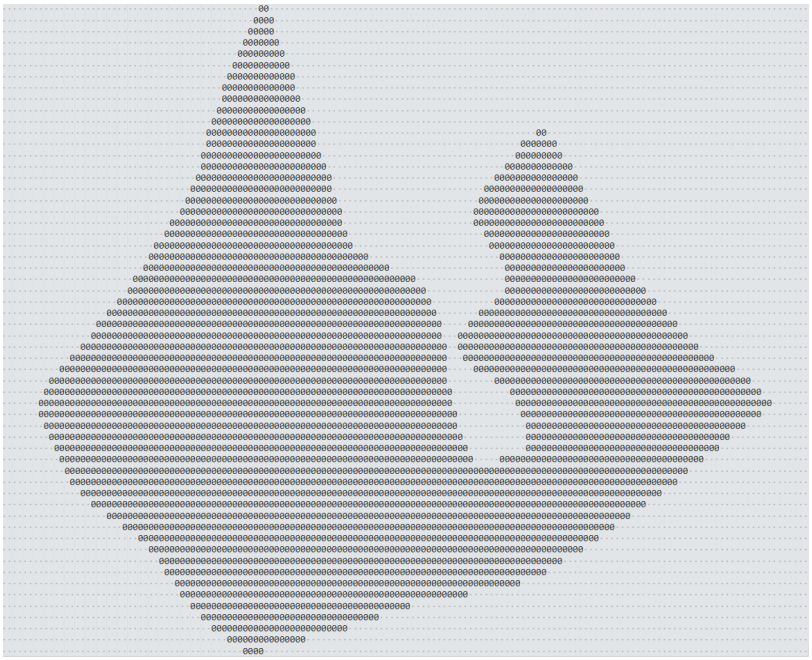


Obr. 7.7: Výsledok funkcie shrink - rozšírenie hraníc polí

Na vytvorenie reprezentácie poľa bola využitá knižnica `grid_map_lib.py`, ktorej autorom je Atsushi Sakai [31]. Je to len jedna časť celej sady algoritmov pre robotiku - open source softvérový projekt PythonRobotics. Ide o kolekciu robotických algoritmov implementovaných v programovacom jazyku Python. Projekt sa zameriava na autonómnu navigáciu s cieľom vysvetliť užívateľom základné princípy týchto algoritmov. Táto práca využíva ako základ pre tvorbu mapy jeden z nich: Coverage path planner, konkrétne Grid based sweep. Je využitá len funkcionalita pre vytvorenie mapy a podporné funkcie, ako napríklad zistenie, ktoré bunky boli preskúmané, alebo transformácia súradníc medzi jednotlivými formátmi.

Vo funkcii `planning` sa teda zavolá funkcia `convert_grid_coordinate`, ktorá transformuje súradnice do formátu, ktorý je využívaný v ďalšom kroku, pri funkcii `setup_grid_map`, ktorá využíva spomínanú knižnicu `grid_map_lib`. Najskôr sa vypočíta výška a šírka mapy a taktiež jej stred. Na základe týchto informácií je vytvorená 2D reprezentácia mapy poľa, ktorú je možné vidieť na obrázku 7.8

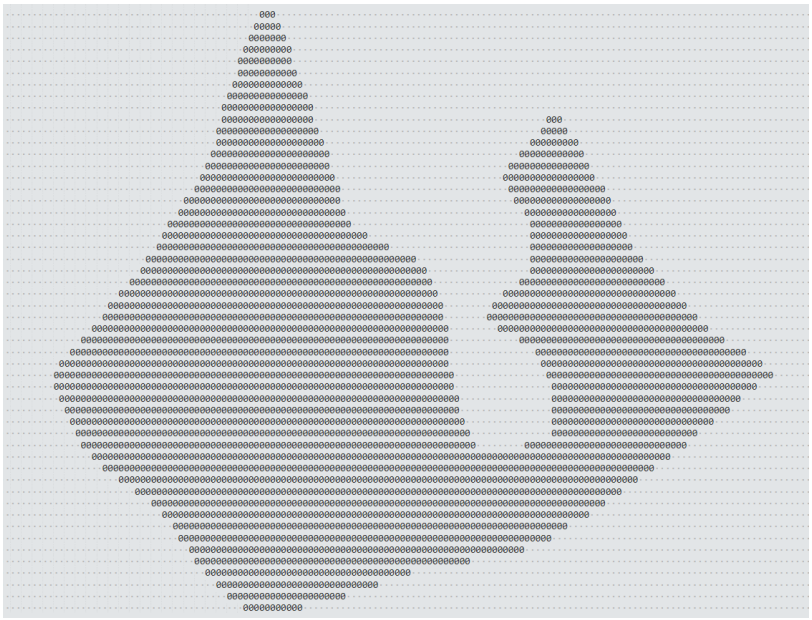
V ďalšej časti je vytvorená premenná `map_occupied`, v ktorej je uložená reprezentácia poľa pre účely vyhľadávania ciest v ňom. Pre danú mapu je zavolaná funkcia `shrink_map`, ktorej cieľom je zmenšiť vygenerovanú mapu v smere jazdy stroja o hodnotu `rotates`,



Obr. 7.8: 2D reprezentácia mapy poľa

ktorá bola vypočítaná na začiatku. Sú využité 4 cykly, každý pre jeden smer, dva v smere pohybu - tam aj späť a dva kolmo na smer pohybu - sprava a zľava. Ako bolo spomenuté, v smere jazdy je mapa zmenšená o hodnotu premennej `rotates` tak, aby mal stroj na konci riadku dostatok miesta na vykonanie otočky. V smere kolmom na smer pohybu je mapa zmenšená o 1 bunku, taktiež kvôli tomu, aby mal stroj dostatok miesta na vykonanie otočky. Nakoniec funkcia obsahuje cyklus, ktorý odstraňuje izolované bunky - také, ktoré vznikli predchádzajúcimi krokmi, keďže algoritmus, ktorý bude popisovaný v ďalšej časti pracuje s priamkami, nie s bodmi. Výsledok tejto funkcie je možné vidieť na obrázku 7.9

Nasleduje časť, ktorá sa venuje vytvoreniu trasy stroja okolo hraníc poľa. Najskôr sa zistí, či je to vôbec možné. V prípade veľmi členitého poľa s úzkymi miestami, alebo prekážkami, ktoré sa nachádzajú veľmi blízko hraníc poľa nie je možné vytvoriť bezpečnú trasu po obvode poľa. Je vypočítaná trasa po obvode poľa a to v bezpečnej vzdialenosti od jeho hranice pomocou funkcie `shrink`. Tá pracuje podobne ako `enlarge`, ale vytvorí trasu po obvode poľa a okolo prekážok v zadanej vzdialenosti. Táto trasa je následne overená pomocou funkcie `intersect`, ktorá každú úsečku tejto trasy porovná s úsečkami tvoriacimi obvod poľa a prekážok. Ak sa nejaké úsečky pretínajú, nie je možné vytvoriť bezpečnú trasu po obvode poľa. Ak je to možné, tak sa trasa vytvorí. Na pokrytie celej plochy, ktorá nebude pokrytá prechodom stroja po poli kvôli otočkám na konci riadkov je v niektorých prípadoch potrebné vykonať viacero okruhov po obvode poľa. Na to slúži už spomínaná premenná `rotates`. V cykle sa pridajú ďalšie okruhy okolo poľa a prekážok, podobne ako pri prvom okruhu. Tiež je tu overené, či sa nejaká trasa nepretína s hranicami poľa alebo prekážok. Po vytvorení okruhov okolo hraníc poľa a prekážok je potrebné tieto trasy spojiť, to sa vykoná v nasledujúcom cykle. Spájajú sa samostatne okruhy okolo poľa a samostatne okruhy okolo jednotlivých prekážok. Následne sa všetky tieto trasy spoja do jednej, ktorá bude pokrývať okolie všetkých hraníc. Pomocou funkcie `find_block` sa nájdu bunky, ktoré odpovedajú GPS súradniciam bodov, ktoré reprezentujú koniec trasy pre jednu prekážku a

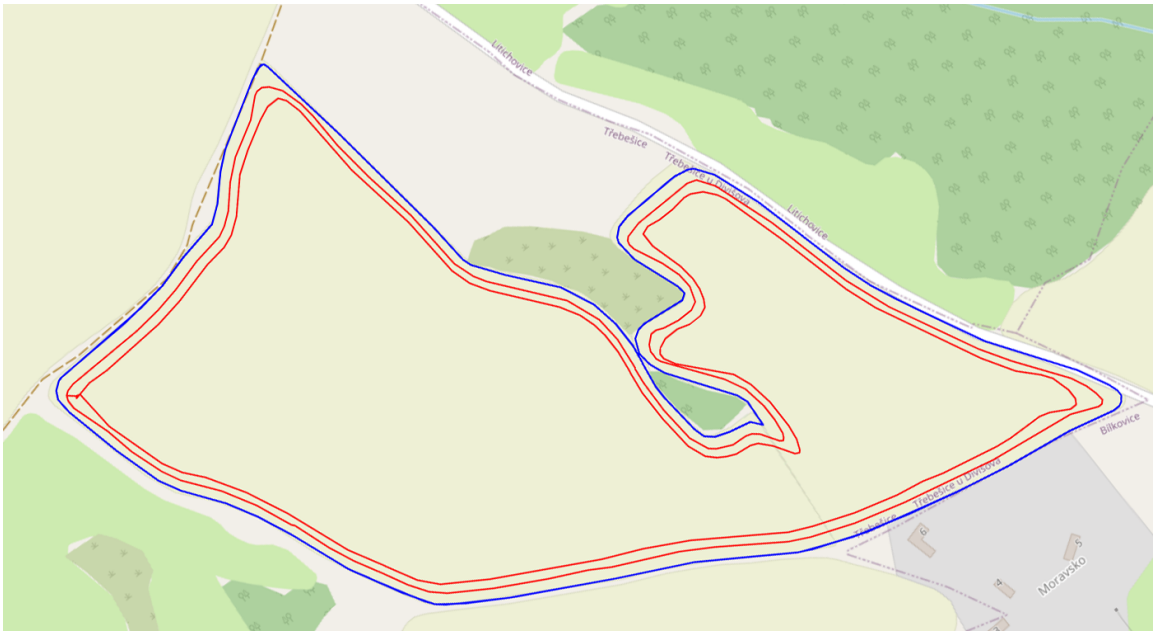


Obr. 7.9: 2D reprezentácia zmenšenej mapy poľa

začiatok trasy pre ďalšiu. Pomocou funkcie `bfs` sa nájde najkratšia trasa medzi dvoma prekážkami. Výsledkom je jedna trasa, ktorá pokrýva okolie hraníc poľa a všetkých prekážok na ňom, ako je možné vidieť na obrázku 7.10.

V ďalšej časti je vytvorený objekt `SweepSearcher` a sú inicializované jeho premenné. Tento objekt je požitý ako parameter v ďalšej funkcii - `sweep_path_search`. Ako parameter sa funkcii predáva aj vytvorená mapa `grid_map`, `map_occupied` a taktiež hodnota `rotates`. Na začiatku funkcie sa vyhľadá počiatočný bod prehľadávania a overí sa, či nie je obsadený. Ak by bol obsadený, tak sa nepodarilo vytvoriť trasu a je vrátené prázdne pole. Nasleduje cyklus, v ktorom sa vykonáva celé plánovanie. Základným prvkom tejto funkcie je funkcia `move_target_grid` s argumentmi aktuálneho indexu na ose x, indexu na ose y, mapou `grid_map` a zoznamom `ret_pos`, ktorý slúži na ukladanie pozícií, na ktoré sa ma stroj vrátiť po zmene trasy, napríklad keď sa vracia na miesta, ktoré neboli pokryté. `move_target_grid` obsahuje algoritmus na plánovanie trasy, ktorý využíva premenné `can_continue` - určuje, či je možné pokračovať v trase aj na ďalšom riadku a `go_back` - určuje, či je potrebné vrátiť sa niekam späť a `next_sweep_direction` - akým smerom sa bude stroj pohybovať keď dokončí terajší riadok. Celý algoritmus vyzerá nasledovne:

```
if (grid_map(n_x_index, n_y_index) is free):
  if (can_continue == False and
    grid_map(n_x_index, n_y_index + sweep_direction) is free):
    can_continue = True
  if (grid_map(n_x_index, n_y_index - sweep_direction) is free):
    if (next_sweep_direction == sweep_direction):
      next_sweep_direction = - next_sweep_direction
  return n_x_index, n_y_index, False, False
else:
  if (sweep_direction != next_sweep_direction):
    sweep_direction = next_sweep_direction
```



Obr. 7.10: Trasa v okolí hranice poľa

```
if not (go_back):
  ret_pos.append(c_x_index, c_y_index, self.moving_direction)
go_back = False
can_continue = False
if(grid_map(c_x_index, c_y_index + sweep_direction) is free):
  n_x_index = c_x_index
  n_y_index = c_y_index + sweep_direction
  while (grid_map(n_x_index + moving_direction, n_y_index) is free):
    n_x_index = n_x_index + moving_direction
  swap_moving_direction()
  if (n_x_index == c_x_index and n_y_index == c_y_index):
    return None, None, False, False
  return n_x_index, n_y_index, False, True
else:
  n_x_index = c_x_index
  n_y_index = c_y_index + sweep_direction
  while (grid_map(n_x_index - moving_direction, n_y_index) is free):
    n_x_index = n_x_index - self.moving_direction
    if (n_x_index < 0 or n_x_index >= grid_map.width):
      swap_moving_direction()
      n_x_index = c_x_index
      while (grid_map(n_x_index - moving_direction, n_y_index) is free):
        n_x_index = n_x_index - self.moving_direction
        if (n_x_index < 0 or n_x_index >= grid_map.width):
          return None, None, False, True
    swap_moving_direction()
    if (n_x_index == c_x_index and n_y_index == c_y_index):
```

```
        return None, None, False, False
      return n_x_index, n_y_index, False, True
else:
  if (grid_map(c_x_index, c_y_index + sweep_direction) is free):
    can_continue = False
    n_x_index = c_x_index
    n_y_index = c_y_index + sweep_direction
    x = None
    y = None
    while (grid_map(n_x_index + moving_direction, n_y_index) is free):
      if (not go_back and
      grid_map(n_x_index, n_y_index - sweep_direction) is free):
        go_back = True
        x = n_x_index
        y = n_y_index - self.sweep_direction
      n_x_index = n_x_index + self.moving_direction
    if (go_back):
      ret_pos.append(n_x_index, n_y_index, -moving_direction)
    return x, y, False, True
  else:
    if not (can_continue):
      if ret_pos == []:
        return None, None, False, False
      tmp = ret_pos[-1]
      del ret_pos[-1]
      sweep_direction = -sweep_direction
      next_sweep_direction = self.sweep_direction
      moving_direction = tmp[2]
      return tmp[0], tmp[1], True, False
    self.can_continue = False
    n_x_index = c_x_index
    n_y_index = c_y_index + self.sweep_direction
    while (grid_map(n_x_index - moving_direction, n_y_index) is free):
      n_x_index = n_x_index - self.moving_direction
    self.swap_moving_direction()
    if n_x_index == c_x_index and n_y_index == c_y_index:
      return None, None, False, False
    return n_x_index, n_y_index, False, True
```

Vysvetlenie algoritmu: najskôr sa overí, ča sa dá pokračovať priamo vpred na aktuálnom riadku. Ak áno, pozrie sa vedľa seba v smere prechodu poľom, či sa dá pokračovať ďalej, ak áno, nastaví hodnotu `can_contunie` na `True`. Následne overí, či sa dá ísť vedľa seba proti prechodu poľom, to znamená, že overí, či sa nejaká časť poľa nevynechala. To môže nastať, ak má pole nepravidelný tvar a na niektorom mieste sa rozširuje. V takom prípade sa prehodí hodnota premennej `next_sweep_direction`, ak ešte nebola prehodená. Nakoniec sa vráti nová pozícia. V prípade, že sa ďalej nedá pokračovať v jednom riadku - to nastáva, ak je stroj na konci riadku, tak sa overí, či je hodnota `next_sweep_direction` iná, ako hodnota `sweep_direction`.

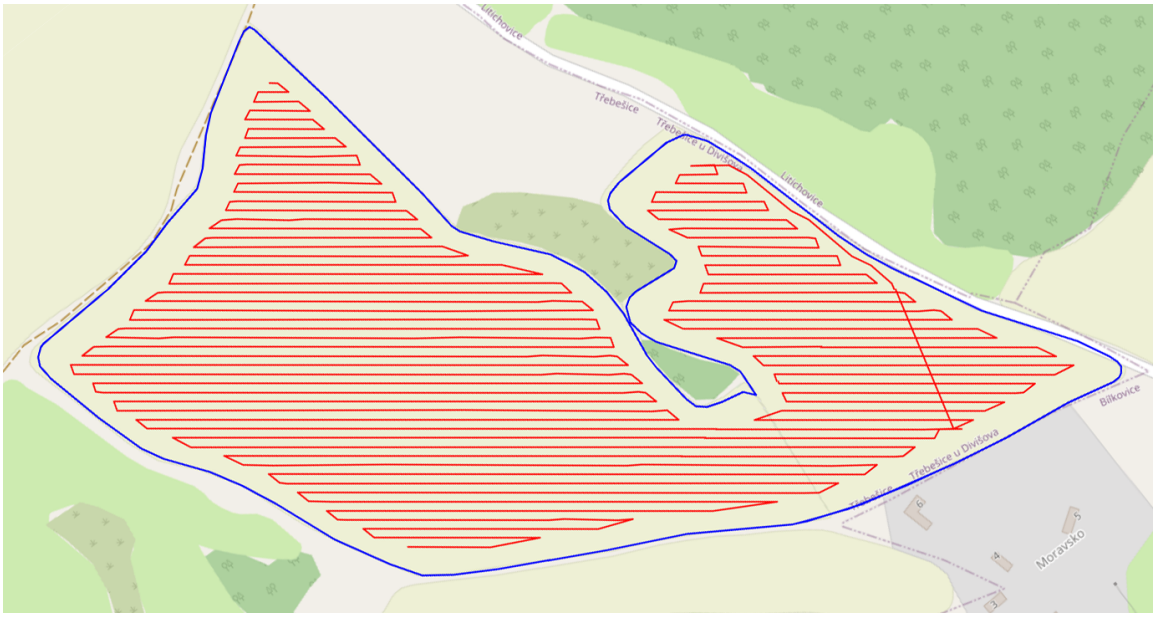
Ak áno, znamená to, že je potrebné pokryť plochu, ktorá bola vynechaná. Hodnota `sweep_direction` sa nastaví na hodnotu `next_sweep_direction`, ak bola nastavená hodnota `go_back` na `False`, je do zoznamu `ret_pos` pridaná pozícia, na ktorú sa dá vrátiť a premenné `go_back` a `can_continie` sú nastavené na `False`. Následne sa pozrieme na vedľajší riadok proti smeru prechádzania poľa a zistíme, či je vedľajšia pozícia voľná. Ak je, v cykle nájdeme poslednú voľnú pozíciu v danom riadku a vrátime ju. Ak však vedľajšia pozícia nie je voľná, musíme nájsť prvú voľnú pozíciu na vedľajšom riadku proti aktuálnemu smeru pohybu stroja. Ak sa nepodarí nájsť voľnú pozíciu, vráti sa `None`, v prípade úspechu sa vráti nová pozícia.

Ak je hodnota `next_sweep_direction` rovnaká, ako hodnota `sweep_direction`, pozrieme sa, či je vedľajšia pozícia voľná. Ak áno, nastavíme `can_continue` na `False` a pokračujeme až na koniec riadku. Medzitým však pozeráme aj na riadok, kde sme práve boli, či sa ďalej nenachádza nejaká voľná pozícia, ak áno, nastavíme hodnotu `go_back` na `True` a aktualizujeme indexy. Ak sa tak udialo, vrátime tieto nové pozície na aktuálnom riadku. Inak vrátime poslednú pozíciu na novom riadku. Ak je hodnota `next_sweep_direction` rovnaká, ako hodnota `sweep_direction`, ale vedľajšia pozícia nie je voľná, pozrieme sa na hodnotu premennej `can_continue`. Ak je jej hodnota `False`, skontrolujeme obsah zoznamu `ret_pos`, ak je prázdny, vrátime `None`, inak vrátime poslednú pozíciu v zozname a zo zoznamu ju odstránime. Ak je hodnota `can_continue` `True`, nastavíme ju na `False` a nájdeme prvú voľnú pozíciu na vedľajšom riadku smerom späť - proti aktuálnemu smeru stroja. Ak sme novú pozíciu nenašli, vrátime `None`, inak vrátime novú pozíciu.

Funkcia vracia štvoricu: `n_x_index` - nasledujúci index v smere x, `n_y_index` - nasledujúci index v smere y, hodnotu `returning`, ktorá udáva, či sa stroj vracia na miesto, ktoré vynechal a hodnotu `turning` udávajúcu, či sa v danom bode bude zatáčať.

Ak je hodnota `n_x_index` alebo hodnota `n_y_index` `None` a zároveň je hodnota `returning` `False`, vyhľadá sa najbližšia voľná pozícia pomocou funkcie `find_closest`. Využíva prehľadávanie do šírky a používa na to frontu, do ktorej sa ukladajú nasledujúce pozície, ktoré sú v medziach poľa a neboli ešte preskúmané týmto algoritmom. Preskúmané pozície sa ukladajú do zoznamu navštívených pozícií. Funkcia vracia najbližšiu voľnú pozíciu, alebo `None`, ak žiadnu voľnú pozíciu nenájde. V takom prípade sa prehľadávanie ukončí. Ak našiel novú pozíciu, je k nej nájdená najbližšia cesta z aktuálnej pozície a to pomocou funkcie `bfs`, ktorá funguje podobne ako funkcia `find_closest`, ale pri prehľadávaní kontroluje aj to, či nevyšiel z poľa. Na to sa využíva predtým spomínané pole `map_occupied`, čiže 2D reprezentácia poľa a jeho voľných pozícií. Funkcia vracia najkratšiu cestu z aktuálnej pozície na novú pozíciu. Ak vráti `None`, prehľadávanie je ukončené. V opačnom prípade sa na získanú trasu zavolá funkcia `remove_points`, ktorá odstráni prebytočné body a zaistí, že bude trasa plynulejšia a nebude obsahovať príliš ostré zákruty. Výsledná trasa je transformovaná na iný formát súradníc a je pridaná k výslednej trase.

Ak je hodnota `returning` `True`, čiže sa stroj vracia späť na pôvodné miesto po pokrytí plochy, ktorú predtým vynechal, opäť sa overí, či existuje nejaké nepokryté miesto na mape, ak áno, pomocou funkcie `bfs` sa nájde cesta k najbližšiemu nepokrytému miestu, opäť sa odstránia prebytočné body výsledná trasa je transformovaná na iné súradnice a pridaná k výslednej ceste. Nakoniec je pozícia označená ako preskúmaná a cyklus sa opakuje, kým sa nepokryje celá mapa. Funkcia `sweep_path_search` nakoniec vráti trasu, ktorá pokrýva všetky voľné pozície poľa. Táto trasa je transformovaná na GPS súradnice. Keď sme už získali trasu pokrývajúcu celé pole aj trasu pokrývajúcu okolie hraníc poľa a prekážok, je potrebné tieto trasy spojiť do jednej. Tu sa rozlišuje, aký stroj vykonáva prácu. Ak je to kombajn, tak ten musí najskôr pozbierať úrodu na hraniciach poľa a prekážok, aby sa tam



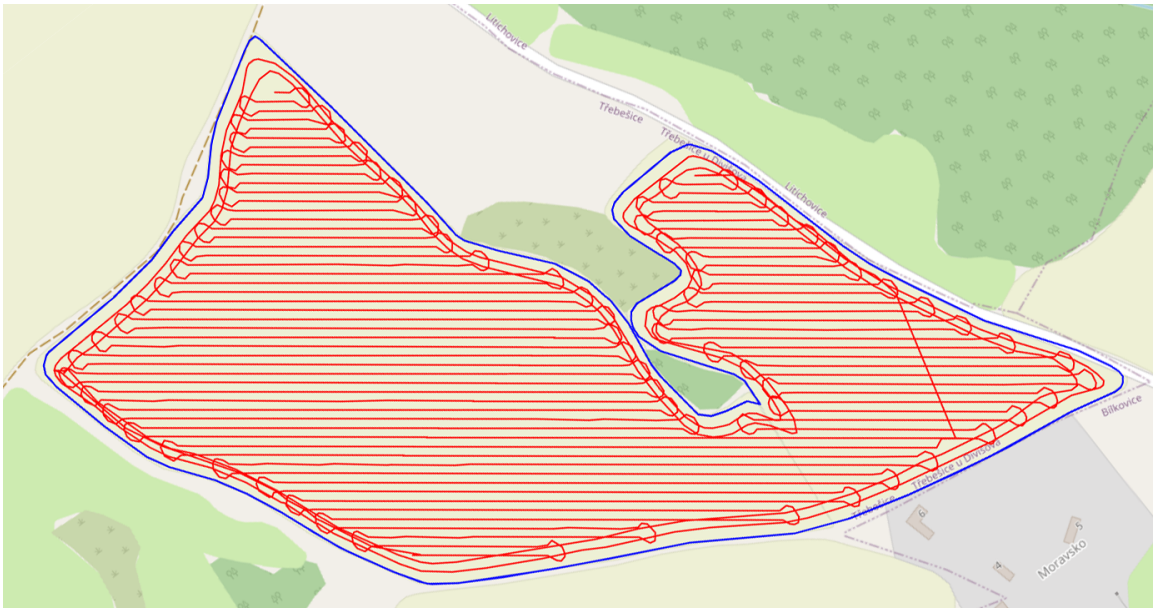
Obr. 7.11: Trasa

mohol pri jazde cez pole otáčať a nepoškodil tak úrodu. Výsledná trasa sa vytvorí spojením trasy pokrývajúcej okolie hraníc poľa a prekážok, trasy spájajúcu túto trasu s trasou pokrývajúcou pole a trasou pokrývajúcou pole. Pri ostatných strojoch je lepšie využiť opačný postup - najskôr pokryť pole a nakoniec okolie hraníc poľa a prekážok, napríklad pri činnosti ako orba alebo sejba. Táto trasa je vrátená funkcii `path_planning`. Ak nebolo možné vytvoriť trasu okolo hraníc poľa a prekážok, je vrátená len trasa pokrývajúca pole.

Na trasu je aplikovaný algoritmus rdp - Ramer-Douglas-Peucker algoritmus, ktorý slúži na odstránenie bodov, ktoré s istou odchylkou ležia na jednej priamke. Algoritmus vráti zoznam hodnôt `True`/`False`, ktoré označujú, ktoré body môžu byť odstránené. Na základe tejto masky sú odstránené niektoré súradnice a taktiež je zredukované pole `turns`, ktoré udáva, v ktorom bode sa bude vykonávať odbočka na ďalší riadok. Trasa je zobrazená na obrázku 7.11

Na obrázku 7.11 je možné vidieť, že trasa obsahuje ostré zákruty na konci riadkov, preto je na trasu aplikovaná funkcia `add_turns` s argumentmi súradníc v ose x, y, vektor, v ktorom sa stroj pohyboval, práve získané pole s identifikáciou otočiek na konci riadkov, šírku záberu príslušenstva a polomer zatáčania stroja. Funkcia obsahuje cyklus, ktorý sa opakuje dovtedy, kým nie je prejdená celá trasa. To, či je potrebné pridať otočku je určené hodnotou spomínanej premennej `turns` na konkrétnom indexe. Ak sa nemá pridávať otočka, jednoducho sa zvýši index. V opačnom prípade sa vypočíta uhol medzi jednotlivými bodmi a taktiež základný uhol priamky určujúcej trasu a stred bodov, medzi ktorými sa má zatáčať.

Ak je uhol medzi bodmi zatáčania iný ako takmer 90, resp. 270 stupňov, musí byť pridaný nový bod tak, aby bolo možné správne pridať ďalšie body, ktoré budú tvoriť otočku. Pridaním nového bodu sa docieli, že bude uhol medzi nimi presne 90, resp. 270 stupňov. Tento bod sa pridá tak, že sa na základe uhlu vykoná konkrétna vetva programu. V oboch vetvách sa vypočíta vzdialenosť a uhol nového bodu od konkrétneho bodu, ktorý sa líši v týchto dvoch vetvách programu. Tento bod je pridaný do zoznamu súradníc a je vypočítaná nová hodnota `midpoint` - stredového bodu.



Obr. 7.12: Výsledok plánovania trasy

Po pridaní nového bodu sa opäť vypočíta nový uhol, ktorý body zvierajú. V ďalšej časti sa rozlišuje, aká otočka sa bude vykonávať, či tá jednoduchšia, alebo zložitejšia. Pre jednoduchšiu otočku platí, že dvojnásobná hodnota polomeru zatáčania stroja musí byť menšia alebo rovná ako šírka záberu príslušenstva, s ktorým stroj pracuje. Nové body sa vypočítajú na základe šírky záberu príslušenstva, polohy stredového bodu (`midpoint`), polohy okrajových bodov a ich uhlu. Po vytvorení týchto bodov sú pridané do celkovej trasy na správny index. Pri zložitejšej otočke, keď je dvojnásobná hodnota polomeru zatáčania väčšia ako šírka záberu príslušenstva, musí sa vypočítať bod, okolo ktorého sa bude vytvárať kružnica, po ktorej sa bude stroj otáčať. Tento bod sa vypočíta na základe stredového bodu (`midpoint`), polomeru otáčania stroja a uhlu, v ktorom sa body nachádzajú. Je taktiež vypočítaný uhol, na základe ktorého sa určí počet bodov, ktoré budú tvoriť kružnicu. Samotná tvorba bodov je podobná ako v predchádzajúcom prípade, okolo stredového bodu sa vytvárajú nové body s určitým rozostupom. Tieto body sú nakoniec pridané do celkovej trasy. Algoritmus pridávania otočiek končí, keď sa pokryje celá trasa. Výsledná trasa je funkciou vrátená do funkcie `path_planning` a z nej je trasa vrátená do funkcie `generate`, kde je následne vykreslená pomocou knižnice `mplleaflet`. Výsledok je možné vidieť na obrázku 7.12

Spustenie programu

Celý program sa nachádza v priečinku `src`. Pre jeho spustenie je potrebné mať nainštalovaný Python 3, ideálne 3.10.10. Vo Windows príkazovom riadku zadáme príkaz `python -m venv 'nazov_venv'` a vytvoríme tak virtuálne prostredie. Následne ho aktivujeme príkazom `'nazov_venv'\Scripts\activate`. Pre fungovanie programu je potrebné nainštalovať všetky závislosti, to vykonáme zadaním príkazu `pip install -r requirements.txt`. Nainštalovaná knižnica `mplleaflet` však obsahuje menšiu chybu, takže je tam potrebné nahradiť jeden súbor. Skopírujeme teda súbor `utils.py` z adresára `src`

a vložíme ho do `'nazov_venv'\Lib\site-packages\mplleaflet\mplexporter`. Program následne spustíme príkazom `python app.py` v adresári `src`.

Obmedzenia a testovanie

Keďže je program napísaný v jazyku Python, nemal by byť problém s jeho spustením na rôznych platformách, je však potrebné mať nainštalované Python a knižnice v potrebných verziách. Primárne bol však tento program vyvíjaný pre operačný systém Windows 10 a na ňom bol aj testovaný. Testovanie programu bolo vykonané na súbore s poľami, ktoré je súčasťou odovzdaných súborov. Boli z neho odstránené polia, ktoré nevyhovujú navrhnutému algoritmu - polia boli príliš malé (pár metrov na šírku/dĺžku), alebo boli veľmi členité, čiže navrhnutý algoritmus pre ne nebol vhodný. Algoritmus je vhodné používať pre dostatočne veľké polia, ktoré nie sú príliš členité, alebo neobsahujú úzke miesta, kde nie je možné vygenerovať trasu. Taktiež nie je vhodné kombinovať veľké stroje s malým príslušenstvom, ktoré pre ne nie je určené. To isté platí aj pre používanie príliš veľkého príslušenstva na malých poliach. Čo sa týka menežovania strojov a príslušenstva k nim, tam dopadlo testovanie podľa očakávania, je možné stroje pridávať, upravovať a aj odstraňovať. Výsledky programu je možné vidieť v prílohe A.

Kapitola 8

Záver

Cieľom tejto práce bolo naštudovať algoritmy pre plánovanie cesty, konkrétne sa zamerať na algoritmy používané pre kompletné pokrytie priestoru (tzv. coverage path planning). Bolo potrebné sa zamerať na plánovanie cesty s ohľadom na vstupné a výstupné body, brať do úvahy reálne obmedzenia stroja a taktiež brať do úvahy sklon poľa. V neposlednom rade bolo potrebné navrhnúť a implementovať program, ktorý umožní nahrať tvar oblasti a vytvoriť pre ňu plán cesty s ohľadom na obmedzenia reálneho stroja, ako je šírka záberu, polomer otáčania, alebo jeho rýchlosť. Začiatok tejto práce je zameraný na zadefinovanie základných pojmov, za ktorými nasleduje zhrnutie histórie v tejto oblasti a najväčších výrobcov, ktorí sa snažia o vývoj autonómnych poľnohospodárskych strojov. V ďalšej časti boli zhrnuté algoritmy typu point-to-point. V chronologickom poradí boli zhrnuté najzaujímavejšie princípy a algoritmy tohto typu plánovania cesty. Následne sa riešia algoritmy typu coverage routing. Tu boli taktiež vymenované základné princípy týchto algoritmov, ako aj rôzne spôsoby ich implementácie. Po tejto časti nasledoval popis reálneho využitia algoritmov typu coverage routing v oblasti poľnohospodárstva, kde sa brali do úvahy vlastnosti stroja, ako je šírka záberu, alebo spotreba, ktorá sa odvíjala od správneho naplánovania trasy v závislosti od sklonu poľa.

Ďalšia časť práce sa venovala návrhu programu. Na základe požiadavok vyplývajúcich zo zadania práce bol navrhnutý program, ktorý dokáže pracovať s rôznymi poľami a strojmi, ktoré je možné rôzne upravovať. V návrhu sme sa taktiež venovali jednotlivým technológiám, ktoré budú využité pri implementácií programu a taktiež zdroje dát, či už pre polia, alebo pre stroje.

Jednou z posledných častí bola samotná implementácia programu. Implementácia bola rozdelená do dvoch častí - implementácia frontendu a backendu. V implementácii frontendu sme sa venovali tvorbe samotného zobrazenia aplikácie a rozmiestneniu jednotlivých prvkov. Implementácia backendu bola rozdelená do dvoch častí - menežovanie polí a strojov a samotné plánovanie trasy a jej zobrazenie. Aplikácia umožňuje zadaný súboru so strojmi načítať, stroje pridávať, upravovať a odstraňovať. Pre samotné plánovanie trasy bola využitá metóda bunkovej dekompozície a bol vytvorený algoritmus na pokrytie daného priestoru metódou coverage routing, ako to vyplývalo zo zadania tejto práce. V poslednej časti bolo vykonané testovanie tohto programu a boli predstavené výsledky práce. Z výsledkov práce vyplýva, že jej cieľ bol splnený. V budúcnosti je možné samotný algoritmus plánovania trasy ešte zefektívniť a taktiež ho rozšíriť o možnosť plánovania trasy medzi viacerými poľami.

Literatúra

[1] *Autonomous Tractor Corporation*. Dostupné z: https://agfundernews.com/wp-content/uploads/2015/09/326_1autonomous.jpg.

[2] *Case IH*. Dostupné z: https://uncrate.com/assets_c/2016/09/case-magnum-auto-tractor-1-thumb-960xauto-65848.jpg.

[3] *Eagri*. Dostupné z: https://eagri.cz/public/web/mze/.

[4] *Fendt*. Dostupné z: http://zlatex.com/public/files/attached_images/1467/file/bg-BG/gc_maisstoppel_archiv_kl.jpg.

[5] *John Deere*. Dostupné z: https://static.country-guide.ca/wp-content/uploads/2021/06/17171110/Deere.jpeg.

[6] *Rapidly exploring random tree*. Dostupné z: https://mappingignorance.org/app/uploads/2013/02/Blanco1.png.

[7] *Algoritmus*. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: https://sk.wikipedia.org/wiki/Algoritmus.

[8] *Algoritmus*. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: https://cs.wikipedia.org/wiki/Algoritmus.

[9] *Driverless tractor*. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: https://en.wikipedia.org/wiki/Driverless_tractor.

[10] Acar, E. U., Choset, H., Zhang, Y. a Schervish, M. Path Planning for Robotic Demining: Robust Sensor-Based Coverage of Unstructured Environments and Probabilistic Methods. *The International Journal of Robotics Research*. 2003, zv. 22, 7-8, s. 441–466. DOI: 10.1177/02783649030227002. Dostupné z: https://doi.org/10.1177/02783649030227002.

[11] Cao, Z., Huang, Y. a Hall, E. Region filling operations with random obstacle avoidance for mobile robots. *Journal of Robotic Systems*. Apríl 1988, zv. 5, s. 87 – 102. DOI: 10.1002/rob.4620050202.

[12] Castillo, O., Trujillo, L. a Melin, P. Multiple Objective Genetic Algorithms for Path-planning Optimization in Autonomous Mobile Robots. *Soft Computing*. 2007, zv. 2007, č. 11, s. 269–279. DOI: 10.1007/s00500-006-0068-4.

[13] Chakraborty, S., Elangovan, D., Govindarajan, P. L., ELnaggar, M. F., Alrashed, M. M. et al. A Comprehensive Review of Path Planning for Agricultural

Ground Robots. *Sustainability*. 2022, zv. 14, č. 15. DOI: 10.3390/su14159156. ISSN 2071-1050. Dostupné z: https://www.mdpi.com/2071-1050/14/15/9156.

[14] Choset, H. Coverage for robotics - A survey of recent results. *Ann. Math. Artif. Intell.* Október 2001, zv. 31, s. 113–126. DOI: 10.1023/A:1016639210559.

[15] Costa, M. M. a Silva, M. F. A Survey on Path Planning Algorithms for Mobile Robots. In: *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. 2019, s. 1–7. DOI: 10.1109/ICARSC.2019.8733623.

[16] Dorigo, M., Maniezzo, V. a Colorni, A. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*. 1996, zv. 26, č. 1, s. 29–41. DOI: 10.1109/3477.484436.

[17] Fernandes, E., Costa, P., Lima, J. a Veiga, G. Towards an orientation enhanced astar algorithm for robotic navigation. In: *2015 IEEE International Conference on Industrial Technology (ICIT)*. 2015, s. 3320–3325. DOI: 10.1109/ICIT.2015.7125590.

[18] Galceran, E. a Carreras, M. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*. 2013, zv. 61, č. 12, s. 1258–1276. DOI: https://doi.org/10.1016/j.robot.2013.09.004. ISSN 0921-8890. Dostupné z: https://www.sciencedirect.com/science/article/pii/S092188901300167X.

[19] Goto, T., Kosaka, T. a Noborio, H. On the heuristics of A* or A algorithm in ITS and robot path-planning. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. 2003, sv. 2, s. 1159–1166 vol.2. DOI: 10.1109/IROS.2003.1248802.

[20] Hameed, I., la Cour-Harbo, A. a Osen, O. Side-to-side 3D coverage path planning approach for agricultural robots to minimize skip/overlap areas between swaths. *Robotics and Autonomous Systems*. 2016, zv. 76, s. 36–45. DOI: https://doi.org/10.1016/j.robot.2015.11.009. ISSN 0921-8890. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0921889015002973.

[21] Hameed, I. Intelligent Coverage Path Planning for Agricultural Robots and Autonomous Machines on Three-Dimensional Terrain. *Journal of Intelligent & Robotic Systems*. Jún 2013, zv. 74. DOI: 10.1007/s10846-013-9834-6.

[22] Hert, S., Tiwari, S. a Lumelsky, V. J. A terrain-covering algorithm for an AUV. *Autonomous Robots*. 1996, zv. 3, s. 91–119.

[23] Huang, W. Optimal line-sweep-based decompositions for coverage algorithms. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. 2001, sv. 1, s. 27–32 vol.1. DOI: 10.1109/ROBOT.2001.932525.

[24] Karaman, S. a Frazzoli, E. Sampling-based Algorithms for Optimal Motion Planning. *CoRR*. 2011, abs/1105.1186. Dostupné z: http://arxiv.org/abs/1105.1186.

[25] Knuth, D. E. *Umění programování*. Vyd. 1. Brno: Computer Press, 2008. ISBN 978-80-251-2025-5.

[26] LaValle, S. M. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*. 1998.

[27] Lozano Perez. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*. 1983, C-32, č. 2, s. 108–120. DOI: 10.1109/TC.1983.1676196.

[28] Lumelsky, V., Mukhopadhyay, S. a Sun, K. Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation*. 1990, zv. 6, č. 4, s. 462–472. DOI: 10.1109/70.59357.

[29] Oksanen, T. a Visala, A. Coverage path planning algorithms for agricultural field machines. *Journal of Field Robotics*. 2009, zv. 26, č. 8, s. 651–668. DOI: https://doi.org/10.1002/rob.20300. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20300.

[30] Pivtoraiko, M., Knepper, R. A. a Kelly, A. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*. 2009, zv. 26, č. 3, s. 308–333. DOI: https://doi.org/10.1002/rob.20285. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20285.

[31] Sakai, A., Ingram, D., Dinius, J., Chawla, K., Raffin, A. et al. *PythonRobotics: a Python code collection of robotics algorithms*. 2018.

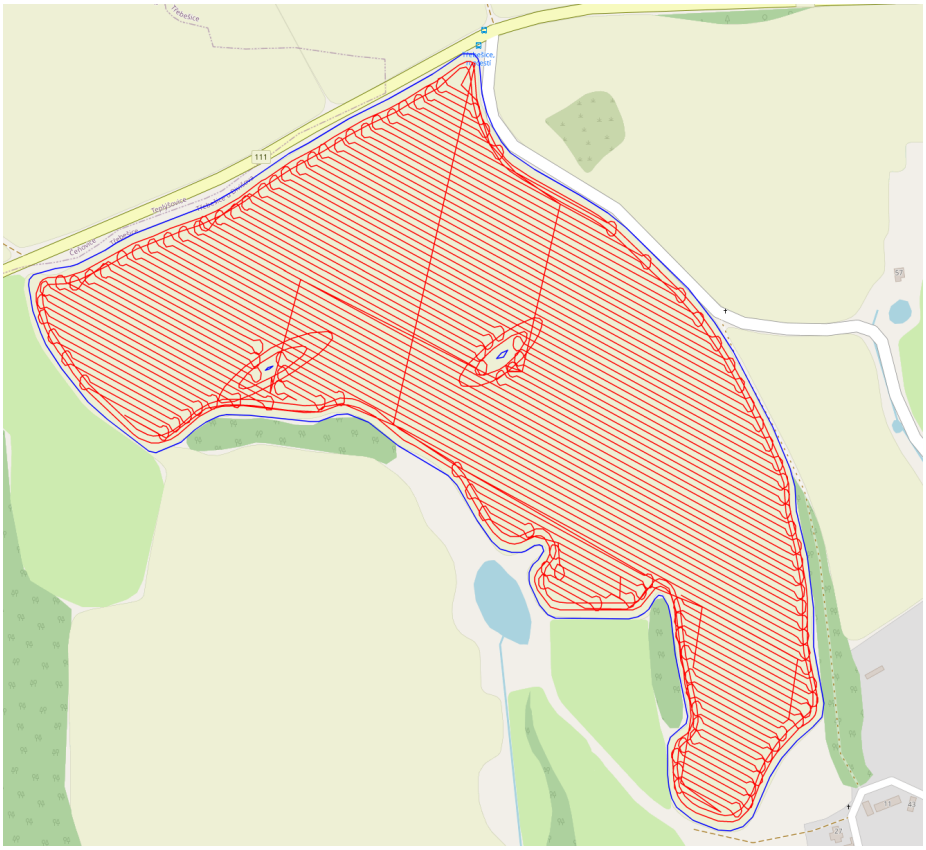
[32] Schäfle, T. R., Mohamed, S., Uchiyama, N. a Sawodny, O. Coverage path planning for mobile robots using genetic algorithm with energy optimization. In: *2016 International Electronics Symposium (IES)*. 2016, s. 99–104. DOI: 10.1109/ELECSYM.2016.7860983.

[33] Yang, S. a Luo, C. A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*. 2004, zv. 34, č. 1, s. 718–724. DOI: 10.1109/TSMCB.2003.811769.

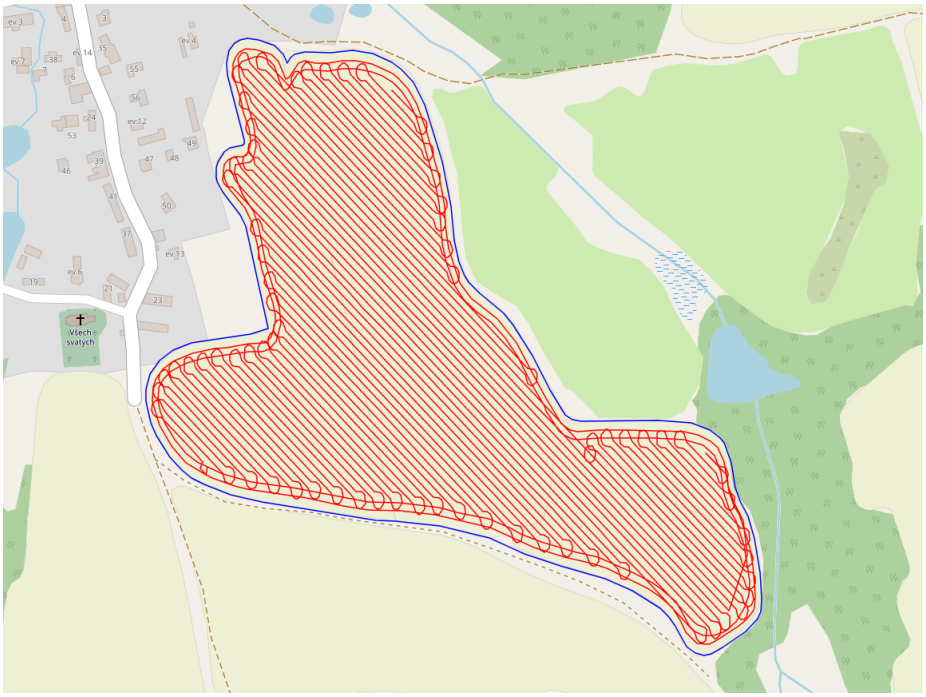
[34] Zelinsky, A., Jarvis, R. A., Byrne, J. a Yuta, S. Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. In:. 2007.

Príloha A

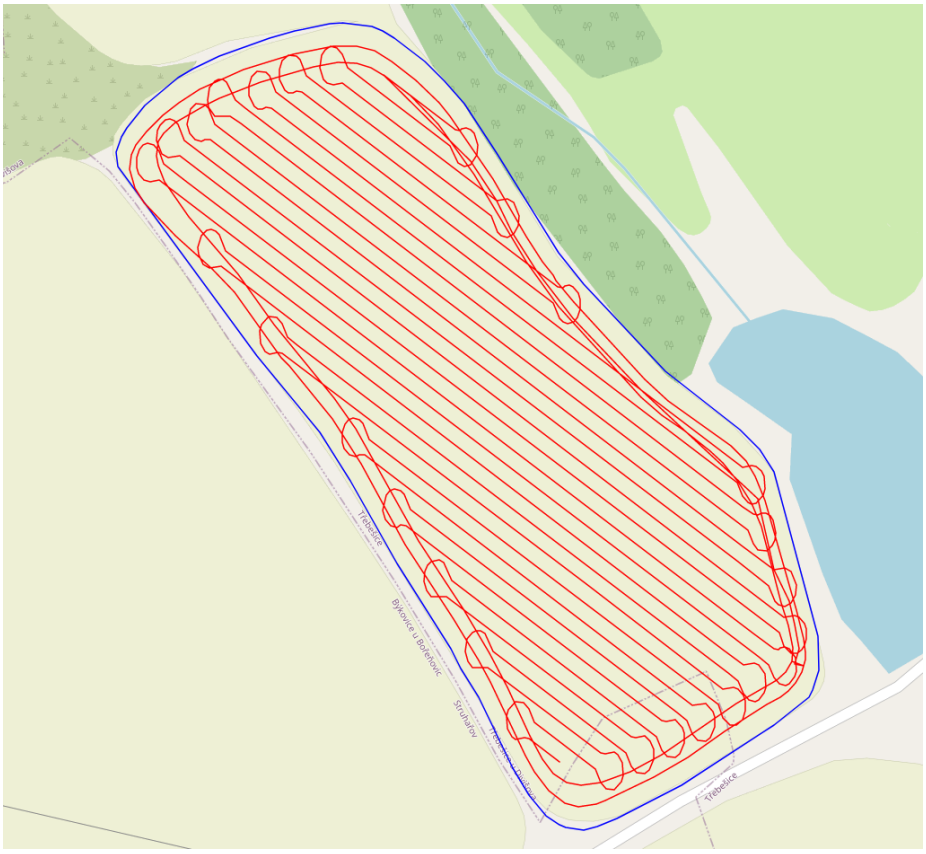
Výsledky programu



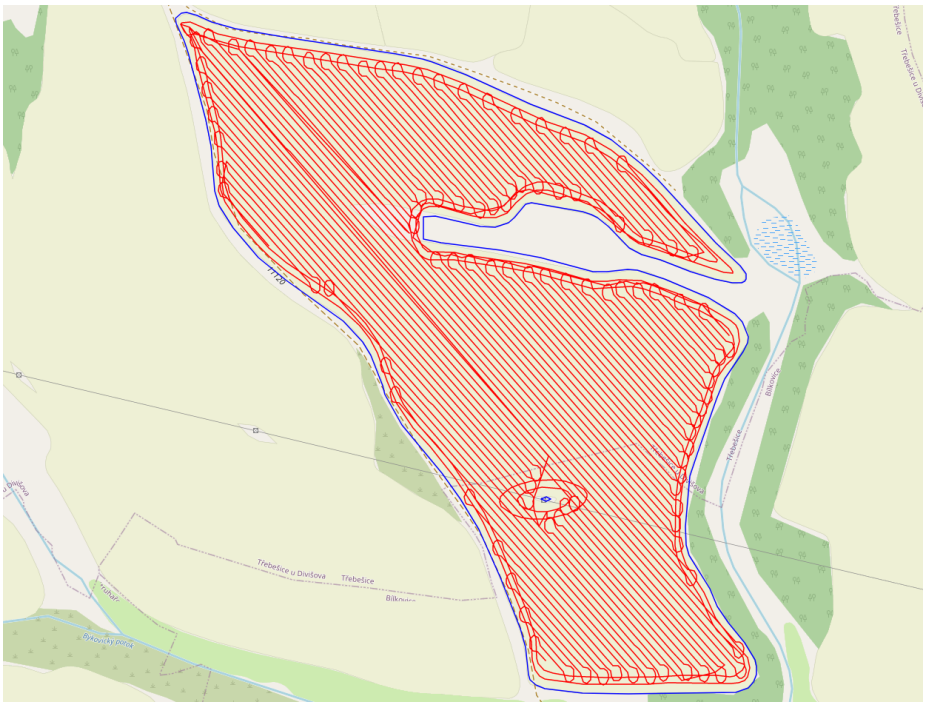
Obr. A.1: Výsledok programu



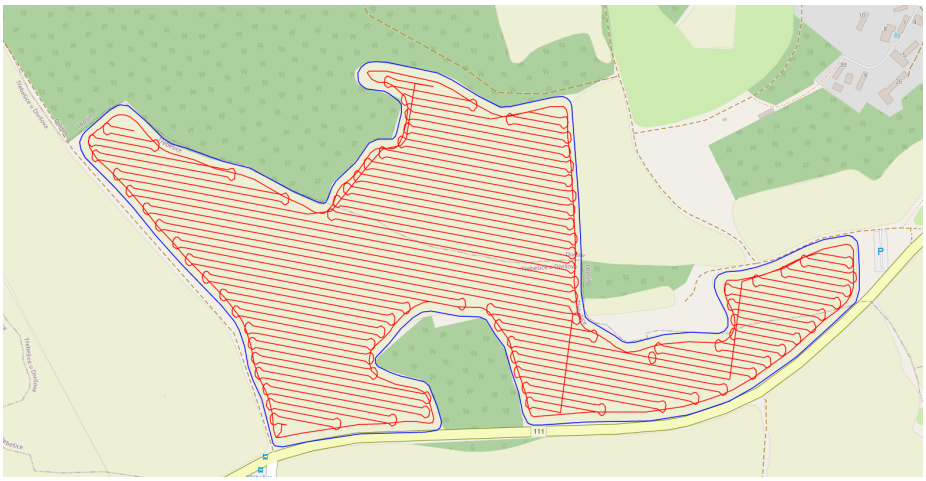
Obr. A.2: Výsledok programu



Obr. A.3: Výsledok programu



Obr. A.4: Výsledok programu



Obr. A.5: Výsledok programu

Príloha B

CD

Obsah CD:

- **src**: priečinok obsahuje zdrojové súbory,
- **README.txt**: súbor obsahuje návod na spustenie programu,
- **technicka_sprava**: priečinok obsahuje zdrojové súbory technickej správy,
- **technicka_sprava.pdf**: PDF súbor technickej správy,
- **technicka_sprava_tlac.pdf**: PDF súbor technickej správy určenej na tlač.