ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STROJNÍ

ÚSTAV MECHANIKY, BIOMECHANIKY A MECHATRONIKY

Odbor mechaniky a mechatroniky

Bakalářská práce

Plánovač trajektorie pro autonomní vozítko

Praha, 2024 **Matouš Kolář**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kolář** Jméno: **Matouš** Osobní číslo: **509613**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Plánovač trajektorie pro autonomní vozítko

Název bakalářské práce anglicky:

Trajectory planner for autonomous rover

Pokyny pro vypracování:

1 - Seznamte se s metodami plánování trajektorie mobilních robotů.
2 - Připravte simulační model vozítka.
3 - Implementujte vybraný algoritmus.
4 - Proveďte simulační experiment.

Seznam doporučené literatury:

[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The international journal of robotics research, vol. 30, no. 7, 2011, pp. 846–894.
[2] L. Xu, et al., "An Efficient Trajectory Planner for Car-Like Robots on Uneven Terrain." 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2023.
[3] L. Palmieri, S. Koenig and K. O. Arras, "RRT-based nonholonomic motion planning using any-angle path biasing," 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016, pp. 2775-2781.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Beneš, Ph.D. odbor mechaniky a mechatroniky FS

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

Datum zadání bakalářské práce: **26.04.2024** Termín odevzdání bakalářské práce: **16.08.2024**

Platnost zadání bakalářské práce: _____

Ing. Petr Beneš, Ph.D.	prof. Ing. Michael Valášek, DrSc.	doc. Ing. Miroslav Španiel, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

Datum převzetí zadání _____ Podpis studenta _____

Anotační list

Jméno autora: Matouš Kolář

Název bakalářské práce: Plánovač trajektorie pro autonomní vozítko

Anglický název: Trajectory planner for autonomous rover

Akademický rok: 2023/2023

Obor studia: Teoretický základ strojního inženýrství

Ústav/odbor: Ústav mechaniky, biomechaniky a mechatroniky
Odbor Mechaniky a mechatroniky

Vedoucí bakalářské práce: Ing. Petr Beneš, Ph. D

Bibliografické údaje: Počet stran: 65
Počet obrázků: 20
Počet příloh: Dva skripty – RTT, A* algoritmus

Klíčová slova:

Autonomní vozidlo, plánování trajektorie, algoritmus A*, mobilní robotika, simulace, Ackermannova kinematika, Matlab-Simulink, senzorová technologie, optimalizace cesty, heuristické metody, stochastické metody, RTT algoritmus, robotická vozidla, UGV

Keywords:

Autonomous vehicle, trajectory planning, A* algorithm, mobile robotics, simulation, Ackermann kinematics, Matlab-Simulink, sensor technology, path optimization, heuristic methods, stochastic methods, RTT algorithm, robotic vehicles, UGV

Anotace:

Tato bakalářská práce se zaměřuje na vývoj a implementaci plánovače trajektorie pro autonomní vozítko, přičemž klade důraz na využití moderních algoritmů pro navigaci v neznámém a členitém terénu. Hlavním cílem práce je navrhnout a ověřit efektivní metody plánování trajektorie, které umožní autonomnímu vozítku bezpečně dosáhnout cílového bodu.

Práce začíná teoretickým úvodem, ve kterém jsou představeny klíčové algoritmy, jako je A* a RRT (Rapidly-exploring Random Tree). Následuje praktická část zaměřená na implementaci plánovače trajektorie v prostředí MATLAB/Simulink. V této části je vytvořen simulační model, který slouží jako platforma pro testování navržených algoritmů.

Funkčnost plánovače je ověřena na příkladu výpočtu trajektorie pro šestikolový rover, který se pohybuje v simulovaném prostředí obsahujícím členitý terén a statické překážky. Tato práce tak přináší příspěvek na úrovni bakalářského studia k výzkumu v oblasti autonomních systémů a může sloužit jako základ pro další rozvoj a zlepšování navigačních algoritmů pro autonomní vozidla.

Abstract:

This bachelor's thesis focuses on the development and implementation of a trajectory planner for an autonomous vehicle, with an emphasis on utilizing modern algorithms for navigation in unknown and rugged terrain. The primary goal of the thesis is to design and verify effective trajectory planning methods that enable the autonomous vehicle to safely reach its target point.

The thesis begins with a theoretical introduction, presenting key algorithms such as A* and RRT (Rapidly-exploring Random Tree). This is followed by a practical section focused on the implementation of the trajectory planner in the MATLAB/Simulink environment. In this part, a simulation model is created to serve as a platform for testing the proposed algorithms.

The functionality of the trajectory planner is validated through the calculation of a trajectory for a six-wheeled rover, which navigates a simulated environment containing rugged terrain and static obstacles. This thesis thus contributes to undergraduate research in the field of autonomous systems and can serve as a foundation for further development and improvement of navigation algorithms for autonomous vehicles.

.

Poděkování

Tímto způsobem bych chtěl poděkovat rodině za podporu během studia, panu Ing. Petru Benešovi, Ph. D za vedení práce a týmu CTU Robotics za cenné rady během této práce.

Čestné prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

V Praze, dne …… ……
Podpis

Obsah

Seznam obrázků

1. Úvod

Plánování trajektorie pro autonomní vozidla a mobilní roboty představuje jeden z klíčových problémů, který stojí na pomezí technických věd a aplikovaného výzkumu v současné době. Tento proces zahrnuje nejen nalezení optimální cesty z bodu A do bodu B, ale také zajištění bezpečného a efektivního pohybu v dynamicky se měnícím prostředí, kde mohou nastat nepředvídatelné situace a překážky. Jak technologie postupuje vpřed, rostou i nároky na robustnost a spolehlivost těchto systémů, neboť se stávají nedílnou součástí moderního života – od průmyslové automatizace až po osobní dopravu.

Výzvy, které se v této oblasti objevují, jsou komplexní a mnohostranné. Plánovače trajektorie musí nejen efektivně pracovat s omezenými zdroji a výpočetním výkonem, ale zároveň musí být schopny adaptace na měnící se podmínky v reálném čase. To zahrnuje nejen detekci a vyhýbání se překážkám, ale také schopnost navigovat v členitém terénu nebo v prostředí, kde dochází k náhlým změnám, jako jsou pohyblivé objekty či změny povrchu.

Současné trendy ve vývoji těchto systémů ukazují na rostoucí integraci pokročilých algoritmů, jako jsou strojové učení a umělá inteligence, které umožňují robotům a vozidlům lépe porozumět svému okolí a učit se z něj. Díky tomu jsou schopny zlepšovat svou výkonnost nejen na základě předem naprogramovaných pravidel, ale také z vlastních zkušeností. To vede k vývoji systémů, které jsou nejen efektivní, ale také schopné pružně reagovat na nové a neznámé situace.

Zároveň se objevuje otázka etiky a bezpečnosti, kdy se autonomní systémy musí rozhodovat v situacích, které mohou mít zásadní dopad na lidský život. To klade důraz na transparentnost a ověřitelnost rozhodovacích procesů těchto systémů, což je nezbytné pro jejich široké přijetí a důvěru ze strany veřejnosti.

Výzkum v oblasti plánování trajektorie proto není jen technickým problémem, ale také součástí širšího diskurzu o budoucnosti lidské společnosti, kde autonomní systémy hrají stále větší roli. Propojení teorie s praxí, a neustálé zlepšování těchto technologií, je tedy nejen otázkou technického pokroku, ale i cestou k bezpečnější a efektivnější budoucnosti.

2. Cíl práce

Hlavní cíl

Cílem této bakalářské práce je poskytnout čtenáři základní pohled na problematiku plánování trajektorie pro autonomní vozidla, se zaměřením na vývoj a implementaci efektivního plánovače trajektorie pro mobilní robot. Čtenář by měl po přečtení práce získat základní vědomosti, jak plánovač trajektorie funguje. Samotný plánovač je omezen na generování ve staticky předem daném prostředí.

Dílčí cíle:

1) **Teoretický přehled metod plánování trajektorie:**

   Seznámit čtenáře s hlavním algoritmy používanými pro plánování trajektorie, jako jsou A* algoritmus, RRT.

2) **Fyzikální modelování vozidla:**

   Vysvětlit principy Ackermannovy kinematiky mobilního robota, které jsou základem pro správné pochopení pohybu a fyzikálních limitů vozidla.

3) **Implementace plánovače trajektorie v MATLAB/Simulink:**

   Představit postupy a kroky při vývoji a implementaci plánovače trajektorie v simulačním prostředí MATLAB/Simulink, včetně tvorby simulačního modelu šestikolového roveru.

4) **Prověření funkčnosti plánovače prostřednictvím simulací:**

   Provést a analyzovat simulační experiment, který ověří schopnost plánovače efektivně navigovat autonomní vozidlo v předem daných podmínkách se zaměřením na bezpečnost a optimalizaci trasy.

5) **Diskuse a zhodnocení výsledků:**

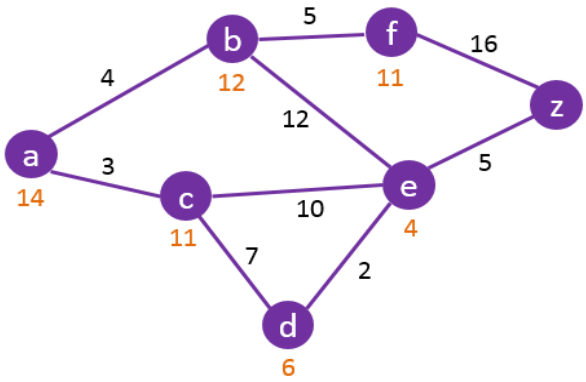
   Zhodnotit dosažené výsledky, diskutovat o výhodách a omezeních navrženého řešení, a navrhnout možná vylepšení nebo směr pro další výzkum v této oblasti.

3. Plánování trajektorie

3.1. Plánovací algoritmus

Plánování trajektorie mobilních robotů je nezbytné pro autonomní navigovaní v prostředí, v němž se nacházejí překážky, kterým se daný mobilní stroj vyhne za účelem dosažení předem daného cíle. Pro tento proces je důležité, aby autonomně řízený stroj obsahoval příslušnou senzorickou technologii, jenž daný proces řízení úspěšné vyhodnotí. Plánovací algoritmus je matematický postup, který určuje sekvenci kroků nebo cestu, kterou musí robot nebo autonomní vozidlo následovat, aby dosáhlo svého cíle bez kolize s překážkami [1][2].

Obecně mezi hlavní metody patří algoritmy pro hledání nejkratší cesty, resp. heuristické a stochastické algoritmy. Algoritmy pro nejkratší cesty zajišťují efektivní trasu mezi body, což je klíčové pro optimalizaci času a energie. Heuristické algoritmy využívají odhady k rychlejšímu nalezení cesty v komplexních prostředích. Jedním z nejrozšířenějších algoritmů je A* (A-star), který používá heuristiku pro rychlé nalezení optimální trasy [1][2].

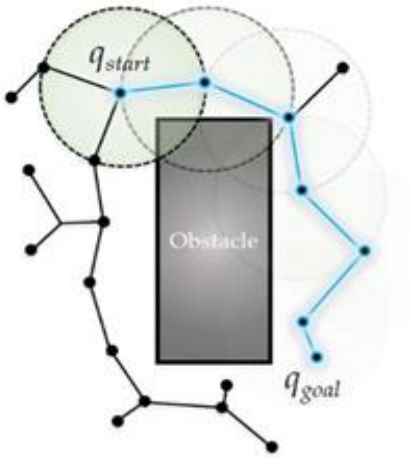


Obr 1. – Grafické znázornění A* – převzato z [3]

Na Obr 1. můžete vidět grafické znázornění uzlů a hran, kde každý uzel je označen písmenem (a, b, c, d, e, f, z) a propojen s ostatními uzly hranami s přiřazenými náklady (černá čísla). Červené hodnoty vedle uzlů představují heuristické odhady vzdálenosti od každého uzlu k cílovému uzlu z. Tento graf je typickým příkladem použití algoritmu A*, který kombinuje skutečné náklady cesty s odhadem zbývajících nákladů, aby nalezl nejefektivnější cestu od počátečního uzlu

„a“ k cílovému uzlu „z“. Tento diagram dobře ilustruje, jak A* naviguje v grafu, aby minimalizoval celkové náklady na cestu [3].

Stochastické algoritmy jsou vhodné pro prostředí s mnoha překážkami. Velice známým příkladem je RTT algoritmus (Rapidly-exploring Random Tree), který je užitečný v komplexních prostředích. Tento algoritmus vytváří strom možností, který se postupně rozšiřuje od počátečního bodu k cíli náhodným vzorkováním prostoru [4].



Obr 2. – Grafické znázornění RTT algoritmu – převzato z [4]

Na Obr 2. je vidět grafické znázornění cesty od výchozího bodu q_{start} k cílovému bodu q_{goal} s překážkou mezi těmito body. Obrázek zobrazuje hledání cesty algoritmem, který zohledňuje překážky. Překážka je znázorněna jako šedý obdélník uprostřed, zatímco cesta je reprezentována modrou křivkou vedoucí od startu k cíli. Oblasti kolem uzlů jsou znázorněny kruhy, které představují možné oblasti pohybu nebo rozšíření cesty. Tento diagram je typický pro vizualizaci algoritmů prohledávání prostoru, jako je například Rapidly-exploring Random Tree (RRT) nebo podobné algoritmy pro navigaci v prostředí s překážkami [4][5]

Pro popis trajektorie vozidla můžeme použít i kinematický model, který je reprezentován diferenciálními rovnicemi [5].

3.2. Způsoby plánování trajektorie

Plánování trajektorie zahrnuje určování efektivních a bezpečných tras, které musí robot nebo vozidlo sledovat, aby dosáhl svého cíle bez kolize s překážkami. Existuje několik metod plánování trajektorie, které využívají různé matematické přístupy a techniky. Jedním z nejběžnějších přístupů je metoda grafového vyhledávání, která zahrnuje algoritmy jako A* a Dijkstra [23]. Tyto algoritmy hledají optimální cestu v grafu, kde uzly představují možné stavy a hrany představují možné přechody mezi těmito stavy. Grafové metody jsou efektivní při nalezení globálně optimální trasy v diskrétní síti, ale mohou ztratit rozlišení kvůli diskretizaci stavového prostoru [6].

Optimalizace trajektorie je další metoda pro autonomní navigaci, která zahrnuje hledání lokálně optimálního řešení v prostoru pomocí variačních technik. Tento přístup využívá numerické metody k minimalizaci nákladové funkce a splnění daných okrajových podmínek. Optimalizace trajektorie je užitečná pro generování hladkých a spojitých trajektorií, které lépe vyhovují dynamickým omezením robotů [7][8].

Pro optimalizaci trajektorie je nutné definovat počáteční a cílové podmínky, dynamiku systému a nákladovou funkci. Nákladová funkce se často formuluje jako integrál Lagrangiánu přes čas. Lagrangián L je definován jako rozdíl kinetické a potencionální energie [8].

Dále se zavádí matematický pojem zvaný Funkcionál. Funkcionál je matematická funkce, která přiřazuje reálné číslo k celé funkci. Formální energetický zápis funkcionálu je definován takto [8]:

$$S[q(t)] = \int_{t_o}^{t_f} L(q(t), \dot{q}(t), t)dt \tag{3.1}$$

- $f = q(t)$ – funkce pro polohu měnící se v čase
- $t_0[s]$ – počátek časového intervalu
- $t_f[s]$ – konec časového intervalu

Princip nejmenší akce říká, že správná trajektorie $q(t)$ minimalizuje tuto akci. K nalezení trajektorie, která minimalizuje funkcionál $S[q(t)]$ používáme Eulerovy-Lagrangeovy rovnice [8]:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \tag{3.2}$$

- $L[J]$ - Lagrangián
- q_i – generalizovaná souřadnice (např. poloha)
- $\dot{q}_i$ – časová derivace generalizované souřadnice (rychlost)
- Q_i – zobecněný silový účinek zahrnující nekonzervativní a na rychlostně závislé síly a další účinky, které nejsou zahrnuté v potenciální energii.

Vzhledem ke složitosti řešeného problému se používají také numerické metody, jako je metoda konečných prvků nebo gradientní sestup, k nalezení přibližného řešení [10]

Optimalizace trajektorie pomocí variačních technik a Lagrangiánu poskytuje mocný nástroj pro nalezení optimálních trajektorií, které minimalizují nákladovou funkci a splňují dynamická omezení. Tento přístup je široce využitelný v robotice a dalších oblastech [11][12].

3.3. A* algoritmus

Algoritmus A* (A-star) je vyhledávací algoritmus široce používaný v grafové teorii a počítačové vědě pro navigaci a hledání nejkratší cesty mezi dvěma body. Tento algoritmus kombinuje výhody Dijkstrova algoritmu a heuristického přístupu, což umožňuje efektivní hledání optimálních cest. Algoritmus A* se stal standardem pro mnoho aplikací, včetně robotiky, herního vývoje a autonomních vozidel, díky své schopnosti rychle a efektivně nalézt nejkratší cestu v různých prostředích [5][7][23][35].

A* algoritmus pracuje na základě několika klíčových komponent:

1) Grafová reprezentace:

Prostor je reprezentován jako graf, kde vrcholy představují možné stavy nebo pozice a hrany představují možné cesty mezi těmito stavy. Každý uzel má souřadnice a je spojen s dalšími uzly pomocí hran, které reprezentují cesty. Hrany

mohou mít různé náklady, což umožňuje modelovat různé terény a překážky [5][13] [23][35].

2) Heuristická funkce:

A* algoritmus používá heuristickou funkci $h(n)$, která odhaduje nejnižší možnou cenu cesty z aktuálního uzlu n do cílového uzlu. Heuristika musí být přípustná, což znamená, že nikdy nepřekročí skutečné náklady cesty. Mezi běžné heuristické funkce patří Manhattanova vzdálenost, euklidovská vzdálenost a diagonální vzdálenost, které se používají v závislosti na typu mřížky a pohybových možnostech [7][13][14].

Heuristická funkce $h(n)$ je odhad nákladů na dosažení cílového uzlu z uzlu n. Je klíčovým prvkem A* algoritmu, protože určuje, jak efektivně algoritmus prohledává graf. Heuristická funkce by měla být přípustná, což znamená, že nikdy nepřeceňuje skutečné náklady na dosažení cíle [7][13][14][23].

Příklady heuristických funkcí:

a) Manhattanská vzdálenost [7]**:**

Používá se v mřížkovém prostoru, kde se pohybujeme pouze vodorovně a svisle.

$$h(n) = |x_n - x_{cíl}| + |y_n - y_{cíl}| \tag{3.3}$$

- x_n, y_n – referenční bod
- $x_{cíl}, y_{cíl}$ – koncový bod

b) Euklidovská vzdálenost [7]:

Používá se, když se můžeme pohybovat diagonálně.

$$h(n) = \sqrt{(x_n - x_{cíl})^2 + (y_n - y_{cíl})^2} \tag{3.4}$$

c) Čebyševova vzdálenost [7]:

Používá se, pokud se můžeme pohybovat horizontálně, vertikálně i diagonálně za stejný náklad.

$$h(n) = \max(|x_n - x_{cíl}|, |y_n - y_{cíl}|) \tag{3.5}$$

3) Funkce nákladů:

Algoritmus také udržuje hodnotu $g(n)$, která představuje nejnižší náklady cesty z počátečního uzlu do uzlu n. Tyto náklady se průběžně aktualizují, jak algoritmus prozkoumává různé cesty [5][7].

Výpočet nákladů:

a) V mřížkovém prostoru s jednotkovými náklady:

Pokud je náklad pohybu do sousedního uzlu vždy stejný (např. 1):

$$g(n) = g\big(parent(n)\big) + 1 \tag{3.6}$$

- $parent(n)$ - předchůdce aktuálního uzlu na cestě

b) V mřížkovém prostoru s různými náklady:

Pokud se náklady na pohyb do sousedního uzlu liší (např. různé terény):

$$g(n) = g\big(parent(n)\big) + cost(parent(n), n) \tag{3.7}$$

- $parent(n)$ - předchůdce aktuálního uzlu na cestě
- $cost(parent(n), n)$ - představuje specifické náklady na přesun z uzlu n

c) Celková nákladová funkce:

Algoritmus vybírá uzly k prozkoumání na základě nejnižší hodnoty f(n). Celkové náklady se vyjadřují jako:

$$f(n) = g(n) + h(n) \tag{3.8}$$

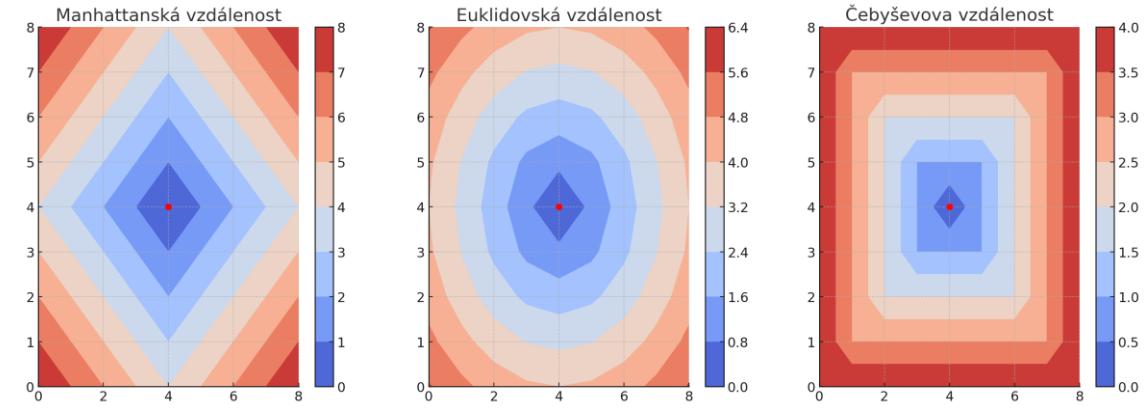
- $h(n)$ – funkce odhadující nejnižší náklady uzlu n do cíle.
- $g(n)$ – skutečný náklad z počátečního uzlu do uzlu n.

..

Celková nákladová funkce $f(n)$ v algoritmu A* je součtem dvou komponent: nákladů cesty $g(n)$ a heuristického odhadu $h(n)$. Tento součet slouží k výběru uzlů, které mají být prozkoumány, a umožňuje algoritmu efektivně nalézt nejkratší cestu v grafu.

Tento vzorec kombinuje aktuální náklady na cestu z počátečního uzlu do uzlu n s odhadem nákladů na dosažení cíle z uzlu n. Algoritmus A* vybírá uzly k prozkoumání na základě nejnižší hodnoty $f(n)$, což zajišťuje efektivní hledání nejkratší cesty [5][13][35].

Celková nákladová funkce $f(n)$ je klíčovým prvkem algoritmu A*, protože integruje jak skutečné, tak odhadované náklady, což umožňuje vyvážené a efektivní hledání cesty. Heuristika pomáhá algoritmu orientovat se v prostoru a minimalizovat prohledávané oblasti, zatímco náklady cesty zajišťují, že nalezená cesta je optimální [7][13][35].



Obr.3 – Znázornění heuristických funkcí

Příklady a aplikace:

1. **V autonomních vozidlech**:

   Celková nákladová funkce je kritická pro plánování tras autonomních vozidel, kde je nutné rychle a efektivně nalézt bezpečné a optimální cesty v dynamických prostředích. Použití vhodné heuristiky může výrazně zlepšit výkon plánování tras v reálném čase.

2. **V robotech v nerovném terénu**: Pro roboty pohybující se v nerovném terénu umožňuje celková nákladová funkce zohlednit různé terénní překážky a optimalizovat pohyb tak, aby minimalizoval náklady a rizika spojená s různými typy povrchu.

Postupové kroky A*star algoritmu [5][13][35]

a) Inicializace:

Na začátku se vytvoří dvě množiny: otevřená množina, která obsahuje počáteční uzel, a uzavřená množina, která je prázdná. Otevřená množina obsahuje uzly, které čekají na prozkoumání, zatímco uzavřená množina obsahuje uzly, které již byly prozkoumány.

b) Prohledávání:

Opakujte následující kroky, dokud otevřená množina není prázdná:

- Vyberte uzel *n* z otevřené množiny s nejnižší hodnotou *f(n)*.
- Pokud je *n* cílový uzel, cesta byla nalezena.
- Přesuňte *n* z otevřené množiny do uzavřené množiny.
- Pro každý sousední uzel *m*:
  - Pokud *m* je v uzavřené množině, ignorujte ho.
  - Vypočítejte náklady cesty *g(m)* a odhadované náklady *h(m)*.
  - Pokud *m* není v otevřené množině nebo nový výpočet *f(m)* je nižší než dřívější hodnota:
    - Nastavte hodnoty *g(m)* a *f(m).*
    - Nastavte předchůdce uzlu *m* na uzel *n*.
    - Pokud *m* není v otevřené množině, přidejte ho tam.

c) Výstup:

Pokud byl nalezen cílový uzel, rekonstruujte cestu pomocí předchůdců uzlů. Pokud otevřená množina je prázdná a cílový uzel nebyl nalezen, cesta neexistuje.

3.4. Hybridní A* algoritmus

Hybridní A* algoritmus: efektivní plánování cest v nerovném terénu

Hybridní A* algoritmus představuje pokročilý přístup k plánování tras, který kombinuje různé techniky a vylepšuje tradiční A* algoritmus tak, aby byl efektivní v náročných 3D prostředích s nerovným terénem. Tento algoritmus kombinuje různé heuristiky a nákladové funkce, aby lépe zohlednil složitost terénu a optimalizoval trasu pro vozidla nebo roboty, které se v těchto prostředích pohybují [15][16][36].

Kombinace heuristik v hybridním A* algoritmu

Heuristika hraje klíčovou roli v každém algoritmu pro plánování tras. V tradičním A* algoritmu se heuristika často omezuje na jednoduchý odhad vzdálenosti, jako je euklidovská vzdálenost. Tento přístup může být dostačující pro 2D prostředí, ale v 3D prostředích s nerovným terénem jsou takové jednoduché heuristiky nedostatečné [16][36].

Vícesložková heuristika [17][18][36].

V hybridním A* algoritmu je heuristická funkce rozšířena o více složek, které lépe odrážejí složitost prostředí. Tato kombinovaná heuristika je definována jako:

$$g(n) = \alpha . h_{vzdálenost}(n) + \beta . h_{normála}(n) + \gamma . h_{elevace}(n) \qquad (3.9)$$

- $h_{vzdálenost}(n)$: Tradiční vzdálenostní heuristika, která odhaduje zbývající vzdálenost k cíli, například pomocí euklidovské vzdálenosti.
- $h_{normála}(n)$: Heuristika zohledňující úhel mezi normálovými vektory povrchu, což je klíčové při navigaci v nerovném terénu. Tato složka je užitečná pro predikci obtížnosti průchodu terénem.
- $h_{elevace}(n)$: Heuristika, která zohledňuje změny nadmořské výšky mezi aktuálním bodem a cílem. Tento aspekt je kritický při plánování tras v kopcovitém nebo horském terénu
- α, β, γ: Váhy jsou to parametry, které lze upravit podle specifikace daného prostředí nebo aplikace, aby se dosáhlo optimálního výkonu algoritmu

Výhody a přínosy

Použití vícesložkové heuristiky poskytuje hybridnímu A* algoritmu výhodu v přesnosti odhadu cesty. Tradiční heuristiky, zaměřené pouze na vzdálenost, často přehlížejí složitost terénu, jako je sklon nebo nerovnosti. Integrace těchto aspektů do heuristiky umožňuje algoritmu lépe odhadnout skutečné náklady na pohyb a vyhnout se potenciálně riskantním trasám. Tento přístup vede k nalezení tras, které jsou bezpečnější a lépe přizpůsobené konkrétním podmínkám prostředí.

Nákladová funkce v hybridním A* algoritmu

Dalším klíčovým prvkem hybridního A* algoritmu je jeho nákladová funkce, která určuje skutečné náklady na přesun mezi uzly v grafu. Zohlednění více faktorů v této funkci umožňuje lepší modelování nákladů na pohyb v terénu a tím optimalizaci plánování tras [17][36].

Definice nákladové funkce

Nákladová funkce g(n) je definována jako:

$$g(n) = g\big(parent(n)\big) + cost(parent(n), n).(1 + k.(1 - \cos(\theta))) \qquad (4.0)$$

Kde:

- $g\big(parent(n)\big)$: Celkové náklady na dosažení rodičovského uzlu.
- $cost(parent(n), n)$: Náklady na přesun z rodičovského uzlu do uzlu n.
- θ: Úhel mezi normálovými vektory povrchu v rodičovském uzlu a v uzlu n.
- $k\ je\ parametr$, který může být upraven podle specifikací prostředí nebo aplikace

Tato funkce kombinuje jak přímé náklady na pohyb, tak i faktory zohledňující sklon terénu, což umožňuje přesnější plánování tras v náročných prostředích.

Algorithm 1 Hybrid A*

Input:
X_{start}, X_{goal} : Start and goal configuration
grid : Grid
$\mathcal{O}$: Obstacles

```
1:  procedure HA*
2:      q_start = X_start
3:      grid(q_start).list.insert(q_start)
4:      OpenList = PriorityQueue()
5:      OpenList.insert(q_start)
6:      while OpenList! = ∅ do
7:          q_exp = OpenList.pop()
8:          if q_exp ∈ X_goal^region then
9:              return q_exp
10:         end if
11:         q_new = Expand(q_exp, O)
12:         for each q̂_new ∈ q_new do
13:             if Valid(q̂_new, grid) then
14:                 grid(q̂_new).list.insert(q̂_new)
15:                 OpenList.insert(q̂_new)
16:             end if
17:         end for
18:     end while
19:     return ∅
20: end procedure
```

Obr.4 – Příklad psedokódu použitého pro hybridní A* - převzato z [20]

Praktické aplikace hybridního A* algoritmu

Hybridní A* algoritmus má široké spektrum aplikací, zejména v oblastech, kde je vyžadována vysoká přesnost a spolehlivost při plánování tras v komplexních prostředích.

1) Autonomní vozidla

V oblasti autonomních vozidel je tento algoritmus používán pro plánování tras, které musí zohledňovat jak dopravní podmínky, tak i terénní překážky. Hybridní A* algoritmus umožňuje autonomním vozidlům lépe navigovat v různých prostředích, od městských silnic po venkovské a horské cesty. Jeho schopnost zohlednit faktory

jako sklon silnice a nadmořskou výšku umožňuje efektivnější a bezpečnější provoz autonomních vozidel [18].

2) Robotika a drony

V robotice a při řízení dronů je hybridní A* algoritmus využíván pro plánování tras v 3D prostředích, kde je důležité zohlednit jak horizontální, tak vertikální pohyby. Tento algoritmus je zvláště užitečný v aplikacích, kde je nutné navigovat v těsné blízkosti překážek, jako jsou budovy nebo stromy, nebo v prostředích s proměnlivým terénem, jako jsou hory nebo lesy. Použití hybridního A* algoritmu umožňuje těmto systémům plánovat trasy, které minimalizují riziko kolize a optimalizují spotřebu energie [17].

3) Navigace v nerovném terénu

Jednou z hlavních výhod hybridního A* algoritmu je jeho schopnost efektivně plánovat trasy v nerovném terénu, kde tradiční algoritmy často selhávají. Tento algoritmus je schopen zohlednit složitost terénu, jako je sklon nebo nadmořská výška, což je klíčové pro aplikace, které vyžadují pohyb v obtížných podmínkách, jako jsou vojenské operace, záchranné mise nebo průzkum neznámých oblastí [17].

Výzvy a omezující faktory hybridního A* algoritmu

Navzdory mnoha výhodám hybridního A* algoritmu existují také výzvy a omezení spojené s jeho použitím. Jedním z hlavních omezení je zvýšená složitost výpočtů ve srovnání s tradičním A* algoritmem. Tento algoritmus vyžaduje více výpočetního výkonu a času, což může být problémem zejména v reálném čase, kdy je nutné rychlé rozhodování [17].

Dalším problémem je potřeba přesných a aktuálních dat o terénu. Aby mohl hybridní A* algoritmus správně fungovat, je třeba mít k dispozici detailní informace o terénu, včetně přesných modelů nadmořské výšky a povrchových vlastností. V prostředích, kde jsou tyto informace nedostupné nebo nepřesné, může být výkon algoritmu omezen [15][16].

3.5. RTT algoritmus

RRT (Rapidly-exploring Random Tree) algoritmus je jednou z nejčastěji používaných metod pro plánování cest v komplexních a neznámých prostředích. Tento algoritmus je obzvláště užitečný pro řešení problémů s vysokou dimenzí a v dynamických prostředích, kde tradiční metody často selhávají. RRT algoritmus využívá náhodného vzorkování pro rychlé prozkoumání prostoru a postupné vytvoření stromu, který pokrývá dosažitelný prostor. Tento algoritmus je výjimečný svou schopností rychle a efektivně prozkoumávat velké a složité prostory díky svému náhodnému charakteru, což mu umožňuje nalézat cesty i v situacích, kde by tradiční algoritmy nebyly efektivní [19][20][21].

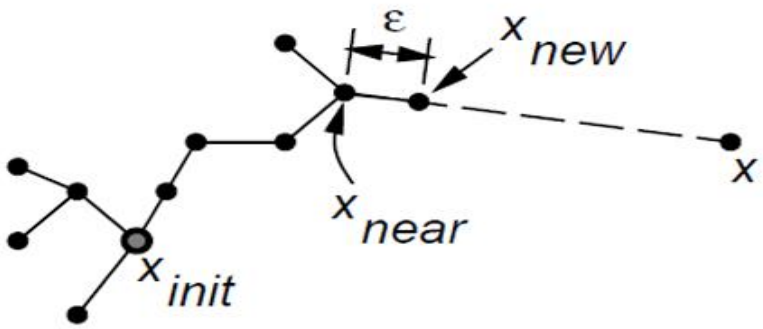
Matematický popis RRT algoritmu

RRT algoritmus začíná od počátečního bodu a postupně rozšiřuje strom pomocí náhodného vzorkování prostoru. Každý nový bod je přidán k nejbližšímu bodu v existujícím stromě a nová hrana je vytvořena mezi těmito dvěma body. Cílem je vytvořit strom, který rychle pokryje dosažitelný prostor, a nakonec dosáhne cílového bodu [19][20][21].

Matematicky je růst stromu řízen následujícími kroky:

1. **Náhodné vzorkování:** Generování náhodného bodu v prostoru.
2. **Nearest Neighbour (Nejbližší soused):** Nalezení nejbližšího bodu ve stromě k náhodně generovanému bodu.
3. **Rozšíření:** Vytvoření nové hrany mezi nejbližším bodem a náhodným bodem, pokud je tato hrana platná (tj. nedochází ke kolizi).
4. **Přidání nového bodu:** Pokud je hrana platná, nový bod je přidán do stromu.

Tyto kroky se opakují, dokud není dosaženo cílového bodu nebo dokud není dosažen maximální počet iterací. RRT algoritmus je znám svou jednoduchostí a efektivitou při náhodném vzorkování prostoru, což umožňuje rychlé prozkoumání velkých a složitých prostředí.



Obr.5 – Ukázka vzorkovaní cesty – převzato z [22]

Jedním z klíčových parametrů RRT algoritmu je délka kroku při rozšiřování stromu. Tento parametr určuje, jak daleko se nový bod může nacházet od nejbližšího bodu ve stromě. Příliš malá délka kroku může vést k pomalému růstu stromu, zatímco příliš velká délka kroku může způsobit, že strom vynechá úzké průchody nebo koliduje s překážkami. Volba správné délky kroku je kritická pro efektivní fungování algoritmu, a její optimalizace může výrazně ovlivnit výsledky, zejména v prostředích s komplexními překážkami [19][20][21].

Rozšíření na RTT* algoritmus

RRT algoritmus lze také rozšířit na RRT* (RRT-Star), který zajišťuje, že nalezená cesta je optimální. RRT* iterativně vylepšuje cestu tím, že zkoumá, zda existují levnější cesty k dosažení již prozkoumaných bodů, a přidává nové hrany k minimalizaci celkových nákladů. Tento přístup zajišťuje, že nalezená cesta není pouze schůdná, ale také optimalizovaná z hlediska délky či dalších nákladových kritérií, což je klíčové pro aplikace, kde jsou tyto faktory důležité [19][20][21].

Definice prostorů, stavů

Prostor stavů X představuje množinu všech možných stavů, které může robot zaujmout v daném prostředí. Prostor stavů může mít různou dimenzi v závislosti na povaze problému a typu robota. V nejjednodušším případě, například v dvourozměrném prostoru bez orientace, může být prostor stavů definován pouze dvojicí souřadnic (x, y). Avšak v reálných aplikacích, kde hraje roli i orientace robota, se často používá prostor stavů definovaný jako $SE(2)$ – speciální euklidovská grupa ve dvou rozměrech, kde každý stav je reprezentován trojicí

(x, y, θ), kde x a y jsou souřadnice v rovině a θ je úhel orientace robota vůči určité ose. Tento prostor je popsán následovně:

$$X = \{(x, y, \theta)|x, y \in \mathbb{R},\ \theta \in (0, 2\pi)\} \quad (4.1)$$

Tato definice stavového prostoru je klíčová pro správnou funkci RRT algoritmu, protože umožňuje algoritmu vzorkovat možné stavy, které robot může zaujmout, a plánovat trajektorii mezi těmito stavy [19][20][21].

Inicializace stromu

Strom T je datová struktura, která se postupně rozšiřuje během procesu plánování cesty. Inicializace stromu začíná vytvořením kořenového uzlu, který odpovídá počátečnímu stavu robota, tedy X_{start}.Tento kořenový uzel je jediným uzlem ve stromě na začátku algoritmu, což je reprezentováno takto:

$$T = \{X_{start}\} \quad (4.2)$$

Tato počáteční podmínka slouží jako výchozí bod, od kterého se algoritmus začne rozšiřovat a prozkoumávat dosažitelný prostor. Během každé iterace algoritmu se strom rozšiřuje přidáváním nových uzlů, což vede k pokrytí stále většího prostoru a přibližování se k cíli [19][20][21].

Náhodné vzorkování stavu

RRT algoritmus využívá techniku náhodného vzorkování, aby efektivně prozkoumal dostupný prostor. V každé iteraci algoritmus náhodně vybere nový stav X_{rand} z prostoru stavů X. Tento náhodně vybraný stav může být kdekoliv v prostoru, což umožňuje algoritmu prozkoumávat různé oblasti prostoru a hledat potenciálně optimální cesty. Náhodné vzorkování je klíčovým faktorem, který umožňuje RRT algoritmu vyhnout se lokálním minimům a zajistit, že bude prozkoumán celý dosažitelný prostor [19][20][21].

$$X_{rand} \in X \quad (4.3)$$

Najděte nejbližší uzel

Jakmile je vybrán náhodný stav X_{rand} , dalším krokem je nalezení nejbližšího uzlu ve stromě T, který je již prozkoumán. Tento uzel, označený jako X_{near}, je vybrán na

základě minimální Euklidovské vzdálenosti mezi X_{rand} a všemi uzly T . Tento krok je zásadní, protože určuje, odkud bude strom dále rozšiřován:

$$X_{near} = \arg\min_{X_i \in T} d(X_i, X_{rand}) \tag{4.4}$$

kde d představuje Euklidovskou vzdálenost. Tato metoda zajišťuje, že nový uzel bude přidán co nejblíže k náhodně vybranému bodu, ale zároveň zůstane propojen s existující strukturou stromu [19][20][21].

Vytvoření nového uzlu

Po nalezení nejbližšího uzlu X_{near} se vytvoří nový uzel X_{new}, který směřuje od X_{near} směrem k X_{rand}. Tento nový uzel je generován tak, aby byl posunut o určitou délku kroku δ směrem k X_{rand}:

$$X_{new} = X_{near} + \delta . \frac{X_{rand} - X_{near}}{\|X_{rand} - X_{near}\|} \tag{4.5}$$

Délka kroku δ je důležitý parametr, který ovlivňuje rychlost růstu stromu a jeho schopnost obcházet překážky. Příliš malý krok může vést k příliš pomalému růstu stromu, zatímco příliš velký krok může způsobit, že strom nebude schopen efektivně prozkoumat úzké průchody nebo se vyhne překážkám [19][20][21].

Kontrola kolizí

Předtím, než je nový uzel X_{new} přidán do stromu, je třeba zkontrolovat, zda cesta mezi X_{near} a X_{new} neprochází překážkou. Pokud není žádná kolize detekována, uzel je přidán do stromu:

$$T = T \cup \{X_{new}\} \tag{4.6}$$

Tento krok je klíčový pro zajištění, že generovaná cesta bude proveditelná a bezpečná pro robota. V prostředích s mnoha překážkami může kontrola kolizí představovat značnou výpočetní zátěž, ale je nezbytná pro správné fungování algoritmu [19][20][21].

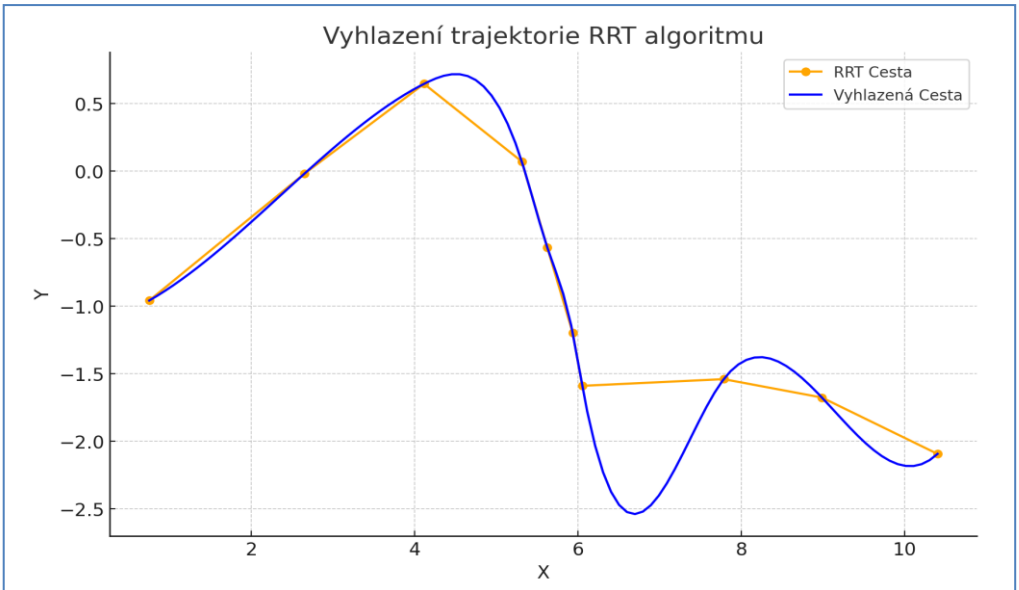
Opakování procesu

Celý proces, od náhodného vzorkování po kontrolu kolizí, se opakuje, dokud není dosaženo cílového stavu X_{goal} nebo dokud nejsou splněny jiné ukončovací podmínky, jako je dosažení maximálního počtu iterací. Opakování tohoto procesu zajišťuje, že strom postupně pokryje celý dosažitelný prostor a že bude nalezena cesta, která spojuje počáteční bod s cílem [19][20][21].

Optimalizace pomocí RRT*

RRT* algoritmus je vylepšenou verzí původního RRT algoritmu, která se zaměřuje na optimalizaci nalezené cesty. Při každé iteraci, když je přidán nový uzel X_{new}, algoritmus kontroluje, zda existují lepší (kratší nebo levnější) cesty, které by mohly být použity k dosažení tohoto uzlu. Tento proces se nazývá „re-wiring“ stromu.

Další fází optimalizace je „smoothing“ (vyhlazení) nalezené cesty, kde se odstraní redundantní uzly a omezí se maximální zakřivení cesty, aby byla cesta plynulejší a vhodnější pro praktické použití. Tato optimalizace je zvláště důležitá v reálných aplikacích, kde je třeba minimalizovat spotřebu energie, zkrátit čas cesty nebo zajistit bezpečnost robota při pohybu v prostředí [19][20][21].



Obr 6. – Ukázka optimalizace pomocí vyhlazení pro RTT algoritmus

3.6. Porovnání A* algoritmu a RTT algoritmu

A* (A-star) algoritmus a RRT (Rapidly-exploring Random Tree) algoritmus patří mezi nejpoužívanější techniky v oblasti plánování cest a vyhledávání tras v robotice a počítačové vědě. I když oba algoritmy slouží ke stejnému účelu – nalezení cesty z výchozího bodu do cílového bodu – liší se v přístupu, vlastnostech, výhodách a nevýhodách [19][23].

A* (A-star) Algoritmus

A* algoritmus je deterministický algoritmus, který kombinuje prvky Dijkstra algoritmu a Greedy Best-First-Search, což mu umožňuje najít optimální cestu mezi dvěma body na grafu efektivněji.

Výhody:

- **Optimalita**: A* algoritmus zaručuje nalezení optimální cesty, pokud je použita konzervativní heuristika.
- **Efektivita v pevných prostředích**: Je velmi efektivní v předem známých a strukturovaných prostředích, kde jsou překážky statické a dobře definované.
- **Flexibilita s heuristikami**: Heuristická funkce může být přizpůsobena specifickým požadavkům a vlastnostem prostředí, což může zlepšit výkon algoritmu.

Nevýhody:

- **Výpočetní náročnost**: A* algoritmus může být velmi náročný na výpočetní výkon, zejména ve velkých nebo vysoce dimenzionálních prostorech, kde je třeba prohledávat velké množství uzlů.
- **Závislost na kvalitě heuristiky**: Rychlost a efektivita vyhledávání jsou silně závislé na kvalitě použité heuristiky. Nevhodná heuristika může vést k neefektivnímu prohledávání.

Vhodnost:

- A* algoritmus je ideální pro použití v statických a dobře strukturovaných prostředích, kde jsou překážky neměnné a je možné předem znát strukturu prostoru.
- Je také vhodný tam, kde je důležité najít **optimální cestu** a není problém s delším výpočetním časem, například v aplikacích, kde je klíčová přesnost a spolehlivost.

RRT (Rapidly-exploring Random Tree) algoritmus

RRT algoritmus je stochastický algoritmus, který využívá náhodného vzorkování pro rychlé prozkoumání prostoru. RRT je obzvláště vhodný pro práci v komplexních a

vysoce dimenzionálních prostorech, kde by deterministické metody byly příliš pomalé nebo neefektivní [20][24].

Výhody:

- **Rychlé pokrytí prostoru**: Díky náhodnému vzorkování dokáže RRT rychle pokrýt velké a složité prostory, což je výhodné zejména v dynamických a neznámých prostředích.
- **Flexibilita a adaptabilita**: RRT může snadno adaptovat na změny v prostředí, například na pohyb překážek, což ho činí vhodným pro dynamické prostředí.
- **Schopnost pracovat ve vysoce dimenzionálních prostorech**: RRT je efektivní i v prostředích s mnoha dimenzemi, kde tradiční metody selhávají.

Nevýhody:

- **Nezaručuje optimální cestu**: RRT algoritmus nezaručuje, že nalezená cesta bude optimální. Cesta může být klikatá a nehladká, což vyžaduje další optimalizaci.
- **Stochastická povaha**: Kvůli náhodnému vzorkování mohou být výsledky nestabilní, což znamená, že různé běhy algoritmu mohou vést k různým cestám.

Vhodnost:

- RRT algoritmus je ideální pro použití v dynamických a neznámých prostředích, kde se překážky mohou měnit nebo kde není možné předem znát strukturu prostoru.
- Je také vhodný tam, kde je potřeba rychle prozkoumat velký vyhledávací prostor a najít schůdnou cestu, i když nemusí být optimální. To je výhodné v aplikacích, kde je důležitá rychlost nalezení řešení spíše než jeho optimalita.

A* a RRT algoritmy mají své specifické přednosti a omezení, které je předurčují k použití v různých typech prostředí a pro různé účely. *A algoritmus** je výhodný v situacích, kde je nutné najít optimální cestu v pevném a předem známém prostředí,

například při plánování tras v navigačních systémech, kde jsou silnice pevně dané a nemění se. Naopak RRT algoritmus je ideální pro dynamická a neznámá prostředí, jako je například navigace robotů v neznámém terénu nebo ve velmi komplexních prostorách, kde je třeba rychle najít schůdnou cestu, i když nemusí být optimální. [20][24].

Kombinace těchto algoritmů nebo jejich rozšíření (např. RRT*) může být využita k dosažení lepších výsledků v různých aplikacích, kde je třeba vyvážit rychlost a optimalitu nalezených cest.

3.7. Porovnání stochastické a heuristické metodiky plánování

Dva z nejčastěji používaných přístupů k plánování trajektorií jsou stochastické a heuristické algoritmy, z nichž každý má své specifické výhody a nevýhody.

Stochastické algoritmy, jako je RRT (Rapidly-exploring Random Tree), využívají náhodného vzorkování k prozkoumání prostoru a nalezení cesty k cíli. Algoritmus začíná z výchozího bodu a postupně vytváří strom náhodně generovaných bodů, které postupně pokrývají dosažitelný prostor. Tento přístup je obzvláště užitečný v prostředích s vysokou složitostí nebo neznámou strukturou, kde tradiční deterministické metody nemusí být efektivní [24].

Jednou z hlavních výhod stochastických algoritmů je jejich schopnost rychle pokrýt velké a složité prostory. Náhodné vzorkování umožňuje RRT rychle identifikovat cestu v prostředích, kde je mnoho překážek nebo neznámých prvků. To je obzvláště výhodné v dynamických a neznámých prostředích, kde se překážky mohou měnit a kde není možné předem znát strukturu prostoru. Další výhodou je flexibilita a schopnost přizpůsobit se změnám v prostředí. RRT může snadno upravit svůj strom a pokračovat v prozkoumávání, pokud se změní konfigurace prostředí nebo pokud se objeví nové překážky. Také jsou účinné ve vysoce dimenzionálních prostorech, kde tradiční metody selhávají kvůli výpočetní náročnosti [19][24].

Na druhé straně však stochastické algoritmy mají i své nevýhody. RRT algoritmus nezaručuje nalezení optimální cesty. Výsledné cesty mohou být nehladké a klikaté, což vyžaduje další optimalizaci. Navíc, protože je tento algoritmus založen na náhodném vzorkování, může generovat různé výsledky při různých běžích, což může vést k nestabilním výstupům. Tyto nevýhody jsou kritické zejména v aplikacích, kde je vyžadována vysoká přesnost a optimalita cesty.

Heuristické algoritmy, jako je A* (A-star), naopak využívají heuristiky, což jsou odhady nákladů na dosažení cíle z daného bodu. Tento přístup kombinuje systematičnost Dijkstra algoritmu s efektivitou Greedy Best-First-Search, což umožňuje algoritmu najít optimální cestu, pokud je heuristika konzervativní. Heuristika je matematická funkce, která určuje, jak „dobrá“ je cesta z daného uzlu k cíli, což umožňuje efektivní prohledávání prostoru [23][24].

Mezi hlavní výhody heuristických algoritmů patří jejich schopnost najít optimální cestu. A* algoritmus, pokud je použit s vhodnou heuristikou, zaručuje nalezení nejkratší nebo nejlevnější cesty k cíli. To je obzvláště důležité v situacích, kde je nezbytná přesnost a optimalita, například v městské navigaci nebo indoorových aplikacích, kde jsou překážky pevně dané a předem známé. Navíc je A* velmi efektivní v předem známých a strukturovaných prostředích, kde se překážky nemění. Flexibilita heuristické funkce také umožňuje přizpůsobení algoritmu konkrétním požadavkům prostředí, což může výrazně zlepšit jeho výkon [23][24].

Nicméně, heuristické algoritmy nejsou bez nevýhod. A* algoritmus může být velmi náročný na výpočetní výkon, zejména v prostředích s velkým množstvím uzlů nebo ve velkých prostorech, kde je třeba prohledávat velké množství možných cest . Navíc je jeho výkon silně závislý na kvalitě použité heuristiky. Pokud je heuristika nevhodná nebo špatně nastavená, může algoritmus generovat suboptimální výsledky nebo trvat příliš dlouho, než najde cestu [23][24].

Při výběru mezi stochastickými a heuristickými algoritmy je třeba zvážit povahu prostředí a specifické požadavky aplikace. Stochastické algoritmy jsou nejvhodnější pro dynamická a neznámá prostředí, kde je potřeba rychle prozkoumat velký vyhledávací prostor a najít schůdnou cestu i v situacích, kde tradiční algoritmy selhávají. Jsou také ideální pro aplikace ve vysoce dimenzionálních prostorech, kde jsou jiné metody příliš pomalé nebo neefektivní [23][24].

Na druhou stranu heuristické algoritmy excelují ve statických a strukturovaných prostředích, kde je klíčové najít optimální cestu. Tyto algoritmy jsou ideální pro aplikace, kde je možné předem získat informace o prostředí a kde je prioritou nalezení nejkratší nebo nejlevnější cesty, i když to může trvat delší dobu .

V praxi se často využívají kombinace těchto přístupů, například použitím RRT* algoritmu, který zlepšuje výsledky původního RRT tím, že iterativně optimalizuje nalezenou cestu. Takové kombinace poskytují výhody rychlého prozkoumání

prostoru spolu s možností nalezení optimální nebo téměř optimální cesty, což je zvláště výhodné v komplexních a dynamických prostředích [23][24].

4. Pozemní mobilní roboti – UGV

Pozemní mobilní roboty, známé také jako UGV (Unmanned Ground Vehicles), jsou autonomní nebo dálkově řízená vozidla, která operují na povrchu Země bez přímé lidské přítomnosti. Tyto roboty jsou vybaveny různými senzory, kamerami a algoritmy, které jim umožňují navigaci a plnění úkolů v různých prostředích. UGV se používají v mnoha oblastech, jako je průzkum, záchrana, vojenské operace, zemědělství a průmyslové aplikace. Jejich schopnost pracovat v nebezpečných nebo nepřístupných oblastech z nich činí klíčový nástroj pro moderní technologie a operace.

UGV jsou často vybaveny pokročilými navigačními systémy, které využívají kombinaci GPS, inerciálních měřicích jednotek (IMU) a různých senzorů, jako jsou LIDAR a ultrazvukové senzory. Tyto technologie umožňují robotům mapovat a rozpoznávat prostředí, detekovat překážky a plánovat trasy. Díky pokroku v oblasti umělé inteligence a strojového učení jsou moderní UGV schopny autonomně rozhodovat a adaptovat se na měnící se podmínky v reálném čase [25].

Implementace UGV zahrnuje několik klíčových komponentů. Nejprve je nutné navrhnout mechanickou konstrukci robota, která zahrnuje podvozek, pohonný systém a senzory. Dále je třeba vyvinout řídicí systém, který integruje data ze senzorů a umožňuje robotovi provádět požadované úkoly. Poslední fází je testování a validace systému, což zahrnuje simulace i reálné testy v různých prostředích [25].

UGV jsou široce používány v mnoha aplikacích. Například v průmyslových skladech se UGV používají k přepravě zboží, což zvyšuje efektivitu a snižuje potřebu lidské práce. V zemědělství mohou UGV provádět úkoly jako setí, hnojení a sklizeň plodin. V oblasti bezpečnosti a záchrany se UGV používají k průzkumu nebezpečných oblastí, detekci výbušnin a záchranářským operacím v oblastech postižených přírodními katastrofami [25].

Dělení UGV [26][27]

UGV vozidla lze rozdělit podle různých kritérií, přičemž mezi nejčastější patří dělení podle způsobu řízení, typu pohonu a oblasti použití:

1. **Podle způsobu řízení**:
    - **Autonomní UGV**: Tato vozidla jsou schopna samostatně plánovat a vykonávat své úkoly bez lidského zásahu. Využívají pokročilé senzory a algoritmy umělé inteligence pro rozhodování v reálném čase.
    - **Dálkově řízené UGV**: Tato vozidla jsou řízena na dálku operátorem, který monitoruje a ovládá jejich pohyb a činnost pomocí videokamer a senzorů na vozidle.
2. **Podle typu pohonu**:
    - **Kola**: UGV vybavená kolovým podvozkem jsou rychlá a efektivní na pevném, rovném povrchu, například na silnicích nebo v budovách.
    - **Pásy**: UGV s pásovým podvozkem jsou vhodnější pro náročnější terény, jako jsou písčité, bahnité nebo skalnaté povrchy, kde kola mohou mít omezenou trakci.
    - **Chodící (bipedální či kvadrupedální)**: Některé pokročilé UGV jsou vybaveny nohama, které jim umožňují překonávat složité překážky a pohybovat se v těžko přístupných oblastech.
3. **Podle oblasti použití**:
    - **Vojenské UGV**: Používají se pro průzkum, odstraňování výbušnin a podpůrné logistické operace v bojových zónách.
    - **Průmyslové UGV**: Používají se v továrnách a skladech k přepravě materiálu, inventarizaci a další logistické činnosti.
    - **Záchranářské UGV**: Využívají se v oblastech postižených přírodními katastrofami, při záchranářských operacích nebo při průzkumu nebezpečných oblastí, kde je přítomnost člověka riskantní.

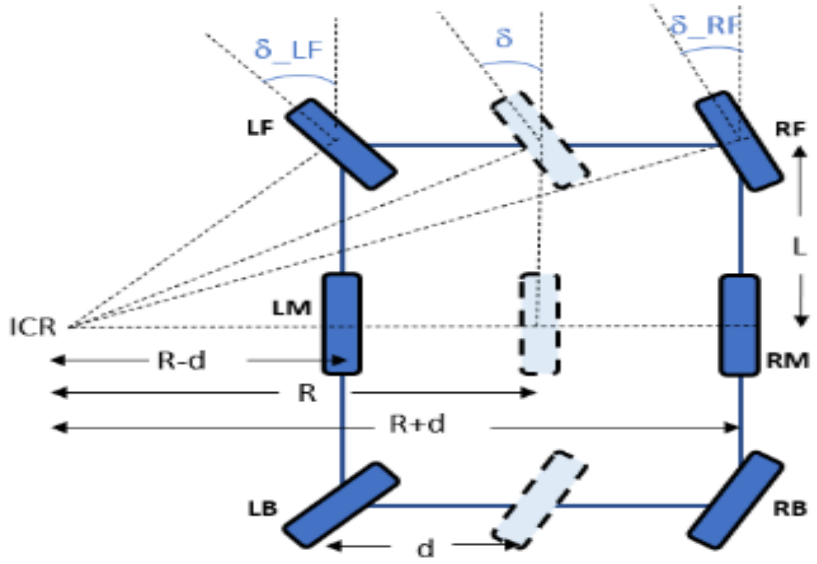
4.1. Ackermannova kinematika

Ackermannova kinematika řízení je klíčovým konceptem při návrhu vozidel s precizním ovládáním kol během zatáčení [28]. Tento systém, pojmenovaný po Rudolfu Ackermannovi, který jej patentoval v roce 1818, byl původně navržen pro koňské povozy, ale dnes se široce používá v moderním automobilovém průmyslu, včetně šestikolových vozidel a roverů, které operují v náročných terénních podmínkách [28].

Ackermannovo řízení – šestikolová vozítka

Vozidla s šesti koly, jako jsou robotické rovery používané pro planetární průzkum, musí při zatáčení zvládat různé poloměry zatáčení pro každou osu [29]. Zatímco základní princip Ackermannovy kinematiky zůstává stejný jako u čtyřkolových vozidel, implementace v šestikolových vozidlech je složitější kvůli potřebě koordinace více náprav [30]. Klíčové aspekty fungování zahrnují:

- **Koordinace více náprav**: V šestikolových vozidlech se Ackermannova kinematika musí postarat o synchronizaci přední a případně i zadní řízené nápravy tak, aby kola na všech řízených osách správně sledovala své ideální dráhy [31].
- **Diferenciace úhlů zatáčení**: Každé kolo musí sledovat svůj specifický poloměr zatáčení, což znamená, že vnitřní kola mají větší úhel natočení než kola vnější. V případě šestikolového vozidla to znamená, že nejen přední, ale i zadní kola mohou být řízena podle Ackermannovy kinematiky, aby se minimalizoval skluz [31].



Obr 7. – Ukázka Ackermannovy kinematiky– převzato z [32]

Matematická formulace Ackermannovy kinematiky v šestikolových vozidlech

Pro šestikolová vozidla je nutné přizpůsobit základní rovnice Ackermannovy kinematiky, aby odpovídaly složitějšímu uspořádání náprav [33]

$$\tan(\delta_{LF}) = \frac{L}{R - \frac{W}{2}} \tag{4.7}$$

$$\tan(\delta_{RF}) = \frac{L}{R + \frac{W}{2}} \tag{4.8}$$

- L - vzdálenost mezi přední a zadní nápravou (rozvor)
- W - vzdálenost mezi levým a pravým kolem (rozchod)
- δ_{LF} - úhel řízení vnitřního kola
- δ_{RF} - úhel řízení vnějšího kola
- R – poloměr zatočení okolo referrenčho středu

Výhody Ackermannovy kinematiky pro rovery

Ackermannova kinematika je obzvláště výhodná pro šestikolové rovery, které operují na nerovném a často neznámém terénu, jako je povrch Marsu:

- **Minimalizace skluzu a opotřebení pneumatik**: V terénních podmínkách, kde rovery operují, je minimalizace skluzu klíčová. Ackermannova kinematika zajišťuje, že každé kolo sleduje optimální dráhu, což snižuje riziko skluzu a opotřebení pneumatik [33].
- **Zlepšená ovladatelnost**: Šestikolové rovery musí být schopny přesně manévrovat v náročném terénu. Ackermannova kinematika poskytuje lepší ovladatelnost tím, že umožňuje optimalizované zatáčení, což je zásadní pro udržení stability a směru jízdy na nerovném povrchu [33].
- **Energetická efektivnost**: Rovy, které mají omezené energetické zdroje, těží z energetické efektivity Ackermannovy kinematiky, protože minimalizuje ztráty energie způsobené třením pneumatik [33].

Nevýhody a omezení [29] [30].

Přestože Ackermannova kinematika nabízí mnoho výhod, má také svá omezení. V kontextu šestikolových roverů mohou být tyto problémy ještě složitější:

- **Komplexnost návrhu**: Implementace Ackermannovy kinematiky v šestikolových vozidlech je složitější kvůli nutnosti přesně koordinovat více náprav.
- **Omezená účinnost při vysokých rychlostech**: Ackermannova kinematika je méně účinná při vysokých rychlostech, což však není kritické u roverů, které se obvykle pohybují pomalu.

Pokročilé úvahy a adaptace

U roverů, zejména těch, které operují v extrémních podmínkách, jako je povrch Marsu, se základní Ackermannova kinematika může kombinovat s moderními technologiemi a algoritmy řízení. Například algoritmy jako Pure Pursuit mohou být integrovány pro zlepšení navigace, přičemž Ackermannova kinematika zajišťuje optimální úhly zatáčení [33].

5. Tvorba plánovače v prostředí MATLAB – Simulink

V rámci praktické části této bakalářské práce bylo nezbytné vytvořit vlastní plánovač trajektorie pro mobilní robot. Tento plánovač byl vyvinut v programátorském prostředí Matlab, kde bylo potřeba implementovat skript, který bude zohledňovat jak statické, tak dynamické plánování. Obecně základem tohoto plánovače je matematický aparát, který respektuje fyzikální limity vozidla a bere v potaz členitost terénu. Tato členitost zahrnuje gradient terénu a různé překážky, které jsou detekovány pomocí přidané senzoriky ve vozidle. Plánovač také zohledňuje jak statické, tak dynamické překážky, které jsou zaznamenávány pomocí senzorů, jako jsou gyroskopické akcelerometry, lidary, ultrazvukové senzory a kamery. Na základě vyhodnocení těchto senzorů pomocí matematického aparátu v rámci programátorského rozhraní se vytvoří trajektorie, která respektuje jízdní schopnosti vozidla.

V některých případech může nastat situace, kdy vozidlo bude vysíláno do neznámého terénu. V takových situacích je potřeba přidat další softwarový aparát, například algoritmus SLAM (Simultaneous Localization and Mapping), který se snaží vytvořit tzv. point cloud prostoru okolo vozidla. Tento point cloud je poté dobře zpracovatelný pro následné fáze plánování. SLAM se běžně používá v robotice k mapování neznámého prostředí a současně k lokalizaci robotu v této mapě. Algoritmus pracuje na základě kombinace dat ze senzorů a kinematických modelů

vozidla, což umožňuje robotu vytvořit mapu svého okolí a určit svou polohu v této mapě.

Tato práce se však zaměří pouze na plánování trajektorie ve známém terénu. Terén bude daným způsobem členitý, avšak nebude obsahovat překážky, a také se nebude brát v potaz specifické fyzikální limitace vozidla. Pro samotné generování trajektorie jsme zvolili hybridní algoritmus A*, kdy budeme brát v úvahu odchylku normálu povrchu a normálu vozidla. Algoritmus A* je klasický algoritmus pro hledání nejkratší cesty v grafu, který je rozšířen o různé heuristiky a omezení, aby bylo možné plánovat trajektorii i ve složitějších podmínkách.

5.1.1. Plánovač – A* algoritmus

Pseudokód a implementace plánovače

Prvním krokem při tvorbě plánovače bylo vytvoření pseudokódu, který sloužil jako návod pro orientaci při psaní skriptu v Matlabu. Tento pseudokód zahrnuje definici prostoru, počáteční a koncové podmínky, a implementaci hybridního algoritmu A*.

Vytvořený pseudokód

Vstupy:

- grid: mřížka s informacemi o prostředí.
- start: startovní bod.
- goal: cílový bod.

Výstupy:

- path: nalezená cesta z start do goal.

Kroky:

1. **Inicializace:**
    - Vytvořit množinu openset, která obsahuje pouze startovní bod.
    - Inicializovat gScore pro startovní bod na 0.
        - gScore uchovává nejnižší známé náklady na dosažení určitého uzlu z počátečního bodu.
    - Vypočet fScore pro startovní bod jako součet gScore(start) a heuristické odhadované hodnoty HeuristicEstimate(start, goal).
        - fScore je kombinace gScore (skutečných nákladů na dosažení uzlu) a heuristického odhadu nákladů k dosažení cíle. Pomáhá algoritmu odhadnout, jak "blízko" je uzel k cíli.

2. **Hlavní smyčka:**
    - Dokud openset není prázdná:
        a. Vyberte uzel current z openset s nejnižší hodnotou fScore.
        b. Pokud je current rovno cílovému bodu:
            - Rekonstruujte cestu pomocí funkce ReconstructPath.
            - Návrat nalezené cesty path.
            - Ukončete smyčku.
        c. Odeberte current z openset.
        d. Pro každého souseda neighbour uzlu current:
            - Vypočet předběžné gScore (gScoreTentative) jako součet gScore(current) a vzdálenosti k sousednímu uzlu.
                - gScoreTentative představuje nové možné náklady na dosažení sousedního uzlu přes aktuální uzel.
            - Pokud je gScoreTentative menší než aktuální gScore(neighbour):
                - Aktualizovat gScore(neighbour) na gScoreTentative.
                - Aktualizovat fScore(neighbour) na součet gScore(neighbour) a heuristické hodnoty HeuristicEstimate(neighbor, goal).
                - Přidat neighbour do openset, pokud tam již není.
                - Tímto způsobem algoritmus aktualizuje své odhady nákladů na dosažení cíle, čímž upřednostňuje uzly, které se zdají být blíže cíli.
3. **Závěr:**
    - Pokud openset je prázdná a nebyla nalezena cesta, vrátit "FAILURE".

Popis vytvořeného kód v Matlabu

Tímto způsobem byla zadána počáteční struktura pro programování v Matlabu, ze které se následně vytvořil daný kód. Následující popis vytvořeného kódu souvisí s kódem, jenž se vyskytuje v příloze. Tato struktura poskytuje základní

rámec pro vývoj algoritmu plánování trajektorie, který má za cíl navigovat autonomní vozidlo v členitém terénu

1. Definice terénu a jeho charakteristik

Kód začíná definicí dvourozměrné mřížky, která představuje prostor pro pohyb roveru. Tato mřížka je definována jako síť bodů s koordináty X a Y, přičemž každý bod má přiřazenou výškovou hodnotu Z. Tato výšková hodnota představuje členitý terén s několika vrcholy a údolími.

Vytvoření základní mřížky

Nejprve je mřížka vytvořena pomocí funkce meshgrid, která generuje rovnoměrně rozložené hodnoty X a Y v rozsahu od 0 do 10. Tato mřížka má rozlišení 100x100 bodů, což poskytuje dostatečnou jemnost pro simulaci terénu. Každý bod na mřížce reprezentuje konkrétní místo v prostoru, kde může rover operovat.

Vytvoření základního terénu

Pro vytvoření základního terénu je použita kombinace několika matematických funkcí:

- **Sinusová a kosinusová funkce:** Tyto funkce generují vlnění na povrchu terénu, což vytváří menší vrcholy a údolí. Funkce sin(X) a cos(Y) zajišťují, že terén bude mít periodický charakter, což odpovídá přirozenému výskytu menších terénních vln v přírodě.
- **Funkce peaks:** Tato funkce generuje složitější povrch s výraznějšími vrcholy a údolími. Jejím přidáním se zajistí, že terén bude mít nejen menší lokální výstupky, ale také větší vrcholy, které simulují obtížnější terénní překážky. Funkce peaks je známá pro svou schopnost vytvářet zajímavé a různorodé terénní tvary.
-

Přidání významných vrcholů

K základnímu terénu jsou přidány čtyři výrazné vrcholy pomocí exponenciálních funkcí. Tyto vrcholy jsou umístěny na specifických pozicích v terénu. Použití exponenciálních funkcí zajišťuje, že tyto vrcholy budou výrazně vyšší než okolní terén a budou představovat významné překážky, které musí rover překonat.

Význam terénu pro simulaci

Tento členitý terén je navržen tak, aby simuloval reálné prostředí, kde by mohl rover operovat. Složitost a členitost povrchu jsou klíčovými faktory pro testování schopnosti plánovače trajektorie vytvářet vhodné a bezpečné cesty. Terén obsahuje jak hladké oblasti, které jsou snadno překonatelné, tak i obtížné vrcholy, které představují výzvy pro algoritmus.

2. Nastavení počátečních a koncových bodů

Po definici terénu je dalším krokem nastavení počátečních a cílových bodů, mezi kterými bude algoritmus hledat optimální trajektorii. Tyto body byly pečlivě vybrány tak, aby zajistily co největší pokrytí terénu a umožnily algoritmu projít přes různé výškové překážky.

Definování startovního a cílového bodu

- **Startovní bod:** Je nastaven na souřadnice [1,1], což odpovídá levému dolnímu rohu mřížky. Toto umístění zajišťuje, že rover začne svou cestu na jednom konci terénu.
- **Cílový bod:** Je nastaven na souřadnice [100,100], což odpovídá pravému hornímu rohu mřížky. Tento bod představuje konec cesty a zajistí, že trajektorie pokryje celý terén.

Význam výběru bodů

Výběr těchto bodů zajišťuje, že trajektorie bude pokrývat celou oblast terénu. Tento rozsah umožňuje algoritmu prověřit svou schopnost efektivně navigovat v členitém terénu a najít cestu, která bude realizovatelná pro rover.

3. Implementace hybridního A* algoritmu

Klíčovou částí tohoto projektu je implementace hybridního A* algoritmu pro plánování trajektorie. Tento algoritmus byl zde upraven tak, aby bral v úvahu nejen vzdálenost mezi body, ale také topografii terénu

Inicializace algoritmu

- **Prioritní fronta (openList):** Používá se pro správu uzlů, které budou prozkoumány. Uzel s nejnižší hodnotou fCost je vždy zpracován jako první.

- **Uzavřená logická matice (closedList):** Sleduje uzly, které již byly zpracovány, aby se zabránilo jejich opětovnému zpracování.
- **Pole cameFrom:** Uchovává informace o předchozím uzlu pro každý bod, což umožňuje zpětné sestavení cesty po nalezení cíle.
- **Náklady gCost a fCost:** gCost představuje náklady na dosažení daného uzlu z počátečního bodu, zatímco fCost je součet gCost a heuristické hodnoty, která odhaduje zbývající náklady na dosažení cíle.

Výpočet heuristiky

Heuristika v tomto algoritmu kombinuje několik faktorů:

- **Vzdálenost:** Tradiční heuristika na bázi euklidovské vzdálenosti, která odhaduje přímou vzdálenost mezi body.
- **Normály povrchu:** Výpočet úhlu mezi normálou povrchu a vertikální osou, což pomáhá algoritmu vyhnout se strmým svahům.
- **Výškový rozdíl:** Absolutní rozdíl v nadmořské výšce mezi daným bodem a cílem, což přispívá k preferenci cesty s menšími výškovými rozdíly.

Hlavní smyčka algoritmu

V hlavní smyčce algoritmu se iterativně vybírá uzel s nejnižší hodnotou fCost z prioritní fronty a zpracovává se jeho sousední uzly. Každý sousední uzel je vyhodnocen na základě několika kritérií:

- **Výškový rozdíl:** Uzel je přijatelný pouze tehdy, pokud výškový rozdíl mezi aktuálním uzlem a sousedním uzlem nepřekračuje stanovenou hodnotu.
- **Úhel sklonu:** Vypočítá se úhel mezi normálou povrchu a vertikální osou (Z-osou). Pokud je tento úhel menší nebo roven maximálnímu povolenému úhlu (30 stupňů), uzel je považován za průchozí.
- **Aktualizace nákladů:** Pokud jsou náklady na dosažení sousedního uzlu nižší než dříve vypočítané náklady, jsou tyto náklady aktualizovány a sousední uzel je přidán do prioritní fronty.

4. Generování trajektorie

Po nalezení optimální cesty mezi startovním a cílovým bodem je dalším krokem rozdělení trajektorie na menší úseky, které budou sloužit jako referenční body pro řízení pohybu vozidla.

Výpočet bodů trajektorie

Kód rozděluje nalezenou cestu na 30 bodů pomocí proměnné num_points. Tyto body jsou vybírány rovnoměrně podél celé trasy, aby se zajistila plynulost pohybu vozidla. Každý bod je reprezentován svými souřadnicemi X, Y a Z, které jsou vypočítány pomocí funkce sub2ind, která převádí dvojice indexů na lineární indexy v mřížce.

Přiřazení souřadnic

Vypočítané souřadnice trajektorie jsou poté přiřazeny do pracovního prostoru Matlabu pomocí funkce assignin. Tímto způsobem mohou být použity pro další zpracování nebo vizualizaci v rámci prostředí Matlab.

5. Vizualizace terénu a trajektorie

Vizualizace je klíčovou součástí tohoto projektu, protože umožňuje lépe porozumět chování algoritmu a trajektorie v reálném prostředí. Pomocí funkcí Matlabu pro trojrozměrné vykreslování je možné zobrazit terén a na něm znázorněnou trajektorii, což poskytuje vizuální zpětnou vazbu na kvalitu a vhodnost generované trajektorie.

Vykreslení terénu

Nejprve je terén vykreslen pomocí funkce surf, která vytváří trojrozměrný povrch. Barevná mapa jet se používá k zobrazení různých výškových hodnot na povrchu, což usnadňuje identifikaci vrcholů a údolí. Přidání barevného pruhu (colorbar) poskytuje informaci o rozsahu výškových hodnot.

Zobrazení trajektorie

Trajektorie je vykreslena jako červená čára, která vede od startovního bodu k cílovému bodu. Tímto způsobem je možné vizuálně sledovat, jak algoritmus naviguje rover přes členitý terén. Startovní bod je označen zeleným kruhem, zatímco cílový bod je označen modrým kruhem, což usnadňuje orientaci na grafu.

6. Výpočet a vizualizace povrchových normál

Pro lepší pochopení, jakým způsobem se vozidlo bude pohybovat po terénu, jsou vykresleny normály povrchu a porovnány s normálou trajektorie.

Výpočet normál

Normály povrchu jsou vypočítány pomocí funkce surfnorm, která bere v úvahu souřadnice X, Y a výškový profil Z. Tato funkce poskytuje vektory normál, které ukazují, jakým směrem je povrch orientován v každém bodě.

Vizualizace normál

Normály jsou vykresleny jako vektory pomocí funkce quiver3. Každý vektor je zmenšen měřítkem, aby byl lépe viditelný a nezakrýval trajektorii. Tyto vektory pomáhají vizuálně ověřit, zda jsou povrchy, po kterých se vozidlo pohybuje, dostatečně realizovatelné pro jízdu.

7. Výpočet úhlů sklonu

Kód také vypočítává úhly mezi normálou povrchu a trajektorií, což je důležité pro zajištění, že vozidlo nebude muset překonávat příliš strmé svahy.

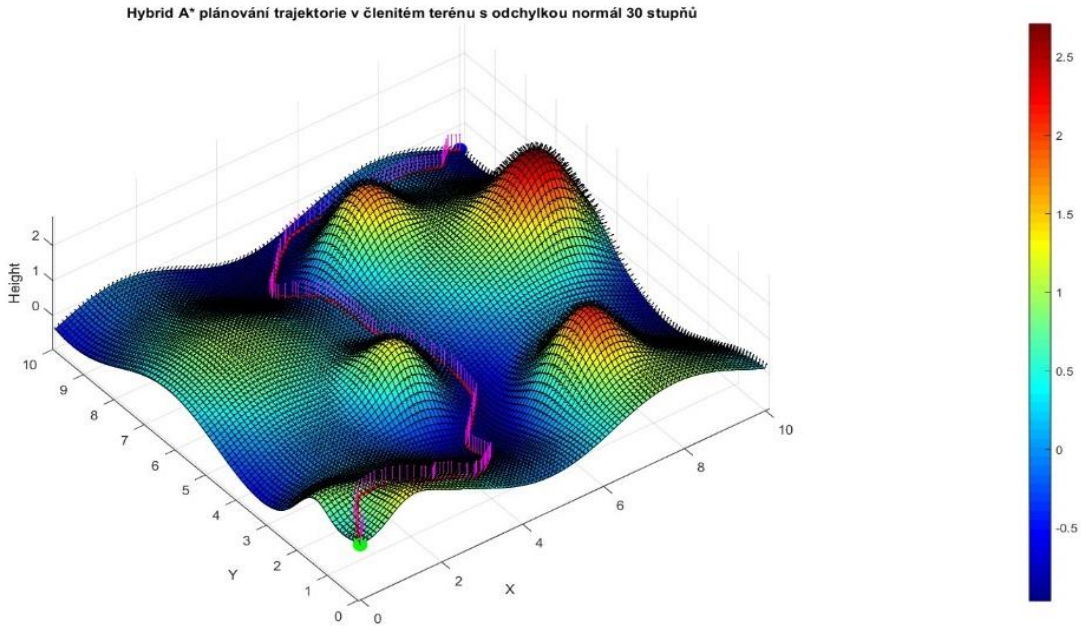
Výpočet úhlů

Úhly jsou vypočítávány pro každý bod na trajektorii pomocí skalárního součinu mezi normálou povrchu a vektorem orientovaným ve směru Z-osy. Tento úhel je poté uložen do matice angles, která obsahuje úhel sklonu pro každý bod na trase.

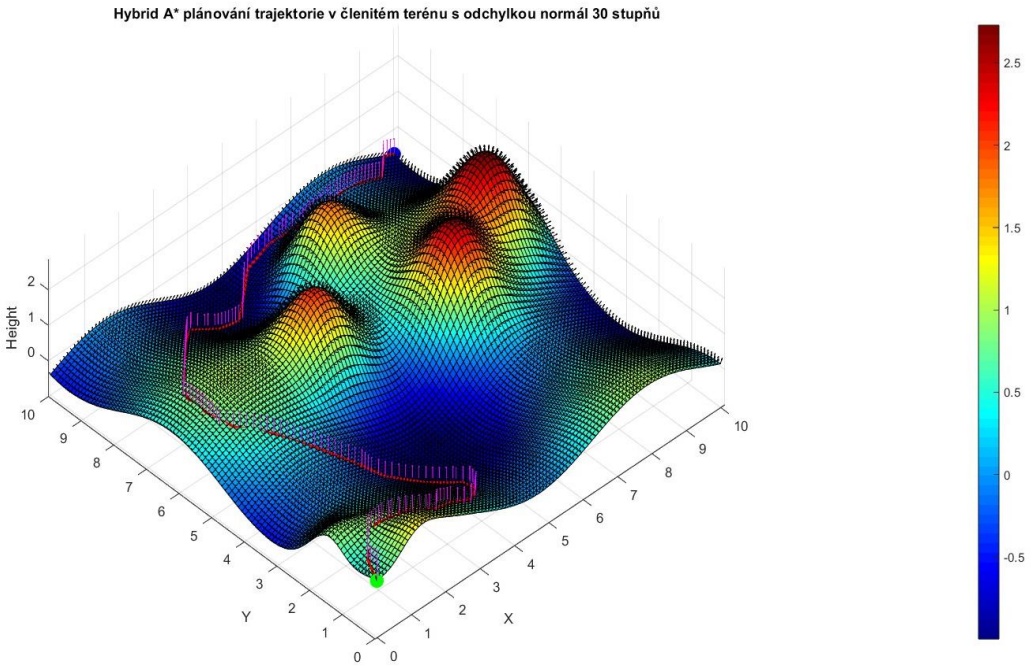
Vizualizace úhlů sklonu

Vykreslení úhlů na trase umožňuje analyzovat, zda nalezená trajektorie vyhovuje stanoveným kritériím. Úhly jsou také zobrazeny jako tabulka, což umožňuje jejich snadné prohlížení a analýzu.

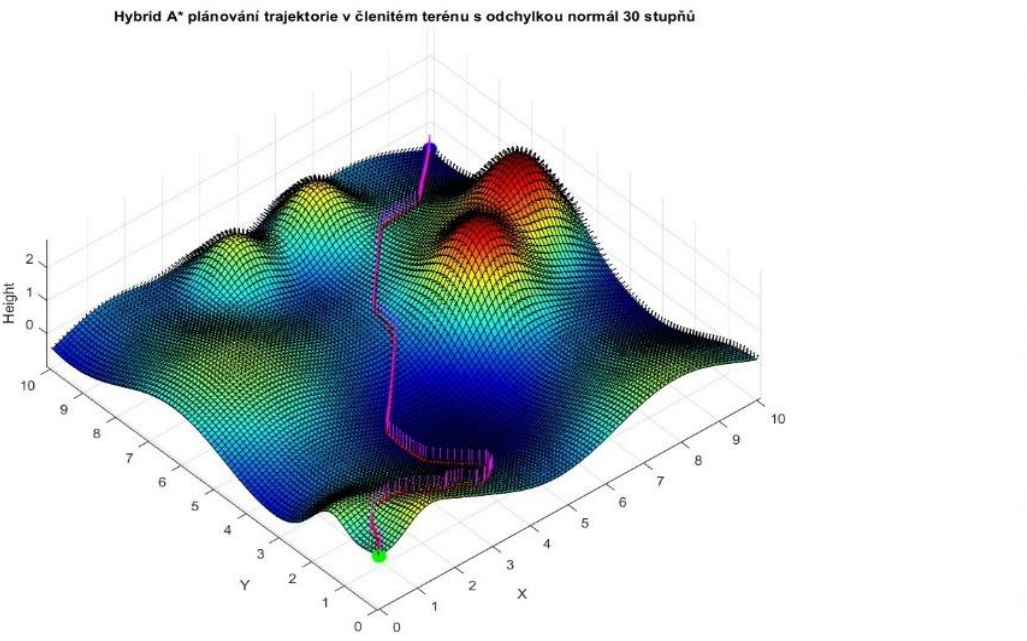
Obrázky přiložené níže, resp. Obr 8., Obr 9., Obr 10. jsou výsledkem daného programu, ve kterém byly upraveny parametry reprezentující daný členitý povrch. Tyto úpravy parametrů umožnily simulovat různé scénáře terénu, což pomohlo otestovat a optimalizovat algoritmus pro plánování trajektorie v různých podmínkách. Změny parametrů měly přímý vliv na tvar a složitost terénu, což vedlo k odlišným výzvám pro navigační algoritmus. Výsledné vizualizace tak poskytují cennou zpětnou vazbu o tom, jak dobře algoritmus zvládá navigaci ve složitých prostředích a jak se dokáže přizpůsobit různým topografickým charakteristikám.



Obr 8. - Ukázka plánovače trajektorie A* algoritmu varianta 1



Obr 9. - Ukázka plánovače trajektorie A* algoritmu varianta 3



Obr 10. - Ukázka plánovače trajektorie A* algoritmu varianta 2

5.1.2. Plánovač – RTT algoritmus

V rámci předchozího sledovače byl vytvořen ještě plánovač fungující na bázi RRT (Rapidly-exploring Random Tree) algoritmu za účelem grafického znázornění a porovnání metod.

Pseudokód

Vstupy:

- X, Y, Z: Mřížka bodů s hodnotami terénu, kde Z představuje výškové hodnoty.
- start: Startovní bod definovaný jako [x, y].
- goal: Cílový bod definovaný jako [x, y].
- max_iter: Maximální počet iterací, po které bude algoritmus běžet.
- step_size: Maximální vzdálenost, o kterou se strom může rozšířit v jednom kroku.
- max_slope_angle: Maximální přípustný úhel sklonu mezi sousedními body.
- max_turn_angle: Maximální přípustný úhel zatáčení mezi směry k novým bodům.
- max_height: Maximální přípustný výškový rozdíl mezi sousedními body.

Výstupy:

- path: Nalezená cesta z start do goal.

Kroky:

1. **Inicializace:**
   - Vytvoř prázdný strom RRT, který obsahuje pouze startovní bod [start, start_z, -1]. Zde start_z je výška startovního bodu a -1 označuje, že start nemá rodičovský uzel.
   - Nastav proměnnou goal_reached na false.

2. **Příprava cílového bodu:**
   - Ověř, zda je cílový bod v rámci hranic mřížky, a vypočítej jeho výšku goal_z z mřížky Z.

3. **Hlavní smyčka** (běžící po max_iter iterací nebo dokud není dosažen cíl):
   - Pro každou iteraci:

- Vzorkování bodu:
  - S pravděpodobností např.30 % vyber cílový bod jako vzor, jinak vyber náhodný bod v rámci mřížky.
- Najdi nejbližší uzel:
  - Prohledávej strom RRT a najdi uzel, který je nejblíže vzorovému bodu.
- Výpočet směru:
  - Vypočítej směr od nejbližšího uzlu k vzorovému bodu.
  - Pokud je vzdálenost větší než step_size, zkraťˇ směr na step_size.
- Vytvoření nové pozice:
  - Vytvoř novou pozici posunutím od nejbližšího uzlu ve vypočítaném směru.
  - Ujisti se, že nová pozice je v rámci hranic mřížky.
- Získání výšky nové pozice:
  - Zjisti výšku nové pozice z mřížky Z.
- Kontrola podmínek:
  - Vypočítej sklon mezi nejbližším uzlem a novou pozicí. Pokud je sklon větší než max_slope_angle, iteraci ukonči.
  - Ověř, zda je výška nové pozice menší než max_height nad výškou nejbližšího uzlu. Pokud ne, iteraci ukonči.
  - Vypočítej úhel mezi směrem k novému bodu a směrem z předchozího bodu k nejbližšímu uzlu. Pokud je úhel větší než max_turn_angle, iteraci ukonči.
- Přidání nové pozice do stromu:
  - Pokud jsou všechny podmínky splněny, přidej novou pozici do stromu RRT a ulož její rodičovský uzel.
- Kontrola dosažení cíle:
  - Pokud je nová pozice dostatečně blízko cílovému bodu, přidej cílový bod do stromu, nastav goal_reached na true a ukonči smyčku.

4. **Rekonstrukce cesty:**
    - Pokud goal_reached je true, rekonstruuj cestu z cílového bodu zpět k počátečnímu bodu pomocí rodičovských uzlů.
    - Pokud goal_reached je false, vyvolej chybu "Cesta nenalezena".

5. **Výstup:**
    - Vrať nalezenou cestu path jako seznam bodů od startovního bodu k cílovému bodu.

Popis vytvořeného kód v Matlabu

1. Funkce main: Inicializace prostředí a parametrů

- **Nastavení výchozího stavu generátoru náhodných čísel:** Prvním krokem je nastavení výchozího stavu generátoru náhodných čísel (rng(42)), což zajišťuje, že výsledky budou při opakovaném spuštění kódu stejné. Tento krok je klíčový pro reprodukovatelnost výsledků, protože RRT algoritmus využívá náhodné vzorkování bodů v prostoru, a tím ovlivňuje průběh generování trajektorie.
- **Generování mřížky a výpočet povrchu:** Vytváří se jemná mřížka X a Y pomocí funkce meshgrid, která generuje hodnoty mezi 1 a 100 s rozlišením 100 bodů. Hodnoty povrchu Z jsou poté vypočítány jako kombinace sinusových a kosinusových funkcí, funkce peaks a několika exponenciálních funkcí, které přidávají výrazné „kopce“ na povrch. Tento krok simuluje členitý terén, který bude použit pro plánování trajektorie.
- **Nastavení startovního a cílového bodu:** Startovní bod je definován jako [1, 1] (levý dolní roh mřížky) a cílový bod jako [100, 100] (pravý horní roh mřížky).
- **Parametry RRT:** Jsou definovány klíčové parametry RRT algoritmu, včetně maximálního počtu iterací (max_iter = 10000), velikosti kroku (step_size = 1), maximálního povoleného sklonu (max_slope_angle = 30 stupňů), maximálního úhlu zatáčky (max_turn_angle = 40 stupňů) a maximální povolené výšky (max_height = 1.5 jednotky).

2. Funkce rrt_surface: Implementace RRT algoritmu

- **Inicializace stromu:** Algoritmus začíná s počátečním uzlem, který obsahuje souřadnice startovního bodu a jeho výšku z povrchu Z. Strom je uložen v proměnné tree.
- **Iterace algoritmu:** Pro každý krok až do max_iter:
    - **Vzorkování bodů:** S 30 % pravděpodobností je vzorkován cílový bod, což pomáhá rychlejšímu sbližování s cílem. Jinak je vzorkován náhodný bod na mřížce.
    - **Vyhledání nejbližšího uzlu:** Pomocí funkce find_nearest je nalezen nejbližší uzel ve stromu k náhodně vzorkovanému bodu.
    - **Výpočet nového bodu:** Vypočítá se směr a vzdálenost k náhodně vzorkovanému bodu, přičemž se zajišťuje, že krok nepřekročí zadanou velikost kroku. Poté se určí nový bod na základě vypočítaného směru.
    - **Kontrola podmínek:** Nový bod je přidán do stromu pouze tehdy, pokud splňuje všechny podmínky – sklon mezi novým bodem a nejbližším uzlem nesmí být větší než max_slope_angle, výška nového bodu nesmí být větší než max_height a úhel zatáčení mezi směry nesmí být větší než max_turn_angle.
    - **Kontrola dosažení cíle:** Pokud je nový bod dostatečně blízko cíli, je cílový bod přidán do stromu a algoritmus končí.

3. Funkce find_nearest: Vyhledání nejbližšího uzlu ve stromu

Tato funkce prohledá celý strom a najde uzel, který je nejblíže k zadanému bodu. Používá se k určení, kam se strom bude dále rozšiřovat.

4. Funkce reconstruct_path: Rekonstrukce trajektorie

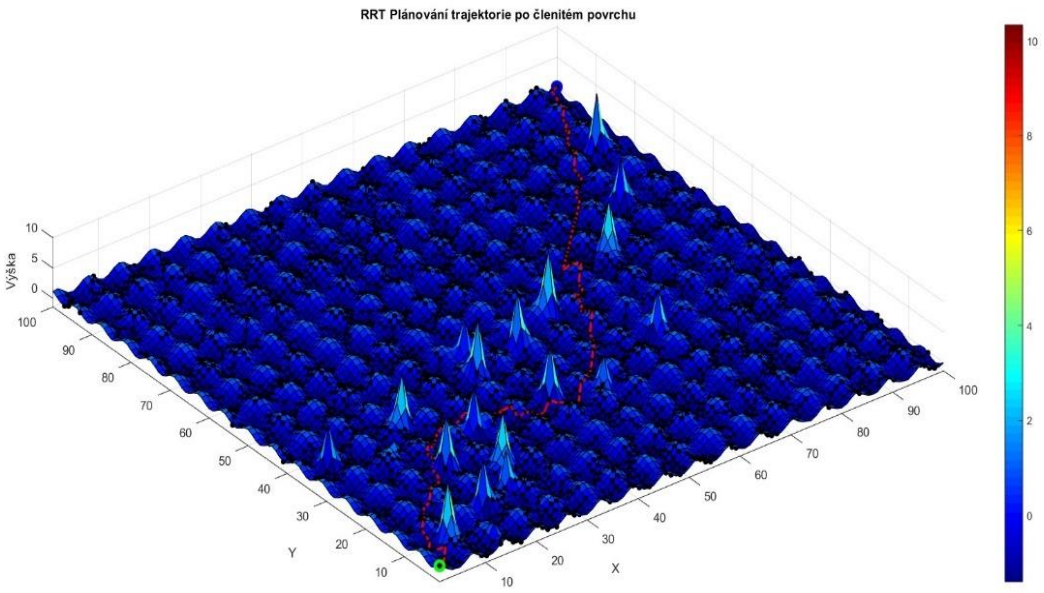
Jakmile algoritmus dosáhne cílového bodu, tato funkce rekonstruuje trajektorii zpětným procházením stromu od cíle ke startu. Každý uzel obsahuje odkaz na svého rodiče, což umožňuje zpětnou rekonstrukci cesty.

5. Funkce peaks: Generování členitého povrchu

Tato funkce vytváří výchozí povrch se složitými terénními prvky pomocí matematických funkcí. Povrch obsahuje různé vrcholy a údolí, které simulují realistický terén.

6. Vizualizace výsledků

Po úspěšném nalezení trajektorie kód vizualizuje terén pomocí 3D grafu. Zobrazí se startovní a cílový bod, strom RRT a nalezená trajektorie. Tato vizualizace poskytuje užitečný pohled na to, jak algoritmus pracoval a jakým způsobem se trajektorie přizpůsobila členitému terénu.



Obr 11. - Ukázka plánovače trajektorie RTT algoritmu varianta 1

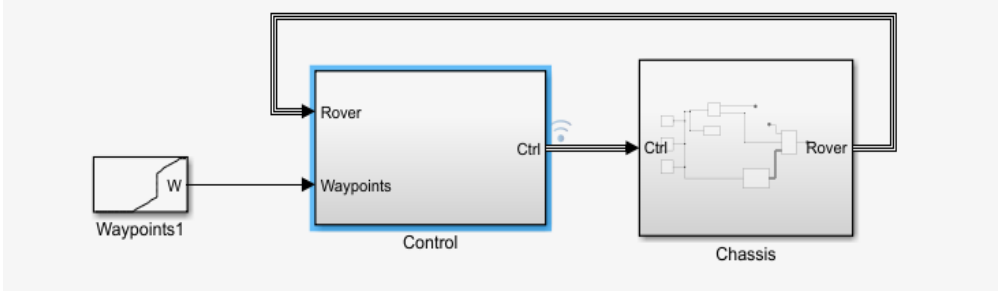
6. Simulační experiment

Po dokončení vývoje plánovače trajektorie v Matlabu bylo provedeno několik simulačních experimentů, jejichž cílem bylo ověřit správnost a efektivitu generované trajektorie. Tyto experimenty zahrnovaly simulaci pohybu vozidla v prostředí Simulink, jehož plánovač trajektorie byl testován pro rozdílné povrchy. V

tomto textu jsou popsány detaily tohoto experimentu, včetně procesu tvorby modelu vozidla, generování trajektorie a samotné simulace v prostředí Simulink.

Výběr simulačního prostředí

Pro simulační experiment bylo zvoleno prostředí Simulink, konkrétně jeho modul Simscape Multibody. Toto prostředí umožňuje simulovat a vizualizovat mechanické systémy včetně pohybu více těles, což se může zdát relativně ideální pro simulaci jízdy vozidla s více koly, jako je příslušné testované šestikolové vozidlo. Výběr Simulinku byl motivován především zkušenostmi získanými během bakalářského studia, kdy jsme s tímto prostředím v několika kurzech.. Simscape Multibody poskytuje robustní nástroje pro modelování fyzikálních systémů, což z něj činí výbornou volbu pro tento typ simulace.



Obr 12. – Vytvořené blokové rozhraní v Simulinku – převzato z [32]

Struktura a rozdělení experiment

Aby bylo možné provést simulaci v prostředí Simulink, bylo nutné tvorbu experimentu rozdělit na několik klíčových částí:

1. Fyzikální model vozidla

První částí projektu bylo vytvoření zjednodušeného modelu vozidla. Tento model zahrnoval základní konstrukci vozidla, jeho pohonné jednotky a základní parametry zejména rozměry a rozložení hmoty. Model byl vytvořen tak, aby co nejlépe odpovídal reálným podmínkám, ačkoliv zjednodušený model neobsahoval všechny detaily reálného vozidla.

2. Tvorba povrchu

Druhou klíčovou částí bylo vytvoření modelu povrchu, po kterém se vozidlo bude pohybovat. Tento model zahrnoval různé typy terénu s různými vlastnostmi, jako jsou svahy, nerovnosti a překážky. Povrch byl navržen tak, aby umožňoval testování vozidla v různých podmínkách, což zahrnovalo jak jednoduché, tak složitější terénní profily. Simulace byla navržena tak, aby bylo možné měnit parametry povrchu a tím testovat různé scénáře.

3. Plánovač trajektorie

Třetí a zároveň nejdůležitější částí experimentu bylo vytvoření plánovače trajektorie. Plánovač byl zodpovědný za generování souřadnic bodů trajektorie, po které se vozidlo mělo pohybovat. Tento plánovač musel být schopen vzít v úvahu různé parametry povrchu, aby byla trajektorie co nejefektivnější a nejpřesnější. Plánovač byl navržen tak, aby byl flexibilní a umožňoval přizpůsobení trajektorie různým podmínkám. Důraz byl kladen na to, aby generovaná trajektorie byla realistická a dosažitelná pro model vozidla.

4. Sledovač trajektorie

Poslední částí projektu bylo vytvoření sledovače trajektorie, který měl za úkol kontrolovat, zda vozidlo sleduje generovanou trajektorii. Tento sledovač byl klíčovým prvkem simulace, protože umožňoval ověřit, zda vozidlo správně reaguje na plánovanou trajektorii, a to i při změnách v terénu. Sledovač byl navržen tak, aby mohl monitorovat odchylky od trajektorie a poskytovat zpětnou vazbu pro další úpravy plánovače.

Většina simulačního experimentu byla inspirována modelem Mars Roveru, který je dostupný na internetu. Tento model poskytoval komplexní autonomní jízdu s funkcemi jako odebrání vzorku horniny na konci jízdy a také kinematickou analýzu jízdních vlastností vozidla. Vzhledem k tomu, že daná úloha v rámci bakalářské práce byla specifická a méně komplexní, bylo rozhodnuto model Mars Roveru zjednodušit a přizpůsobit potřebám experimentu.

Konkrétně z tohoto modelu byl převzat sledovač trajektorie, který se osvědčil jako velmi kvalitní. Samotná práce spočívala v zjednodušení modelu vozidla a

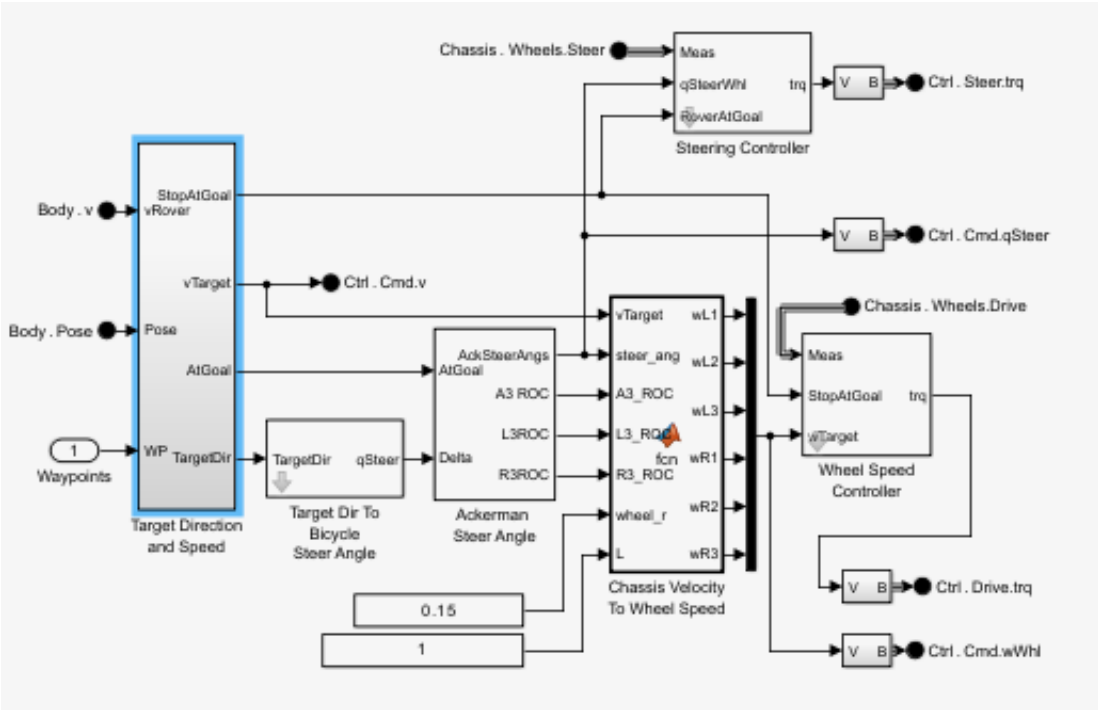
vylepšení generování trajektorie tak, aby bylo možné generovat trajektorii pro různé typy povrchů. Tímto způsobem byla snaha zajistit, jestli vytvořený model bude dostatečně flexibilní a přizpůsobitelný různým podmínkám, které mohou nastat během simulace

Hlavní cíle a zaměření experimentu

Hlavním cílem experimentu bylo ověřit generování trajektorie a následné sledování vozidlem této trajektorie. Experiment byl navržen tak, aby se soustředil na tyto aspekty, zatímco analýza dynamických a kinematických vlastností podvozku

vozidla nebyla prioritou. Bylo důležité zajistit, aby generovaná trajektorie byla realistická a aby vozidlo bylo schopno tuto trajektorii co nejpřesněji sledovat.

Experiment byl navržen tak, aby simuloval různé scénáře a umožňoval testování vozidla v různých podmínkách. Tento přístup umožnil získat podrobný přehled o tom, jak dobře plánovač trajektorie funguje a jak vozidlo reaguje na různé typy povrchů.



Obr 13. – Blokové rozhrání sledovače trajektorie s PID regulátory a ovládací Ackermannovou kinematikou – převzato z [33]

Implementace experimentu

Pro samotný experiment byly vytvořeny tři hlavní skripty v Matlabu, které byly použity pro různé části simulace. Tyto skripty byly následující:

1. **Načtení parametrů vozidla**: První skript byl navržen tak, aby načetl všechny potřebné parametry vozidla, jako jsou rozměry a vlastnosti pohonného systému a další klíčové parametry. Tyto parametry byly následně uloženy do workspace Matlabu, kde s nimi mohl pracovat hlavní program v Simulinku.
2. **Generování trajektorie bodů**: Druhý skript byl zodpovědný za generování bodů trajektorie, které vozidlo mělo sledovat. Tento skript používal plánovač trajektorie, který byl navržen tak, aby bral v úvahu různé parametry povrchu a generoval realistickou a efektivní trajektorii.
3. **Generování povrchu**: Třetí skript byl použit pro generování povrchu, po kterém se vozidlo mělo pohybovat. Tento skript umožňoval vytváření různých typů terénu s různými vlastnostmi, což umožňovalo testování vozidla v různých scénářích.

Po spuštění těchto skriptů byly všechny proměnné načteny do workspace Matlabu, kde s nimi mohl hlavní program v Simulinku pracovat. Tento přístup umožnil oddělit jednotlivé části simulace a zajistit, že každý krok byl proveden správně a efektivně.

6.1. Modelování prostoru pro prověření plánovače trajektorie

Pro modelování prostoru, ve kterém měla být trajektorie testována, byl využit blok zvaný Grid Surface dostupný v prostředí Simulink. Tento blok umožňuje parametricky vytvořit rovinný či nepravidelný povrch v prostoru, což se ukázalo jako efektivní řešení pro simulovaný experiment. Výběr tohoto postupu byl veden především snahou o maximalizaci efektivity programování a snadnou přizpůsobitelnost modelu pro různé scénáře.

Blok Grid Surface v Simulinku umožnil vytvořit mřížkovaný povrch, který bylo možné parametricky upravovat podle potřeb experimentu. Pro každý bod na mřížce bylo možné definovat specifické hodnoty, které reprezentovaly například výšku terénu v daném bodě. Tento přístup nám poskytl značnou flexibilitu při modelování

různých typů povrchů, což bylo zásadní pro testování trajektorie vozidla v různých podmínkách.

Výhody skriptu v Matlabu

Kromě samotného bloku Grid Surface byl pro jeho řízení a úpravy vytvořen skript v Matlabu. Tento skript měl několik klíčových funkcí:

1. **Parametrizace povrchu**: Skript umožňoval definovat povrchové parametry, jako jsou velikost mřížky, rozsah souřadnic, a co je nejdůležitější, tvar a profil povrchu. Díky tomuto skriptu jsme byli schopni rychle měnit topografii simulovaného terénu podle požadavků experimentu.

2. **Generování a načítání proměnných**:. Tyto proměnné zahrnovaly informace o povrchu, které byly následně použity v hlavním simulačním programu v Simulinku. Tento krok zajistil, že všechny potřebné údaje byly dostupné pro simulaci a mohly být snadno měněny bez nutnosti zásahu do hlavního simulačního modelu.

3. **Konzistence mezi prostředími**: Vytvoření modelu prostoru pomocí stejných matematických funkcí jako v Matlabu zajistilo, že přechod mezi různými prostředími byl konzistentní. Tím bylo zajištěno, že simulované prostředí odpovídá tomu, co bylo navrženo a vypočteno v Matlabu, což bylo klíčové pro dosažení přesných a spolehlivých výsledků.

Proces modelování prostoru

Prvním krokem při simulaci bylo tedy vytvoření modelu prostoru, ve kterém bude trajektorie testována. Prostor byl navržen jako trojrozměrná mřížka, kde každá buňka reprezentovala specifický bod terénu s určenou výškou. Tento model mřížky umožňoval přesné zobrazení terénu a byl dostatečně flexibilní, aby mohl simulovat různé typy povrchů, od rovinných až po složitě tvarované terény s nerovnostmi.
Aby bylo zajištěno, že tento model bude správně fungovat v simulaci, bylo důležité přesně definovat matematické funkce, které určovaly tvar a výšku povrchu. Byly použity různé trigonometrické a polynomické funkce k vytvoření realistických terénních profilů, které mohly zahrnovat kopce, údolí a jiné topografické prvky. Tyto funkce byly poté implementovány do skriptu v Matlabu, který generoval odpovídající datovou matici pro každý bod mřížky.

Implementace do Simulinku

Jakmile byl prostor modelován a proměnné vygenerovány, byly tyto údaje importovány do hlavního programu v Simulinku. Implementace modelu prostoru v Simulinku umožnila vizualizaci a simulaci pohybu vozidla přes tento povrch. V rámci Simulinku byly data mřížky použity jako vstupní parametry pro blok Grid Surface, který následně generoval odpovídající vizualizaci terénu.

Tento přístup nám umožnil testovat, jak vozidlo reaguje na různé typy povrchů a zda je schopno sledovat generovanou trajektorii i při změnách v terénu.

Efektivita a flexibilita vytvořeného modelu

Jednou z hlavních výhod využití bloku Grid Surface ve spojení se skriptem v Matlabu byla možnost rychlých a efektivních úprav modelu prostoru. Díky skriptu jsme mohli snadno změnit parametry povrchu a generovat nové trajektorie, které odpovídaly změnám v terénu. Tento postup nám poskytl možnost testovat různé scénáře bez nutnosti složitého přepracovávání celého modelu.
Navíc, použití parametrických funkcí a maticových operací v Matlabu umožnilo přesné modelování a generování terénu s minimálním počtem chyb. Tato metodika zjednodušila celý proces simulace a umožnila nám soustředit se na hlavní cíle experimentu, tedy na ověření generované trajektorie a sledování jejího průběhu.

V budoucích experimentech by mohlo být užitečné dále zkoumat možnosti optimalizace modelu povrchu a zlepšení algoritmů pro generování trajektorie, aby bylo možné dosáhnout ještě přesnějších a spolehlivějších výsledků. Možnosti jsou téměř neomezené a s dalším pokrokem v simulačních technologiích můžeme očekávat stále sofistikovanější a efektivnější řešení.

6.2. Zjednodušený prototyp šestikolového vozidla

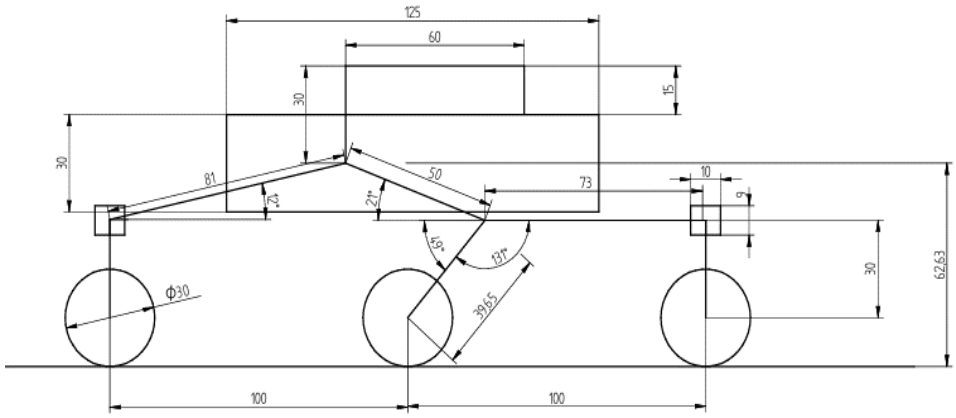
Pro simulaci byl vytvořen zjednodušený prototyp šestikolového vozidla, který byl navržen tak, aby co nejvěrněji napodoboval reálný model roveru NASA Perseverance. Inspirace tímto roverem byla zvolena kvůli jeho aktuálnímu technologickému pokroku a významnému vlivu na současné trendy v oblasti vesmírných vozidel. Prototyp byl navržen s důrazem na schopnost projetí terénem, nikoliv na kinematickou a dynamickou analýzu

Konstrukce a parametry vozidla

Šestikolový rover byl navržen s ohledem na specifické potřeby simulovaného experimentu, zejména funkčnost generátoru trajektorie. Při návrhu tohoto roveru nebyly kladeny vysoké nároky na přesné rozměry vozidla, spíše se soustředilo na funkčnost a schopnost vozítka pohybovat se efektivně bez prokluzu.

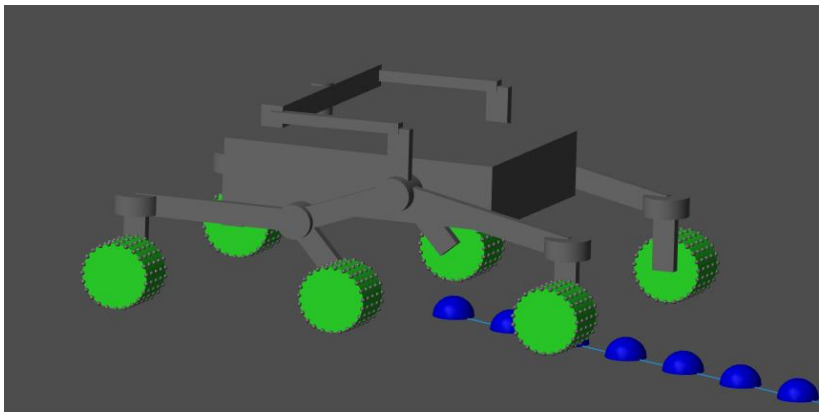
Klíčovým aspektem při návrhu tohoto roveru bylo dosažení optimálního rozložení hmoty, jenž je kritické pro zajištění stability a efektivity pohybu vozítka, zejména na nerovném terénu.

Při návrhu rozměrů tohoto roveru bylo zohledněno především dosažení dobré pohyblivosti. Rozměry byly optimalizovány tak, aby bylo dosaženo adekvátní prezentace funkčnosti plánovače trajektorie. Kromě toho bylo důležité zajistit, aby rozměry neomezovaly schopnost vozítka překonávat různé terénní překážky. Tímto způsobem byl navržen rover, který může efektivně manévrovat v simulovaných podmínkách.



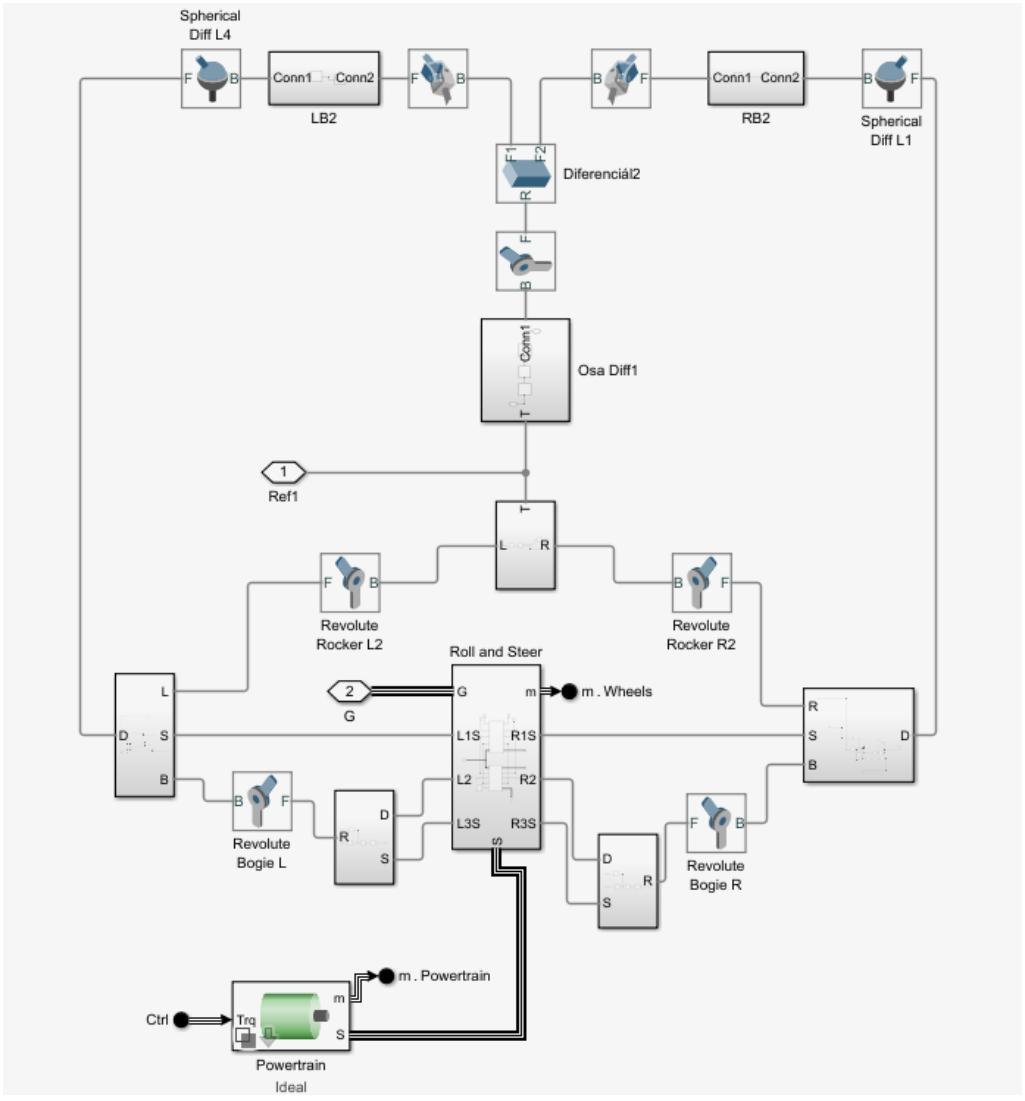
Obr. 14 - Rozměry vozidla v cm

Každé z šesti kol roveru je vybaveno nezávislým pohonem, což umožňuje vozítku přesně kontrolovat každý pohyb a zajišťuje maximální trakci na povrchu. Ve čtyřech kloubech jsou ještě umístěny další nezavislé pohony pro zatáčení Tento systém pohonu je klíčový pro dosažení plynulého pohybu bez prokluzu.



.

Obr 15. – Prototyp šestikolového vozítka



Obr 16. – Blokové rozhraní v Multibody Simulnik převzato z [33]

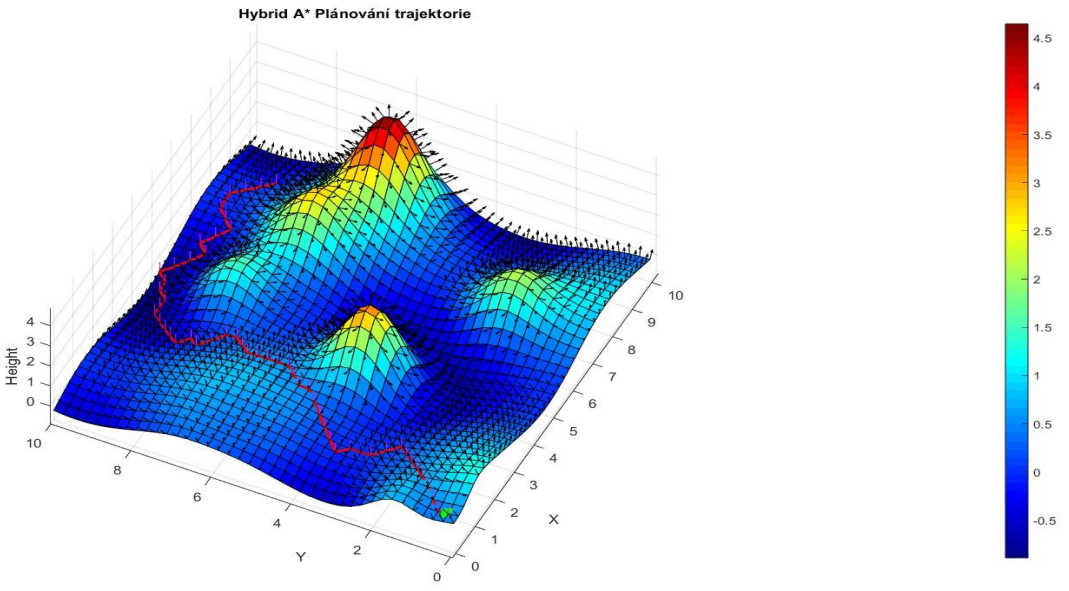
Implementace v Matlabu a Simulinku

Prototyp byl podpořen skriptem vytvořeným v Matlabu, který automaticky načítal hodnoty pro klíčové parametry vozidla. Tento skript umožňoval snadné úpravy vlastností vozidla. Díky tomuto skriptu jsme byli schopni rychle přizpůsobit model vozidla specifickým požadavkům experimentu, aniž bychom museli manuálně měnit nastavení v Simulinku.

Celkově byl model vozidla navržen tak, aby byl dostatečně jednoduchý na implementaci pro plánovač trajektorie, ale zároveň dostatečně komplexní, aby poskytl realistické výsledky simulace.

6.3. Plánovač trajektorie

Pro plánování trajektorie byly vytvořeny dva různé skripty, z nichž každý využívá odlišný algoritmus: *A (A-star)** algoritmus a **Rapidly-exploring Random Tree (RRT)** algoritmus. Oba algoritmy mají své specifické výhody a nevýhody, které je činí vhodnými pro různé typy simulačních úloh.



Obr 17. – Grafické uplatnění plánovače trajektorie varianta 1

Výběr a implementace A* algoritmu

Pro plánovač trajektorie byl využít skript založený na A* algoritmu. Tento algoritmus byl vybrán zejména kvůli své nižší výpočetní náročnosti ve srovnání s

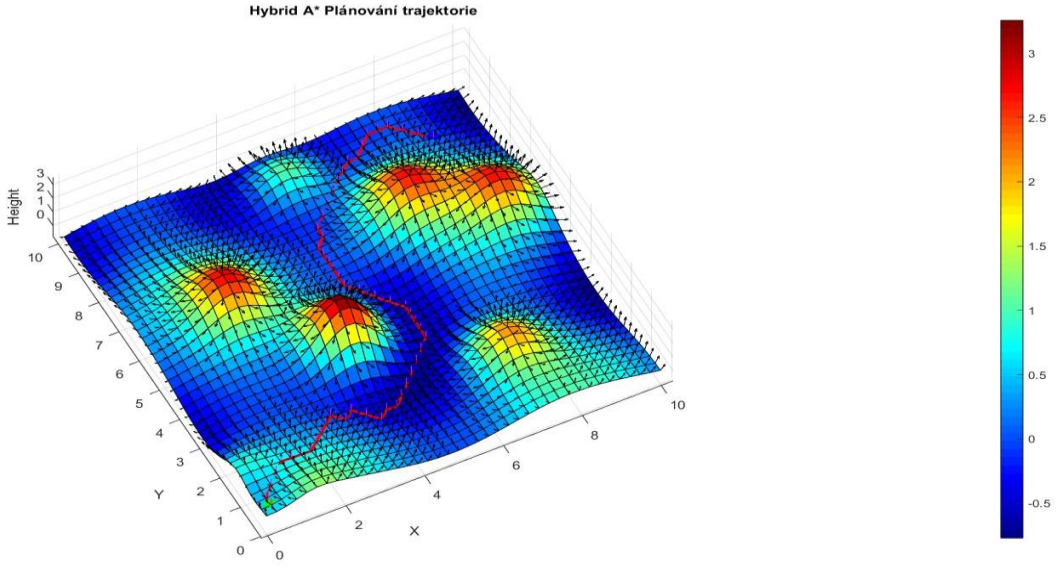
RRT algoritmem a pro svou schopnost efektivně nalézt optimální cestu v prostoru s předem definovanými překážkami. A* algoritmus je známý svou schopností najít nejkratší možnou cestu mezi dvěma body, přičemž zohledňuje jak vzdálenost, tak náklady spojené s překonáváním překážek.

Jedním z klíčových aspektů, které byly u tohoto algoritmu využity, bylo generování trajektorie s ohledem na normály povrchu. Normály byly použity k úpravě plánované trajektorie tak, aby byla realistická a odpovídala reálným fyzikálním podmínkám vozidla, jako je sklony terénu a překážky.

Důvody pro výběr A* algoritmu

A* algoritmus byl zvolen nejen kvůli své efektivitě, ale také kvůli své jasně definované struktuře, která umožňuje snadnou implementaci a kontrolu nad generováním trajektorie. Na rozdíl od RRT algoritmu, který může generovat trajektorie více náhodným a méně předvídatelným způsobem, A* algoritmus poskytuje deterministické výsledky, což je v mnoha simulacích žádoucí. Jeho deterministický charakter zajišťuje, že každý běh algoritmu s identickými vstupy povede ke stejnému výsledku, což usnadňuje ladění a analýzu výsledků.

V rámci simulačního experimentu byla tato volba klíčová pro dosažení spolehlivých a reprodukovatelných výsledků. A* algoritmus nám poskytl flexibilitu při práci s

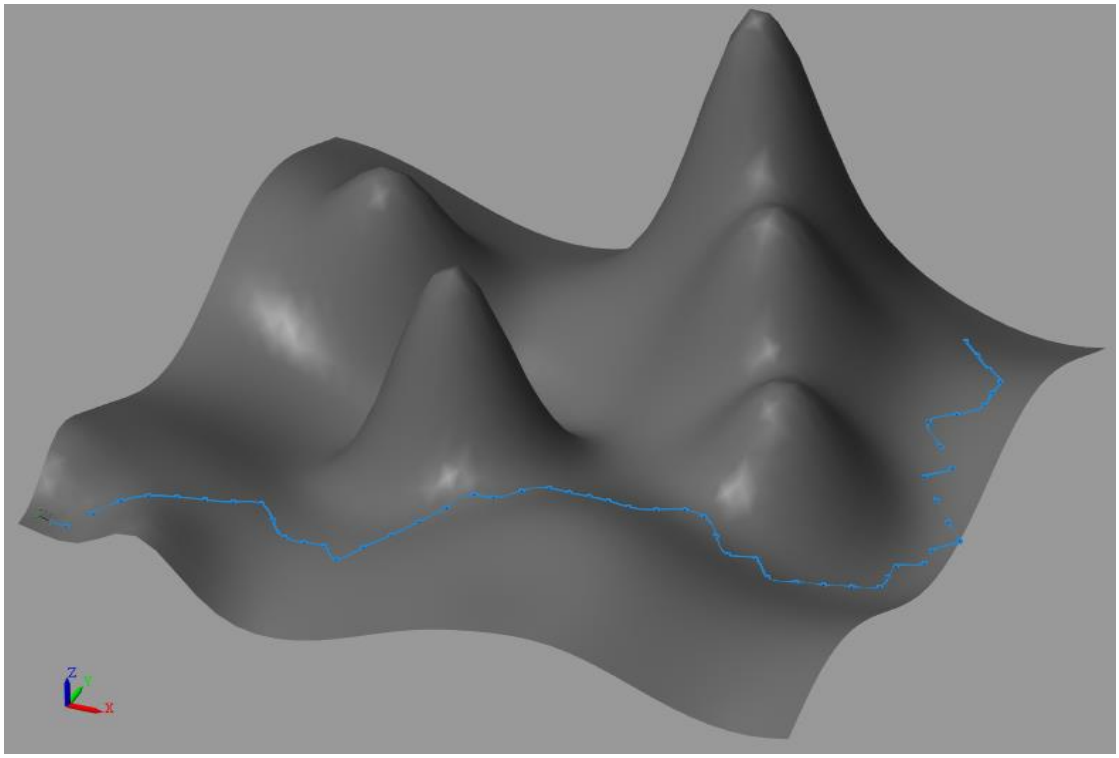


Obr 18. – Grafické uplatnění plánovače trajektorie varianta 2

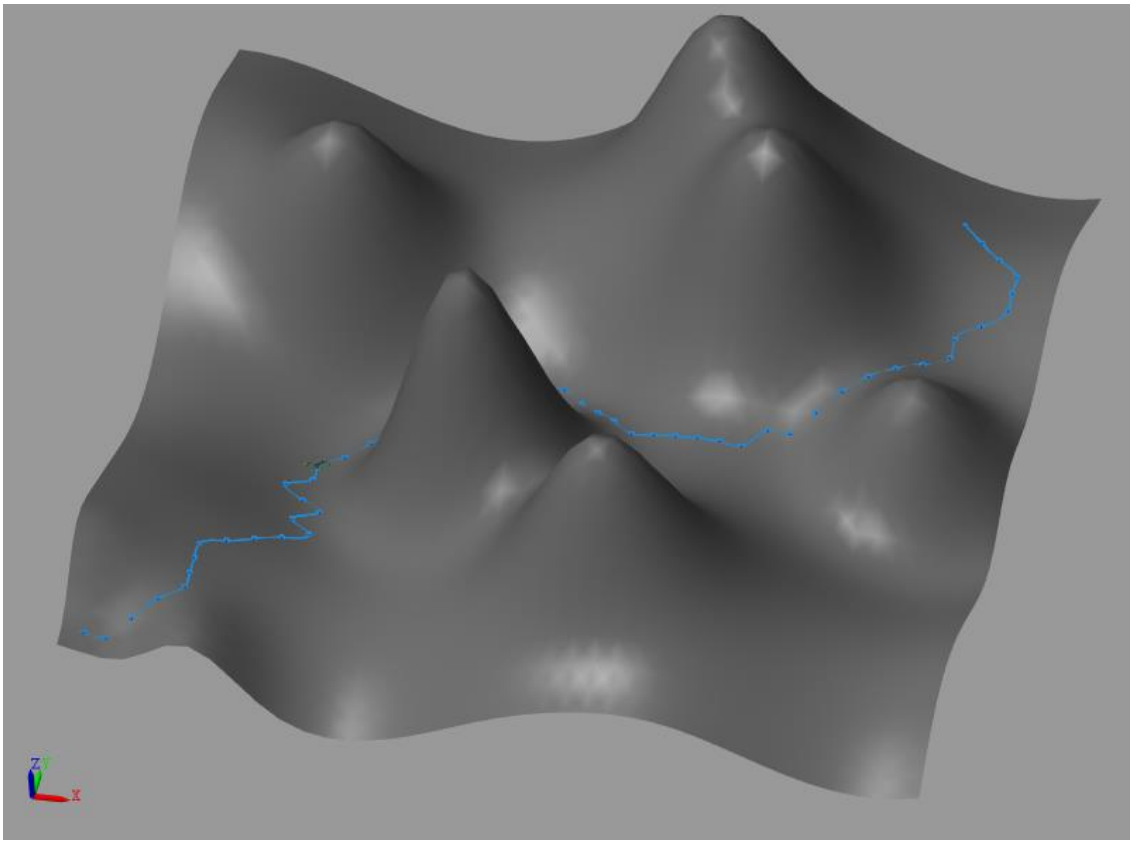
různými typy terénů a překážek, přičemž zůstal výpočetně zvládnutelný, což bylo obzvláště důležité vzhledem k omezením simulace v reálném čase. Celkově nám A* algoritmus umožnil efektivně generovat trajektorie, které splňovaly technické požadavky na optimalizaci trasy.

6.4. Samotná simulace – ověření plánovače

Funkčnost plánovače trajektorie byla relativně úspěšně ověřena při výpočtu dráhy pro šestikolový rover. Plánovač efektivně zohledňuje různé terénní podmínky a optimalizuje cestu tak, aby minimalizoval energetické nároky a maximalizoval efektivitu pohybu roveru. Díky této technologii je možné přesně naplánovat trasu, která zohlední všechny klíčové faktory, jako jsou sklon terénu, překážky a povrchová struktura. Výsledkem je robustní plánovací systém, který zajišťuje bezpečnou a efektivní navigaci roveru v neznámém terénu.



Obr 19. – Znázornění simulačního experimentu pro variantu 1



Obr 20. – Znázornění simulačního experimentu pro variantu 2

7. Závěr

Závěrem lze konstatovat, že plánovač trajektorie vyvinutý v rámci této práce se ukázal jako účinný nástroj pro zajištění autonomního pohybu vozidla v náročných terénních podmínkách. Algoritmy A* a RRT byly úspěšně implementovány a testovány v programovatelném prostředí Matlab a následně A* algoritmus byl ověřen v simulačním prostředí, kde prokázal svou schopnost efektivně navigovat vozidlo přes různé typy terénu a překážek. Výsledky testování naznačují, že použití těchto algoritmů v reálném čase poskytuje flexibilní a robustní řešení pro plánování trajektorie, které je schopné přizpůsobit se dynamickým změnám prostředí.

Hlavní přínos této práce spočívá v její modularitě a možnostech dalšího rozšíření. Plánovač byl navržen tak, aby byl snadno přizpůsobitelný různým typům vozidel a senzorových systémů, což umožňuje jeho využití v různých aplikacích na dané téma. Výhodou je také robustnost algoritmu, který je schopen generovat

spolehlivé trajektorie i při neúplných nebo nepřesných vstupních datech, což je klíčové pro jeho použití v reálných podmínkách.

V rámci práce bylo zohledněno i použití Ackermannovy kinematiky, mimo práci byly vytvořené plánovače trajektorie doplněny o tuto podmínku pro plánováni. Nicméně, pro tento konkrétní šestikolový rover bylo rozhodnuto tuto metodu neaplikovat, protože vozidlo umožňuje prakticky neomezené manévry a je schopno se otočit na místě, což zajišťuje vysokou flexibilitu v pohybu i v omezených prostorech.

Dosažené výsledky potvrzují, že navržený plánovač trajektorie může být efektivně nasazen v autonomních systémech a představuje solidní základ pro další vývoj v oblasti robotické navigace a autonomního řízení. Doporučením pro další výzkum je zaměřit se na integraci pokročilejších algoritmů, jako je SLAM, pro navigaci v neznámém terénu a zlepšení odhadu dynamických překážek.

8. Literatura

[1] Han, Zhichao, et al. An Efficient Spatial-Temporal Trajectory Planner for Autonomous Vehicles in Unstructured Environments. arXiv preprint arXiv:2208.13160, 2022

[2] Xu, Long, et al. An Efficient Trajectory Planner for Car-Like Robots on Uneven Terrain. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[3] Elliott, Preston. A* (Star) Algorithm. Medium, 2019.

[4] Kang, Jin-Gu, et al. Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning. *Sensors*, 2021

[5] Shen, Congkai, et al. An Efficient Global Trajectory Planner for Highly Dynamical Nonholonomic Autonomous Vehicles on 3-D Terrains. *IEEE Transactions on Robotics*, roč. 40, vyd. 1, 2023.

[6] Dube, Raghav, et al. Pathfinding Visualizer: A Survey of the State-of-Art. In: *IOT with Smart Systems*. ICTIS 2023. Lecture Notes in Networks and Systems. Springer, 2023

[7] Meng, Xianchen a Xi Fang. *A UGV Path Planning Algorithm Based on Improved A with Improved Artificial Potential Field**. *Electronics*, 2024,

[8] Introduction to Variational Methods, MIT OpenCourseWare.

[9] Lanczos, Cornelius. The Variational Principles of Mechanics. Heritage. Toronto: University of Toronto Press, 1949.

[10] Barnett, V. D. Numerical Methods for Scientists and Engineers. *Royal Statistical Society. Journal. Series A: General*, roč. 125, č. 4, 1962,

[11] Kurfess, Thomas R., ed. Robotics and Automation Handbook. 1. vyd. Boca Raton: CRC Press, 2005.

[12] Triantafyllopoulos, K. Dynamic Systems and Control. In: *Bayesian Inference of State Space Models*. Springer, 2021.

[13] Wang, Huanwei, et al. *The EBS-A algorithm: An improved A* algorithm for path planning**. *PLOS ONE*, 2022.

[14] Haït, Alain, Thierry Simeon a Michel Taïx. Algorithms for rough terrain trajectory planning. *Advanced Robotics*, roč. 16, č. 8, 2002

[15] Howard, Thomas M. a Alonzo Kelly. Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots. *The International Journal of Robotics Research*, roč. 26, č. 2, 2007.

[16] Ma, Hao, Wenhui Pei a Qi Zhang. Research on Path Planning Algorithm for Driverless Vehicles. *Mathematics*, 2022, -RTT

[17] Xiang, Dan, Hanxi Lin, Jian Ouyang a Dan Huang. *Combined improved A and greedy algorithm for path planning of multi-objective mobile robot**. *Scientific Reports*, 2022,

[18] Campbell, Sean, et al. Path Planning Techniques for Mobile Robots: A Review. In: *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2020.

[19] LaValle, M. Rapidly-exploring random trees: A new tool for path planning. Technická zpráva. Iowa State University, Department of Computer Science, 1998.

[20] Karaman, Sertac a Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, roč. 30, č. 7, 2011.

[21] Palmieri, Luigi, Sven Koenig a Kai O. Arras. RRT-based nonholonomic motion planning using any-angle path biasing. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016. _KONEC

[22] Chochel, Michael. Plánování trajektorie mobilního robotu s diferenciálním podvozkem. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky, 2017. Bakalářská práce. Vedoucí práce doc. Ing. Jiří Krejsa, Ph.D.

[23] Hart, Peter E., Nils J. Nilsson a Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968,

[24] Zammit, Christian a Erik-Jan van Kampen. *Comparison Between A and RRT Algorithms for 3D UAV Path Planning*. *Unmanned Systems*, 2022,

[25] Siciliano, Bruno a Oussama Khatib, eds. Springer Handbook of Robotics. 2. vyd. Berlin: Springer, 2016.

[26] Borenstein, J., H. R. Everett, L. Feng a D. Wehe. Mobile robot positioning: Sensors and techniques. *Journal of Robotic Systems*, 1998.

[27] Gage, Douglas W. UGV HISTORY 101: A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts. *Unmanned Systems Magazine*,1995.

[28] Jazar, Reza N. Vehicle Dynamics: Theory and Application. 3. vyd. Berlin: Springer, 2017.

[29] Robert Bosch GmbH. Automotive Handbook. 9. vyd. Hoboken: Wiley, 2014.

[30] Rajamani, Rajesh. Vehicle Dynamics and Control. 2. vyd. Berlin: Springer, 2012. ISBN 9781461414322.

[31] Reimpell, Jornsen, Helmut Stoll a Jurgen Betzler. The Automotive Chassis: Engineering Principles. 2. vyd. Oxford: Butterworth-Heinemann, 2001

[32] MathWorks, Mars Rover Vehicle Model., MathWorks [online].

[33] Ellery, Alex. Planetary Rovers: Robotic Exploration of the Solar System. Berlin: Springer, 2016. ISBN 9783319099675.