České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Časo-prostorový plánovač letového plánu

Praha, 2009 **Autor: Jaromír Pufler**

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 21.5.2009

podpis

Poděkování

Chtěl bych vyjádřit poděkování panu Ing. Davidu Šišlákovi za vedení, poskytnutí materiálu a konzultace, které mi umožnily, aby byla práce úspěšně dokončena. Mé poděkování patří také mé rodině za podporu během celého studia.

Abstrakt

Tato práce se zabývá plánováním optimálního letového plánu s ohledem na časové a rychlostní omezení v prostředí s bezletovými zónami. Zkoumá původní metodu plánování, která plánuje zvlášť nejprve v prostoru a poté v čase. Navrhuje novou metodu, která plánuje v časo-prostoru, a obě metody porovnává a odhaluje nedostatky původního plánovače.

Návrh nové metody je motivován případy, kde původní metoda nefunguje z důvodu odděleného plánování v prostoru a čase. Takovým případem může být zavislost maximální rychlosti letadla na nadmořské výšce, nebo prostředí, ve kterém se vyskytují bezletové zóny s časovou platností. To je například předpovídané zhoršené počasí, které potrvá určitou dobu.

Rozšířením původního plánovače o dimenzi času, ale i rychlosti naroste prohledávaný prostor. Proto dále práce zkoumá, jaký to má dopad na rychlost nalezení letového plánu.

Časo-prostorový plánovač hledanou trasu zároveň penalizuje za vychýlení rychlosti od optimální, kde má letadlo nejnižší spotřebu. Tím se hledaná trasa přímo optimaluzuje tak, aby i spotřeba byla minimální.

Abstract

This thesis deals with optimal fly plan scheduling considering time and speed limits in an environment with no-fly zones. It examines the former planning method which plans separately in space, at first, and then in time. It comes with a new method which plans in time-space; finally, it compares both formentioned approaches.

The proposal of the new method is motivated by the cases where the original method does not work due to the separate planning in space and in time. The dependency of the maximum velocity on altitude or surroundings including no-fly zones with time validity can be one of these cases. It could be for example worsening weather that lasts for certain time.

The extension of the original planner of the time dimension and the velocity increases the scanned space. That is why the work examines the influence on the speed of the flight plan location.

Spatial-time planner penalizes the deviance from the optimal velocity with the lowest consumption. It optimises the searched route, so that even the consumption is on its lowest level.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Jaromír P u f l e r

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný

Obor: Kybernetika a měření, blok KM2 – Umělá inteligence

Název tématu: Časo-prostorový plánovač letového plánu

Pokyny pro vypracování:

1. Prostudujte různé přístupy v oblasti plánování trasy ve třídimenzionálním prostoru rozšířeném o časovou dimenzi se zaměřením na plánování letové dráhy včetně spojité rychlostní složky v prostředí s bezletovými zónami.
2. Seznamte se se systémem AgentFly a existujícím dvoufázovým přístupem k tomuto plánování, identifikujte nevýhody tohoto řešení.
3. Navrhněte změnu plánovače, tak aby byl schopen hledat optimální řešení i pro složitější optimalizační kritéria, kdy stávající řešení selhává.
4. Navržené řešení implementujte v prostředí AgentFly jako nový plánovací modul.
5. Ověřte, analyzujte a srovnejte implementovanou metodu proti existujícímu přístupu.

Seznam odborné literatury:

[1] LaValle, S.M.; Steven, M.: Planning Algorithms. Cambridge University Press, 2006.
[2] Šišlák, D.; Pěchouček, M.; Volf, P.; Pavlíček, D.; Samek, J.; Mařík, V.; Losiewicz, P.: Agentfly: Towards Multi-Agent Technology in Free Flight Air Traffic Control. Defence Industry Applications of Autonomous Agents and Multi-Agent Systems. Birkhauser Verlag, 2008.

Vedoucí diplomové práce: Ing. David Šišlák

Platnost zadání: do konce zimního semestru 2009/2010

prof. Ing. Vladimír Mařík, CSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 3. 9. 2008

Obsah

Seznam obrázků

Kapitola 1

Úvod

Cílem této práce je prozkoumat existující plánování trasy v systému AgentFly, které probíhá zvlášť v prostoru a poté v čase, a pak navrhnout změnu plánovače tak, aby při generování plánu uvažoval rovnou i čas. Dále práce rozebírá nedostatky původního přístupu k plánování a obě metody mezi sebou porovnává. Návrh nového plánovače je motivován složitějšími optimalizačními kritérii, které původní planovač není schopen řešit. Příkladem může být prostředí, ve kterém se vyskytují bezletové zóny pouze s časovou platností, nebo případ, kde maximální rychlost závisí na nadmořské výšce.

Ve zbytku této kapitoly jsou popsány hlavní prvky systému AgentFly potřebné pro plánování. Následující kapitola je věnována stávajícímu plánování trasy, z něhož nové plánování vychází. V kapitole 3 je popsána nová metoda časo-prostorového plánování. Kapitola 4 je věnována implementaci a v kapitole 5 je časo-prostorový plánovač porovnán s původním plánovačem. Na konec je shrnutí celé problematiky.

1.1 Systém AgentFly

AgentFly je multiagentní systém určený k simulaci letu letadla, plánování trasy a zabránění srážky letadel. Je založený na multiagentním systému A-globe, viz [1]. Pro tuto práci je důležitá hlavně část AgentFly pro plánování trasy, proto popíšu důležité pojmy spojené s plánováním.

1.1.1 Letový plán

Každé letadlo se řídí letovým plánem, který je definován startovním bodem a uspořádanou sekvencí letových manévrů. Startovní bod je definován startovním časem, pozicí, směrovým a normálovým vektorem letadla a počáteční rychlostí letadla.

Letadlo může vykonávat následující typy manévrů:

- **Přímý let** - letadlo nemění svůj směr. Manévr je specifikován délkou a zrychlením. Pouze během tohoto manévru může letadlo měnit rychlost.
- **Horizontální nebo vertikální zatočení** - reprezentuje změnu směru letu. Zatočení je charakterizováno poloměrem zatáčení a úhlem. Horizontální zatáčení se odehrává v rovnině letu. Vertikální zatočení reprezentuje stoupání nebo klesání letadla. Odehrává se v rovině směrového a normálového vektoru letadla.
- **Spirálové stoupání (klesání)** - je podobné horizontálnímu zatáčení, ale zároveň u toho letadlo stoupá nebo klesá. Spirálový manévr je definován poloměrem zatáčení, rychlostí stoupání a dobou stoupání. Podle znaménka poloměru se rozlišuje, zda se zatáčí doleva nebo doprava. Rychlost klesání je rychlost stoupání se zaporným znaménkem. Rychlost stoupání říká, o kolik naroste výška během jednoho okruhu.

Složitější manévry se skládají pouze z těchto tří typů. Příkladem může být vyrovnání letadla do horizontální roviny, což je speciální případ vertikálního zatáčení.

1.2 Plánování trasy

Pro plánování jsou důležité parametry letadla. Letadlo je definované svojí šířkou, délkou, maximální, minimální a optimální rychlostí, maximálním zrychlením a zpomalením, maximálním úhlem stoupání, naklonění a zatáčení. Rozměry letadla jsou potřebné pro testování, zda není v kolizi s nějakým jiným objektem či bezletovou zónou. Úhly určují rozsah pobyhu letadla.

Bezletová zóna je takový prostor, ve kterým se letadlo nesmí nacházet. Celá krajina včetně hor je zabalena do vlastní bezletové zóny. Dalším příkladem bezletové zóny je prostor kolem jaderné elektrárny, nebo nebezpečné nepřátelské území.

Letadlo začíná ve startovním bodě a jsou mu přiřazeny cílové body (Waypoints), kterými musí proletět. Cílový bod je charakterizován polohou, směrovým vektorem, časem

a rychlostí.

- **Poloha bodu** - umístění cílového bodu v prostoru.
- **Směrový vektor** - určuje směr letu letadla skrze cílový bod.
- **Časový parametr** - říká, v jakém čase má letadlo proletět cílovým bodem. Tento parametr nemusí být uveden, v tom případě nehraje roli, v jakou dobu letadlo cílovým bodem proletí, nebo může být ve tvaru časového intervalu.
- **Rychlost** - omezuje letadlo tak, že v cílovém bodě musí letět touto rychlostí. Toto omezení je opět nepovinné, to znamená, že letadlo může v cílovém bodě letět libovolnou rychlostí.

Celá trasa je rozdělená do *segmentů*, které jsou vymezeny startovním bodem a cílovým bodem. Každý cílový bod se stává startovním bodem pro následující cílový bod. Trasa můsí být naplánována tak, aby letadlo proletělo všemi cílovými body v požadovanou dobu a požadovanou rychlostí v prostředí s bezletovými zónami. Jestliže podmínky pro časové a rychlostí omezení nejdou splnit, hledá se takové řešení, které minimalizuje odchylky od požadovaných hodnot.

Kapitola 2

Původní plánování trasy

Tato kapitola rozebírá postup získání plánu pomocí původního plánovače. Je zde ukázáno, jak se hledá trasa neuvažují-li se překážky a poté jak se řeší situace, kde se překážky vyskytují.

Vstupem do plánování jsou segmenty s cílovými body. Pro každý cílový bod segmentu se nejprve určí směrový vektor tak, aby byla trasa optimální. Optimalita výchází s geometrické spojitosti, kde směrový vektor cílového bodu musí být stejný jako směrový vektor startovního bodu dalšího segmentu. V případě výskytu překážek se směrový vektor natočí tak, aby se překážce vyhnulo a zároveň, aby vychýlení bylo minimální od optimálního směru. Směrový vektor definuje směr průletu letadla cílovým bodem.

Následné plánování trasy je prováděno ve dvou fázích. V první fázi probíhá pouze prostorové plánování bez omezujících podmínek pro čas a rychlost. Výsledkem toho je nejkratší trasa letadla, která splňuje požadavek na průnik skrze cílové body, ale v plánu není definováno, jakou rychlostí má letadlo letět. V druhé fázi probíhá časové plánování. Použije se již naplánovaná trasa a dopočítá se zrychlení letadla tak, aby byly splněny požadavky i na časové a rychlostní omezení. Případně se do plánu vkládají časové smyčky (spirálový manévr), jestliže by letadlo přiletělo příliš brzo.

Prostorové plánování z daného bodu do cílového bodu probíhá následovně:

1. Najde se trasa analyticky.
2. Ověří se, zda trasa neprochází bezletovými zónami. V případě, že ano, pokračuje se dál, jinak je analytická trasa řešením.
3. Použije se prostorové prohledávání, viz [2], které hledá přímo v prostoru s bezletovými zónami.

Takto se naplánuje trasa zvlášť pro každý segment a získá se výsledná nejkratší cesta vedoucí přes všechny cílové body. Na této výsledné cestě se provede časové plánování.

2.1 Analytické nalezení trasy do cílového bodu

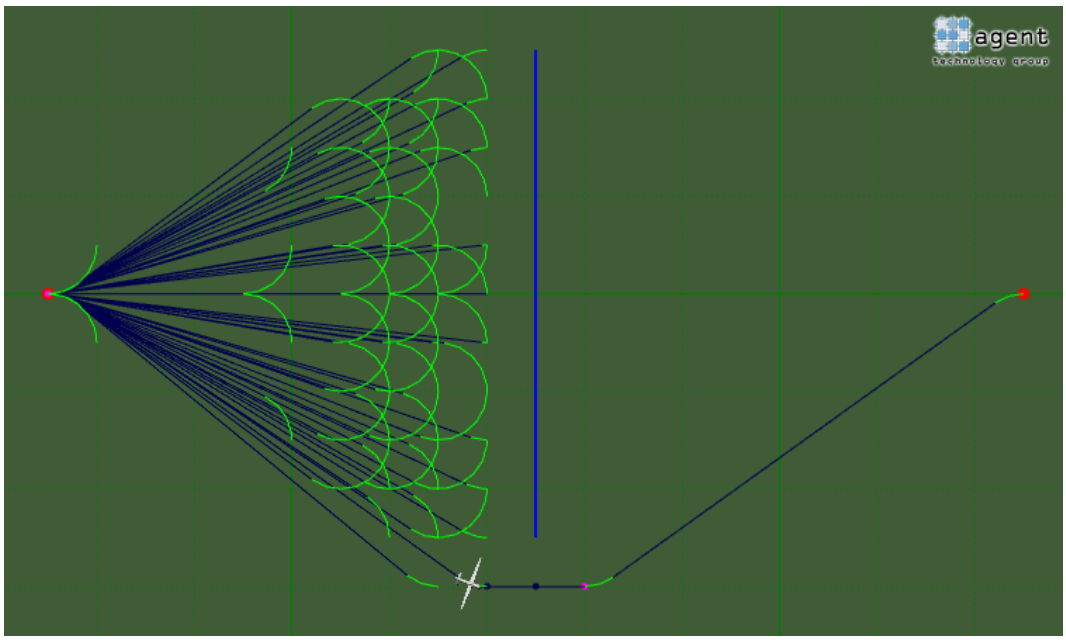
Existuje algoritmus, který najde nejkratší cestu z daného bodu do cílového bodu, kde se neuvažují bezletové zóny, viz [5]. Zjednodušeně najde trasu, která se skládá z počatečních zatočení, rovného úseku a koncových zatočení. Má-li letadlo příliš malý maximální úhel stoupání potřebný pro vystoupání, je do rovného úseku vložena stoupající spirála, tím rovný úsek rozdělí na dva. To samé platí i pro klesání. Tato trasa je nejoptimálnější řešením z hlediska délky dráhy.

2.2 Prostorové prohledávání

Z důvodu rychlosti prohledávání se neprohledáná celý spojitý prostor, proto se prostor diskretizuje. Rozlišení diskrétního vzorkování se dynamicky mění podle vzdálenosti k nejbližší překážce z diskretizované pozice. Rozlišení je vyšší, je-li blíže k překážce. Díky tomu se rychle prohledá velký letový prostor. Rozlišení je také závisle na nastavení plánovače a na rozměrech prostředí, ve kterém se plánuje. Aby se dokázala najít mezera ve velké překážce, je nutné, aby se poblíž překážky s každým krokem zvyšilo dvojnásobně rozlišení. Kvůli diskretnímu vzorkování se trasa průběžně vyhlazuje pomocí analytické trasy.

Prostorové prohledávání hledá sekvenci dílčích manévrů, které vedou ze startovního bodu do cílového bodu tak, aby se minimalizovala kriteriální funkce. Kriteriální funkce může například hodnotit celkovou délku dráhy. Na základě této sekvence se získá letový plán.

K hledání sekvence manévrů se používá A^* algoritmus, viz [2]. Jak algoritmus prohledává je vidět v následující kapitole, kde je popsán časo-prostorový plánovač, který původní algoritmus v modifikované podobě využívá. Obrázek 2.1 ukazuje, jak se stavový prostor prohledává generováním manévrů, kde světle modrá čára znázorňuje překážku, tmavě modrá přímý let a světle zelená zatačecí manévr.



Obrázek 2.1: Prohledávýní stavového prostoru generováním dílčích manévrů.

2.3 Časové plánování

Vstupem je počáteční rychlost, počáteční čas a částečný plán, který nenese informaci o zrychlování a rychlosti letadla. Během tohoto plánování se hledají zrychlení letadla tak, aby se splnily požadavky na časové a rychlostní omezení. V případě, že splnit nejdou, hledá se řešení, které má minimální odchylky od požadavků.

Při hledání řešení se častečný plán rozdělí na úseky, kde je konstantní rychlost a na úseky, kde se může zrychlovat. Vůbec se už neuvažuje pozice v prostoru. V úsecích, kde se může zrychlovat, se najde takové zrychlení, aby se výsledné řešení mělo minimální odchylky od omezujících podmínek.

Nakonec se částečný plán doplní o informace, kde se zrychluje a kam se má vložit čekací smyčka.

2.4 Nedostatky

Hlavní nedostatek tohoto plánování spočívá v tom, že se nejprve vygeneruje prostorovým plánovačem nejoptimálnější trasa, která se dál už upravuje pouze vkládáním časových smyček, ale základ trasy se nemění. Problém nastává, když se nikam časová smyčka nedá vložit, například kvůli překážkám. Pak není žádná možnost, jak najít trasu, která vede jinudy a splňuje časové omezení.

Další nedostatek se může vyskytnout, jestliže by se vyskytovaly bezletové zóny s platnostní v určitém čase. Algoritmus by z principu s takovými zónami by neuměl zacházet. Jelikož v daném prohledávaném bodě nezná čas a tudíž neví, zda bezletová zóna je aktuální.

Vzhledem k tomu, že se v průběhu časového plánování neuvažuje pozice, může nastat poblém, jestliže bychom například požadovali závislost maximální rychlosti na poloze. Kdyby maximální rychlost závisela na nadmořské výšce, tento plánovač by si s tím nedokázal poradit.

Kapitola 3

Návrh časo-prostorového plánovače

Tato kapitola se zaobírá novou metodu plánování, která plánuje v prostoru i čase zároveň, čili uvažuje i rychlost letadla. Oproti původnímu dvoufázovému přístupu probíhá plánování trasy včetně zrychlování najednou. Zrychlování je chápáno i jako zpomalování se záporným znaménkem.

Cílem je nalézt takovou sekvenci letových manévrů, které proletí požadovanými cílovými body s ohledem na časové a rychlostní omezení. V této kapitole jsou popsané typy manévrů, které se využívají při konstruování letového plánu. Dále se zde popisuje postup, jak se hledá analytická trasa, která splňuje omezující podmínky pro čas a rychlost. Analytická trasa je sekvence letových manévrů, pomocí nichž se dostane letadlo do cílového bodu nejkratší cestou, jestliže se neuvažují překážky. Na závěr kapitoly je popsán celý postup získání plánu, kde se uvažují bezletové zóny.

3.1 Letové manévry

V případě prostorového plánování je u jednotlivých manévrů podstatný počáteční bod a směrový vektor. Pak se jednoznačně dá určit koncový bod, směrový vektor na konci manévru a délka celého manévru.

U časo-prostorového plánování je, kromě počátečního bodu a směrového vektoru, důležitý i čas zahájení manévru a počáteční rychlost. Počáteční rychlost jednoznačně určí dobu trvání manévru a koncovou rychlost. Jestliže známe čas zahájení manévru, můžeme pak zjistit čas na konci manévru.

Během prohledávání se generují následující manévry:

- **Přímý let** - délka manévru se zmenšuje s přibližováním ke překážce.
- **Horizontální zatočení** - úhel zatočení je opět závislí na vzdálenosti od překážky.
- **Vertikální stoupání (klesání)** - úhel zatočení je roven maximálnímu úhlu stoupání (klesání).
- **Spirálové stoupání (klesání)** - je definovám počtem okruhů a horizontálně se zatáčí po kružnici s nejmenším možným poloměrem.
- **Vyrovnání do horizontální roviny** - speciální případ vertikálního stoupání, kde na konci manévru je letadlo v horizontální rovině.
- **Vyrovnání do horizontální roviny ve výšce cílového bodu** - skládá se z přímého letu a nasledně se letadlo vyrovná do horizontální roviny tak, aby mělo požadovanou výšku.
- **Analytická trasa** - je speciálním manévrem, jehož popis je v sekci 3.2.

3.1.1 Ohodnocení manévru

Každý manévr je ohodnocen tzv. *fitness* funkcí, která stanovuje kvalitu manévru. Tato funkce je definována podle následujícího vztahu:

$$fitness = w_{length} \cdot length + w_{penalty} \cdot penalty, \tag{3.1}$$

kde w_{length} a $w_{penalty}$ jsou nastavitelné váhy, $length$ je délka manévru a $penalty$ je penalizace za neoptimální rychlost. Penalizace za neoptimální rychlost je znázorněna na obrázku 3.1, kde je zobrazena závislost rychlosti letadla na čase a plocha vymezená mezi rychlostí letadla a optimální rychlostí je hodnota penalizace.

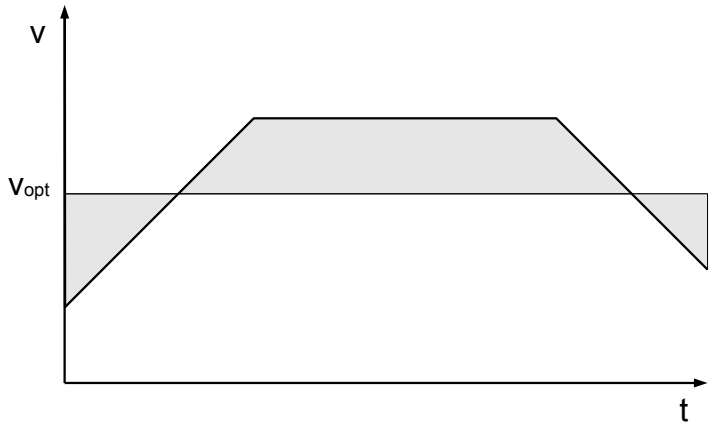
V úseku, kdy letadlo letí konstantní rychlostí, je penalizace

$$penalty = t \cdot |v - v_{opt}|, \tag{3.2}$$

kde t je doba letu přes úsek, v je rychlost letu, v_{opt} je optimální rychlost letadla.

V případě, že letadlo zrychluje s konstantním zrychlením, je penalizace

$$penalty = t \cdot \left| \frac{v_0 + v}{2} - v_{opt} \right|, \tag{3.3}$$



Obrázek 3.1: Penalizace za neoptimální rychlost. Šedá plocha znázorňuje penalizaci, když letadlo letí rozdílnou rychlostí od optimální.

jestliže platí $(v_{opt} > v \wedge v_{opt} > v) \vee (v_{opt} < v \wedge v_{opt} < v)$, jinak

$$penalty = \frac{(v_{opt} - v_0)^2 + (v - v_{opt})^2}{2\,|a|}\,, \tag{3.4}$$

kde t je doba zrychlování, v_0 je počáteční rychlost, v je koncová rychlost a a je zrychelní.

3.2 Analytická trasa v časo-prostorovém plánování

Stejně jako u prostorového plánování, tak i u časo-prostorového plánování hraje důležitou roli analytická trasa. Analytická trasa v časo-prostorovém plánování navíc splňuje omezení času a rychlosti v cílovém bodě. Pro nalezení analytické trasy, která zároveň splňuje požadavek na časové a rychlostní omezení, využívám původní algoritmus, viz [5]. Původním algoritmem se najde nejkratší trasa a poté se podle omezení vypočítají zrychlení letadla. V obecném případě se nalezená trasa skládá z:

- **Počáteční horizontální a vertikální zatočení** - během tohoto úseku letí letadlo konstantní rychlostí. Délka úseku je $s_{initial}$.
- **Přímý let** - zde letadlo může měnit rychlost. To je klíčová část pro časové plánování. Délka úseku je $s_{straight}$.

- **Spirála** - v případě, že se letadlo potřebuje dostat do vyšší polohy a nestačí mu jeho maximální stoupající úhel, vloží se ve středu rovného úseku stoupající spirála, během které je rychlost konstantní. Tím se rovný úsek rozpadne na dva - jeden je před spirálou a druhý po spirále. Délka $s_{straight}$ je obou rovných úseků dohromady. Délka spirály je s_{spiral}. Přidáváním nadbytečných okruhů má význam čekací smyčky v případě, že by letadlo do cíle doletělo příliš brzo.
- **Koncové horizontální a vertikální zatočení** - během tohoto úseku letadlo opět nemůže měnit rychlost. Délka úseku je s_{final}.

V následujících vzorcích budu používat označení proměnných:

- v_s - počáteční rychlost letadla v analytické trase.
- v_f - koncová rychlost letadla.
- v_{opt} - optimální rychlost letadla.
- v_{min} - minimální možná rychlost letadla.
- v_{min} - maximální možná rychlost letadla.
- a_{max} - maximální zrychlení letadla.
- a_{min} - maximální zpomalení letadla. Nabývá zaporné hodnoty.
- t_c - celkový čas trasy.
- t_{min} - minimální možná doba trasy.
- t_{max} - maximální možná doba trasy bez čekacích smyček.
- t_s - doba trvání počátečních zatočení.
- t_f - doba trvání koncových zatočení.
- s - celková délka trasy. Platí

$$s = s_{initial} + s_{straight} + s_{spiral} + s_{final}. \tag{3.5}$$

Poznámka: Zrychluje-li letadlo záporným zrychlením, znamená to, že zpomaluje.

Nezávisle na omezení se dá stanovit doba počatečních zatočení

$$t_s = \frac{s_{initial}}{v_s}\,, \tag{3.6}$$

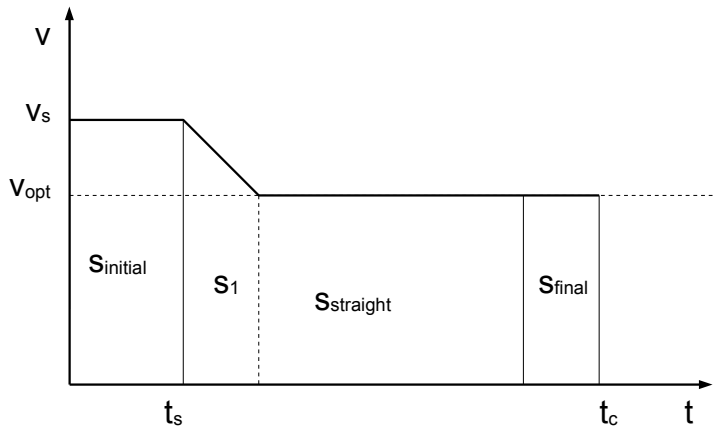
protože v tomto úseku se rychlost měnit nedá.

Podle typu omezení nastavají různé případy.

3.2.1 Plánování bez omezení času a konečné rychlosti

V tomto případě nehraje roli, jak dlouho letadlo poletí, ani jakou bude mít rychlost na konci. Proto se letadlo zrychlí tak, aby letělo optimální rychlostí a tím celý úsek měl minimální penalizaci za neoptimální rychlost.

Obrázek 3.2 ukazuje průběh rychlosti letadla v čase. Během počátečních zatáček letí stále počáteční rychlostí v_s. V rovném úseku začne zrychlovat zrychlením a.



Obrázek 3.2: Průběh rychlosti v analytické trase bez časového omezení.

Je-li $v_s < v_{opt}$, bude letadlo zrychlovat maximálním zrychlením a_{max}, jinak zrychluje maximálním zpomalením a_{min}. Letadlo zrychuje až do dosažení optimální rychlosti. Poté pokračuje již stálou rychlostí a konečná rychlost $v_f = v_{opt}$.

V průběhu zrychlování urazí dráhu

$$s_1 = \frac{v_{opt}^2 - v_s^2}{2a}\,. \tag{3.7}$$

Rovnice 3.7 vychází z

$$v_f = v_s + at, \tag{3.8}$$

$$s_1 = v_s t + \frac{1}{2} a t^2, \tag{3.9}$$

a

$$v_f = v_{opt}, \tag{3.10}$$

kde t je doba zrychlování.

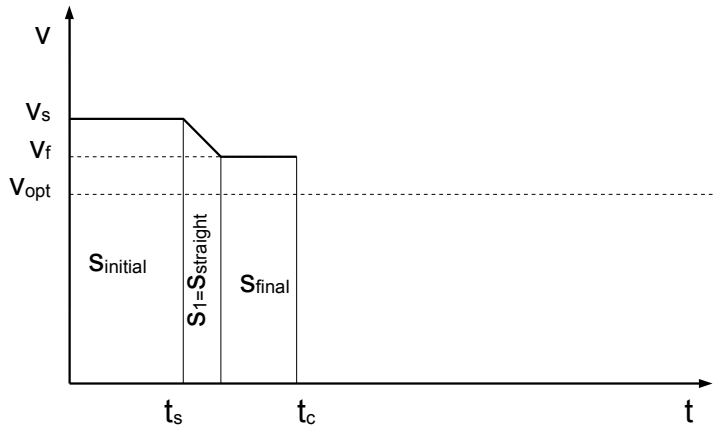
Jestliže je rovný úsek příliš krátký, aby se během něho dosáhla optimální rychlost, zrychluje se v průběhu celého rovného úseku. To znamená, že $s_1 = s_{straight}$. Dosažená konečná rychlost se pak vypočítá podle

$$v_f = \sqrt{2 a s_{straight} + v_s^2}, \tag{3.11}$$

kde a je zrychlení letadla. Rovnice 3.11 vychází z rovnic 3.8, 3.9 a

$$s_1 = s_{straight}. \tag{3.12}$$

Obrázek 3.3 zobrazuje případ, kdy se nestihne dosáhnout optimální rychlosti.



Obrázek 3.3: Nedosažení optimální rychlosti v analytické trase bez časového omezení.

Celková doba analytické trasy se vypočte podle

$$t_c = t_s + \frac{v_f - v_s}{a} + \frac{\left(s_{straight} - s_1 + s_{final}\right)}{v_f}. \tag{3.13}$$

3.2.2 Plánování s časovým omezením bez omezení konečné rychlosti

V tomto případě je pevně daná celková doba trasy t_c a nezáleží na koncové rychlosti v_f. Označím si $t_x = t_c - t_s$.

Postup získání zrychlení během přímeho úseku probíhá následovně:

1. Vypočte se $s_x = s_{straight} + s_{spiral} + s_{final}$.

2. Ověří se, zda je letadlo schopné letět tak pomalu, aby do cíle nedorazilo předčasně. Nejprve označím proměnné:

$$t_{min1} = \frac{v_{min} - v_s}{a_{min}}\,, \tag{3.14}$$

$$s_{min} = \frac{v_{min}^2 - v_s^2}{2a_{min}} + v_{min} \cdot (t_x - t_{min1}). \tag{3.15}$$

   Jestliže je $s_{min} > s_x$ a zároveň $t - t_{min1} > 0$, dorazí letadlo předčasně a musí se vložit čekací smyčka. V opačném případě se pokračuje dalším bodem.

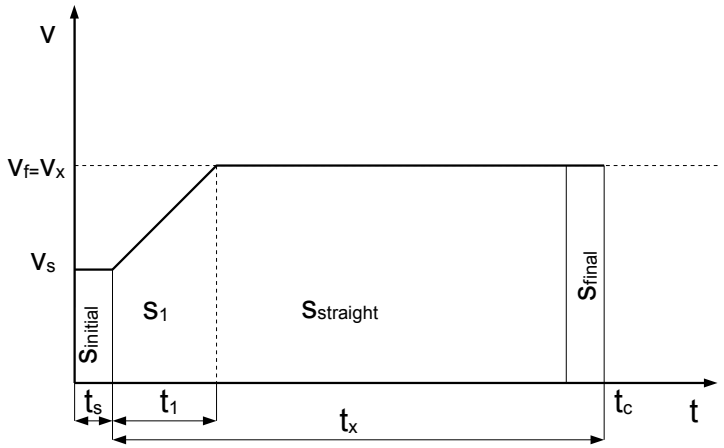
   Jeden okruh čekací smyčky má pevně definovanou délku podle maximálního úhlu zatočení letadla. Tuto délku označme s_{loop}. Potom počet nutných čekacích okruhů se spočte podle rovnice

$$loops = \text{ceil}\left(\frac{s_{min} - s}{s_{loop}}\right), \tag{3.16}$$

   kde funkce ceil vrací zaokrouhlenou celočíselnou hodnotu směrem nahoru. Jestliže původní trasa obsahovala spirálu, přeupraví se trasa tak, aby obsahovala spirála o *loops* okruhů víc. Jinak se vloží do středu rovného úseku spirála o *loops* okruzích. Tím se změní hodnota s_{spiral} a v případě verikálního stoupání i hodnota $s_{straight}$. Algoritmus se spustí znova s upravenýma hodnotama.

3. Jestliže je délka $s_{straight}$ nulová, algoritmus končí, v opačném případě se určí zrychlení a_1 na začátku rovného úseku. Vždy se použije buď maximální zrychlení a_{max}, nebo maximální zpomalení a_{min}. Podle průměrné rychlosti $v_p = \frac{s_x}{t_x}$ se zjistí, zda se bude zrychlovat nebo zpomalovat. Je-li $v_p > v_s$, pak $a_1 = a_{max}$, jinak $a_1 = a_{min}$.

4. Spočítá se rychlost v_x, na kterou letadlo během rovného úseku zrychlí a poletí touto rychlostí celou dobu až do cílového bodu. Na obrázku 3.4 je zobrazen možný případ vývoje rychlosti při letu analyticky nalezenou trasou s časovým omezením. Je zde



Obrázek 3.4: Vývoj rychlosti u analytické trasy s časovým omezením.

vidět, že ve fázi zatáčení letí konstantní rychlostí, poté začne akcelerovat na rychlost v_x.

Při počátečním zrychlování urazí dráhu

$$s_1 = \frac{v_x^2 - v_s^2}{2a_1} \tag{3.17}$$

za dobu

$$t_1 = \frac{v_x - v_s}{a_1}\,. \tag{3.18}$$

Platí rovnice

$$s_x = s_1 + v_x(t_x - t_1), \tag{3.19}$$

do které se dosadí 3.17 a 3.18 a získá se

$$s_x = \frac{v_x^2 - v_s^2}{2a_1} + v_x t_x - \frac{v_x^2 - v_x v_s}{a_1}\,. \tag{3.20}$$

Úpravou dostaneme

$$v_x^2 + (-2a_1 t_x - 2v_s)v_x + (2a_1 s_x + v_s^2) = 0. \tag{3.21}$$

Označím si

$$B = -2a_1 t_x - 2v_s, \tag{3.22}$$

$$C = 2a_1 s_x + v_s^2 \tag{3.23}$$

a

$$D = B^2 - 4C. \tag{3.24}$$

Je-li $t_x < 0$ nebo $D < 0$, potom

$$v_x = v_{max}, \tag{3.25}$$

jinak

$$v_x = \frac{-B \pm \sqrt{D}}{2A}, \tag{3.26}$$

kde v_x je ten kořen, který je nezáporný.

5. Zkontroluje se, zda vypočtená rychlost v_x je letadlem dosažitelná. Je-li $v_x > v_{max}$, pak $v_x = v_{max}$ a je-li $v_x < v_{min}$, pak $v_x = v_{min}$.

6. Otestuje se, jestli $s_1 < \frac{s_{straight}}{2}$, kvůli případné spirále. Jestliže to není splněno, nastaví se $s_1 = \frac{s_{straight}}{2}$ a přepočítá se v_x podle

$$v_x = \sqrt{2a_1 s_1 + v_s^2}. \tag{3.27}$$

7. Spočítá se celková doba trasy a jestliže se liší od požadované, zpenalizuje se to ve *fitness* funkci.

3.2.3 Plánování s časovým omezením a omezením konečné rychlosti

Je pevně daná celková doba trasy t_c a požadovaná koncová rychlost v_f. Tím je jednoznačně stanovena doba koncových zatočení

$$t_f = \frac{s_{final}}{v_f}, \tag{3.28}$$

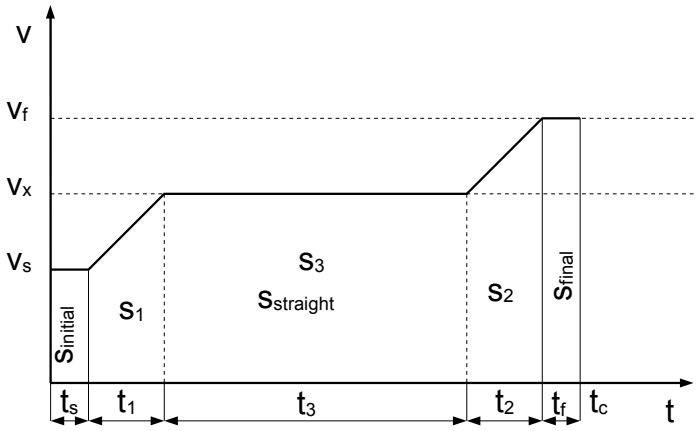
jelikož se v průběhu zatáčení nemůže měnit rychlost.

Označím si $t_x = t_c - t_s - t_f$. Doba t_x má význam trvání přímého úseku včetně případné spirály. Postup získání zrychlení během přímeho úseku probíhá následovně:

1. Vypočte se $s_x = s_{straight} + s_{spiral}$.

2. Ověří se, zda je letadlo schopné letět tak pomalu, aby do cíle nedorazilo předčasně. Nejprve označím proměnné:

$$t_{min1} = \frac{v_{min} - v_s}{a_{min}}, \tag{3.29}$$



Obrázek 3.5: Vývoj rychlosti u analytické trasy s časovým a rychlostním omezením.

$$t_{min2} = \frac{v_f - v_{min}}{a_{max}}, \tag{3.30}$$

$$s_{min} = \frac{v_{min}^2 - v_s^2}{2a_{min}} + \frac{v_f^2 - v_{min}^2}{2a_{max}} + v_{min} \cdot (t_x - t_{min1} - t_{min2}). \tag{3.31}$$

Jestliže je $s_{min} > s_x$ a zároveň $t - t_{min1} - t_{min2} > 0$, dorazí letadlo předčasně a musí se vložit čekací smyčka. V opačném případě se pokračuje dalším bodem. Čekací smyčka se vkládá stejným způsobem jako v případě omezení pouze časem.

3. Jestliže je délka $s_{straight}$ nulová, algoritmus končí, v opačném případě se určí zrychlení a_1 na začátku rovného úseku a zrychlení a_2 na konci úseku. Vždy se použije buď maximální zrychlení a_{max}, nebo maximální zpomalení a_{min}. Problém je pouze určit, zda se bude zrychlovat nebo zpomalovat. Jako odhad slouží průměrná rychlost $v_p = \frac{s_x}{t_x}$. Je-li $v_p > v_s$, pak $a_1 = a_{max}$, jinak $a_1 = a_{min}$. A pro koncové zrychlení platí, je-li $v_f > v_p$, pak $a_2 = a_{max}$, jinak $a_2 = a_{min}$.

4. Spočítá se rychlost v_x, na kterou letadlo během rovného úseku zrychlí a letí touto rychlostí celou dobu až do okamžiku, kdy na konci úseku začne zrychlovat, aby dosáhlo požadovanou koncovou rychlost v_f.

   Na obrázku 3.5 je zobrazen možný případ vývoje rychlosti při letu analyticky nalezenou trasou s časovým a rychlostním omezením. Při počátečním zrychlování urazí

dráhu

$$s_1 = \frac{v_x^2 - v_s^2}{2a_1} \tag{3.32}$$

za dobu

$$t_1 = \frac{v_x - v_s}{a_1}\,. \tag{3.33}$$

Ve fázi koncového zrychlování urazí dráhu

$$s_2 = \frac{v_f - v_x}{2a_2} \tag{3.34}$$

za dobu

$$t_2 = \frac{v_f - v_x}{a_2}\,. \tag{3.35}$$

Po počátečním zrychlování letí konstantní rychlostí v_x po dobu

$$t_3 = t_x - t_1 - t_2 \tag{3.36}$$

a uletí dráhu

$$s_3 = v_x \cdot t_3. \tag{3.37}$$

Zároveň platí vztah

$$s_3 = s_x - s_1 - s_2. \tag{3.38}$$

Z rovnic 3.37 a 3.38 se získá rovnice

$$v_x \left(t_x - \frac{v_x - v_s}{a_1} - \frac{v_f - v_x}{a_2} \right) = s_x - \frac{v_x^2 - v_s^2}{2a_1} - \frac{v_f^2 - v_x^2}{2a_2}\,. \tag{3.39}$$

Úpravou vyjde rovnice

$$(a_1 - a_2)v_x^2 + 2(a_2 v_s - a_1 v_f + a_1 a_2 t_x)v_x + (a_1 v_f^2 - a_2 v_s^2 - 2 s_x a_1 a_2) = 0. \tag{3.40}$$

Označím si

$$A = (a_1 - a_2), \tag{3.41}$$

$$B = 2(a_2 v_s - a_1 v_f + a_1 a_2 t_x), \tag{3.42}$$

$$C = a_1 v_f^2 - a_2 v_s^2 - 2 s_x a_1 a_2 \tag{3.43}$$

a

$$D = B^2 - 4AC. \tag{3.44}$$

Je-li $t_x < 0$ nebo $(A \neq 0 \wedge D < 0)$, potom

$$v_x = v_{max}, \tag{3.45}$$

jinak je-li $A = 0$, pak

$$v_x = -\frac{C}{B}, \tag{3.46}$$

v opačném případě

$$v_x = \frac{-B \pm \sqrt{D}}{2A}, \tag{3.47}$$

kde v_x je ten kořen, který je nezáporný. Poté se vypočítají hodnoty s_1 a s_2 a jestliže jsou obě nezáporné, pokračuje se dalším bodem. Je-li alespoň jedna záporná, znamená to, že se v bodě 3 špatně zvolilo zrychlení a_1 nebo a_2. Zrychlení a_1 a a_2 můžou nabývat pouze dvou hodnot a_{max} a a_{min}, tak se postupně vyzkoušejí ostatní kombinace pro a_1 a a_2, až hodnoty v_x, s_1 a s_2 budou nezáporné.

5. Zkontroluje se, zda vypočtená rychlost v_x je letadlem dosažitelná. Je-li $v_x > v_{max}$, pak $v_x = v_{max}$ a je-li $v_x < v_{min}$, pak $v_x = v_{min}$.

6. Otestuje se, jestli $s_1 < \frac{s_{straight}}{2}$, kvůli případné spirále. Jestliže to není splněno, nastaví se $s_1 = \frac{s_{straight}}{2}$ a přepočítá se v_x podle

$$v_x = \sqrt{2a_1 s_1 + v_s^2}, \tag{3.48}$$

a znova se přepočítají hodnoty t_1, s_2 a t_2.

7. Otestuje se, zda $s_2 < \frac{s_{straight}}{2}$. V případě nesplnění se nastaví $s_2 = \frac{s_{straight}}{2}$ a přepočítá se v_f podle

$$v_f = \sqrt{2a_2 s_2 + v_x^2} \tag{3.49}$$

a t_2. To má za následek, že na konci nebude požadovaná rychlost. To se penalizuje ve *fitness* funkci.

8. Spočítá se celková doba trasy a jestliže se liší od požadované, penalizuje se to opět ve *fitness* funkci.

3.2.4 Ohodnocení analytické trasy

Oproti běžnému manévru, který je ohodnocen podle vztahu 3.1, který zavisí na délce a penalizaci za neoptimální rychlost, je analytická trasa navíc penalizována za nesplnění

omezujících podmínek. Ohodnocovací *fitness* funkce je stanovena podle

$$fitness = w_{length} \cdot length + w_{penalty} \cdot penalty + w_{time} \cdot \Delta t_c + w_{velocity} \cdot \Delta v_f, \tag{3.50}$$

kde w_{time} a $w_{velocity}$ jsou nastavitelné váhy, Δt_c je rozdíl požadované hodnoty celkového času od skutečné a Δv_f je rozdíl požadované koncové rychlosti od výsledné. Váha $w_{velocity}$ má význam pouze když je na konci požadována konkrétní rychlost, jinak nabývá nulové hodnoty. To stejné platí pro w_{time}, kde má význam pouze pro časové omezení. Je-li časové omezení ve formě intervalu $\langle t_{min}, t_{max} \rangle$, pak se Δt_c určí následovně: Je-li skutečná hodtnota $t_c < t_{min}$, pak $\Delta t_c = t_{min} - t_c$, jinak je-li $t_c > t_{max}$, pak $\Delta t_c = t_c - t_{max}$, a v ostatních případech je $\Delta t_c = 0$.

3.2.5 Minimální doba trvání analytické trasy

U časo-protorového prohledávání bude potřeba zjistit nejmenší možnou dobu trvání analytické trasy t_{min}. Tato doba se určí, když letadlo zrychlí maximálním zrychlením a_{max} na maximální možnou rychlost v_{max}. Analytická trasa se uvažuje bez čekacích smyček. Zatímco zrychluje, urazí dráhu

$$s_1 = \frac{v_{max}^2 - v_s^2}{2a_{max}}. \tag{3.51}$$

Kvůli možné spirále se oveří, zda $s_1 < \frac{s_{straight}}{2}$. Jestliže to neplatí, nastaví se

$$s_1 = \frac{s_{straight}}{2} \tag{3.52}$$

a

$$v_f = \sqrt{2a_{max}s_1 + v_s^2}, \tag{3.53}$$

jinak je

$$v_f = v_{max}. \tag{3.54}$$

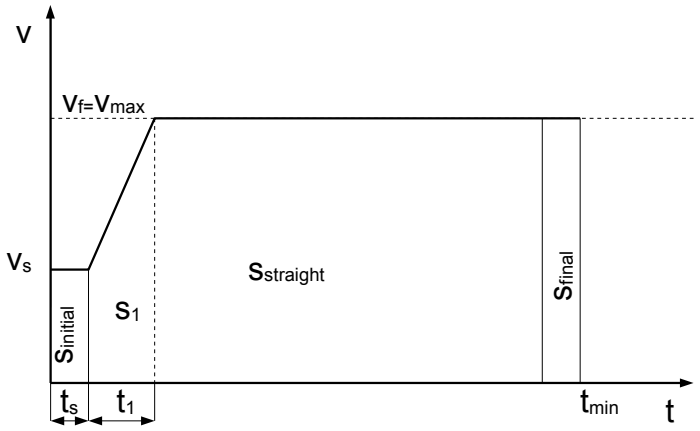
Minimílní doba trasy se pak vypočte podle

$$t_{min} = t_s + t_1 + \frac{s_{straight} - s_1 + s_{spiral} + s_{final}}{v_f}, \tag{3.55}$$

kde

$$t_1 = \frac{v_f - v_s}{a_{max}}. \tag{3.56}$$

Obrázek 3.6 ukazuje průběh rychlosti při zrychlování na maximání rychlost.



Obrázek 3.6: Vývoj rychlosti u analytické trasy s minimální možnou dobou trvání.

3.2.6 Maximální doba trvání trasy analytické trasy bez čekacích smyček

Maximální doba trvání se zjistí, když letadlo zrychlí minimálním zpomalením a_{min} na minimální možnou rychlost v_{min}. Analytická trasa se uvažuje bez čekacích smyček. Zatímco zrychluje, urazí dráhu

$$s_1 = \frac{v_{min}^2 - v_s^2}{2a_{min}}. \tag{3.57}$$

Kvůli možné spirále se oveří, zda $s_1 < \frac{s_{straight}}{2}$. Jestliže to neplatí, nastaví se

$$s_1 = \frac{s_{straight}}{2} \tag{3.58}$$

a

$$v_f = \sqrt{2a_{min}s_1 + v_s^2}, \tag{3.59}$$

jinak je

$$v_f = v_{min}. \tag{3.60}$$

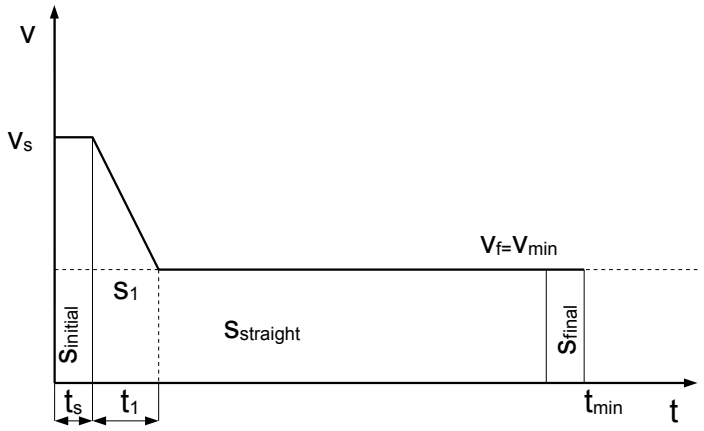
Minimílní doba trasy se pak vypočte podle

$$t_{max} = t_s + t_1 + \frac{s_{straight} - s_1 + s_{spiral} + s_{final}}{v_f}, \tag{3.61}$$

kde

$$t_1 = \frac{v_f - v_s}{a_{min}}. \tag{3.62}$$

Obrázek 3.7 ukazuje průběh rychlosti při zpomalování na minimální rychlost.



Obrázek 3.7: Vývoj rychlosti u analytické trasy bez časových smyček s maximální možnou dobou trvání.

3.3 Časo-prostorové prohledávání

V této části je popsána metoda hledání kompletního plánu včetně zrychlování.

3.3.1 Plánování přes více segmentů

Původní prostorový plánovač se spouští zvlášť pro každý segment. A pak celek proběhne časovým plánovačem. Jelikož jednotlivé cílové body mohou být bez omezení na čas, nebo mohou mít omezení ve tvaru časového intervalu, nemůže plánování probíhát zvlášť po segmentech. Nejprve se proto musí zjistit, pro kolik segmentů se bude najednou plánovat. Algoritmus je následující:

1. Počet plánovaných segmentů *segments* je nastaven na 1 a $S = 1$.
2. Vezme se následující cílový bod a označí se W.
3. Jestliže je W časově omezen hodnotou, algoritmus končí.
4. Do W se přířadí další cílový bod.
5. Ikrementuje se S o 1.
6. Jestliže je W časově omezený, nastaví se $segments = S$. Jedná-li se o časové omezení intervalem, pokračuje se bodem 4, jinak algoritmus končí.

7. Jde se na 4.

Na příkladu 3.1 je ukázáno, proč je důležité plánovat přes více segmentů najednou.

Příklad 3.1 (Plánování zvlášť po segmentech): Uvažujme případ, že by plánování probíhalo zvlášť po segmentech. První cílový bod by byl bez časového omezení a následující bod by byl časově omezen. Kdyby se plánoval první segment zvlášť, letadlo by zrychlilo tak, aby v něm letělo optimální rychlostí, protože není pro cílový bod časové omezení. Problém by nastal v tom, že následující časově omezený cílový bod nemusel být dosažen v požadovaném čase. V tom případě by letadlo muselo letět vyšší rychlostí už v prvním segmentu. Tudíž se musejí oba segmenty při plánování uvažovat najednou.

3.3.2 Stavy v časo-prostorovém prohledávání

Každý stav v prohledávání je definován polohou, směrovým vektorem, časem a rychlostí. Stavy jsou podle těchto parametrů diskretizovány. Diskretizace polohy je v závislosti na vzdálenosti od překážky, viz [2] a diskretizace času a rychlosti je nastavitelná parametrem $\Delta time$ a $\Delta velocity$. Stav má navíc ještě nasledující parametry:

- **g** - celková cena cesty ze startovního bodu do daného stavu.
- **h** - heuristický odhad ceny z daného stavu do posledního cílového bodu.
- **f** - ohodnocení stavu, $f = g + h$.
- **maneuvreType** - označuje typ manévru, pomocí kterého se do daného stavu dostalo. Nabývá hodnot ze {straight, turn_left, turn_right, pitch_up, pitch_down, spiral_left, spiral_right, pitch_to_level, to_end_level, smooth}, které označují:
  - **straight** - přímý let. Tento manévr by se měl generovat s různým zrychlením kvůli časové dimenzi, ale testování ukázalo, že je to zbytečné, neboť zrychlování se řeší jiným způsobem, popsaným níže a zpomalování vzniká přirozeným vytvářením časových smyček v prohledávání.
  - **turn_left** - zatočení doleva (horizonální zatáčení).
  - **turn_right** - zatočení doprava (horizonální zatáčení).
  - **pitch_up** - stoupání (vertikální zatáčení).
  - **pitch_down** - klesání (vertikální zatáčení).

  - **spiral_left** - spirála rotující směrem doleva.
  - **spiral_right** - spirála rotující směrem doprava.
  - **pitch_to_level** - vyrovnání letadla do horizontální roviny.
  - **to_end_level** - vyrovnání letadla do horizontální roviny ve výšce cílového bodu.
  - **smooth** - analytická trasa. Analytickou trasou se řeší zrychlování letadla. Využívá se k tomu postup minimalizace doby trvání analytické trasy popsaný v sekci 3.2.5.

- **pred** - předchozí stav, ze kterého se do daného stavu dostalo.
- **smoothToEndIsUsable** - identifikuje, zda existuje z daného stavu do cílového analytická trasa.
- **segment** - příslušnost stavu k segmentu. Význam má u plánování přes více segmentů. Podle této hodnoty se určuje v pořadí nejbližší cílový bod.
- **lastInSegment** - oznamuje, že následující stav vygenerovaný z daného stavu bude náležet dalšímu segmentu.

Heuristický odhad stavu $s_{segment}$ se počítá jako suma *fitness* funkcí analytických tras A_i, které vedou ze stavu s_i do cílového bodu w_i, kde $i = segment..segments-1$, $segments$ je počet segmentů, přes který se plánuje, w_i je cílový bod v i-tým segmentu a s_{i+1} je koncový stav analytické trasy A_i.

Poznámka: Není-li cílový bod nějak specifikován, je namysli cílový bod segmentu, ke kterému náleží daný stav.

3.3.3 Algoritmus hlednání plánu

Vstup: startovní bod a posloupnost cílových bodů.
Výstup: letový plán.

1. Zjistí se, pro kolik segmentů se bude plánovat najednout. Tím se získa hodnota $segments$.
2. Podle startovního bodu se vytvoří počáteční stav s_0.

3. Oveří se, zda existuje ze stavu s_0 analytická trasa do prvního cílového bodu. Jestliže ano, nastaví se parametr $smoothToEndUsable$ stavu s_0 na 1.

4. Vloží se s_0 do seznamu *Open*. Vlákdání do seznamu je popsáno v následující kapitole v sekci 4.4.1.2.

5. Jestliže je seznam *Open* prázdný, skočí se na 16, jinak se vyjme z *Open* nejlepé ohodnocený stav a přiřadí se do s_{cur}. Nejlépe ohodnocený stav je takový, který má minimální hodnotu f. Je-li tam takových stavů víc, vezme se pouze jeden.

6. Stav s_{cur} se vloží do seznamu *Closed*.

7. Je-li $smoothToEndUsable = 1$ stavu s_{cur} pak:

   (a) Vytvoří se analytická trasa A ze stavu s_{cur} do jeho cílového bodu, který se označí $w_{segment}$.

   (b) Přiřadí se $s_{tmp} = s_{cur}$.

   (c) Koncový stav analytické trasy se přířadí do s_{cur}, tím má už hodnotu $segment$ o 1 větší.

   (d) Přepočítá se heuristika stavu s_{cur}.

   (e) Nastaví se u s_{cur} parametr $lastInSegment = 1$.

   (f) Je-li hodnota $segment$ stavu s_{cur} rovna $segments - 1$, prohledávání končí a jde se na 16.

   (g) Oveří se, zda existuje analytická trasa ze stavu s_{cur} do jeho cílového bodu, který je označen $w_{segment+1}$. Jestliže ano, nastaví se parametr $smoothToEndUsable$ stavu s_{cur} na 1.

   (h) Vloží se s_{cur} do seznamu *Open*.

   (i) Vygenerují se analytické trasy s různými dobami trvání. Jestliže je cílový bod $w_{segment}$ časově omezen intervalem $\langle t_1, t_2 \rangle$, přířadí se $t_{min} = t_1$ a $t_{max} = t_2$, jinak t_{min} je minimální a možná doba analytické trasy A a t_{max} je maximální možná doba.

   (j) Pro všechny $t \in \{t_{min}, t_{min}+2\cdot\Delta time, t_{min}+4\cdot\Delta time, ..., t_{min}+2n\cdot\Delta time, t_{max}\}$, kde $t_{min} + 2n \cdot \Delta time < t_{max}$, se vytvoří analytická trasa A_t ze stavu s_{tmp} do jeho cílového bodu s tím, že se omezí pro časovou hodnotu t. Používá se dvounásobný skok $\Delta time$ z toho důvodu, že při rozlišení $\Delta time$ by stav s časem

rovným $t_{min}+(2k-1)\cdot\Delta time$ byl schodný se stavem s časem $t_{min}+2k\cdot\Delta time$. Z tohoto důvodu nemá význam generovat stavy s nižším časovým rozestupem. Dále se vezme konečný stav trasy A_t, nastaví se mu parametr $lastInSegment = 1$, přepočítá se mu heuristika a ověří se, zda z něho vede analytická trasa do $w_{segment+1}$. Jestli ano, nastaví se parametr $smoothToEndUsable = 1$. Pak jestli tento stav není v *Closed* seznamu, vloží se do seznamu *Open*.

(k) Jde se na 5.

8. Pro stav s_{cur} se vygenerují všechny možné letové manévry, viz sekci 3.1. V nastavením plánovače je definováno, jaké manévry se generují.

9. Pro každý vygenerovaný manévr M se vezme jeho koncový stav a označí se s_{new}.

10. Jestliže je stav s_{new} v seznamu *Closed*, zahodí se a vezme se další koncový stav z vygenerovaných manévrů. Jestli už žádný není, skočí se na 5. Zda je stav obsažen v seznamu je pospáno v sekci 4.4.1.1 následující kapitoli.

11. Ověří se, zda manévr M neprochází bezletovou zónou. Jestliže prochází, stav s_{new} se pro tento manévr opět zahodí. Jedná-li se o spirálovitý manévr, otestuje se spirála na opačnou stranu a jestliže ta už bezletovou zónou neprochází, stav se ponechá.

12. Spočítá se heuristika stavu s_{new} a v případě, že vede z něho analytická trasa do cílového bodu, nastaví se mu $smoothToEndUsable = 1$.

13. Zjistí se, zda do stavu s_{new} nevede optimálnější cesta. Rozepíšou se předchůdci stavu s_{new} jako $s_0, s_1, ..., s_{n-1}$, kde $s_n = s_{new}$ a s_{i-1} je předchůdce stavu s_i. Jestliže existuje stav s_i takový, že $i < n-1$ a je nejmenší možné a zároveň existuje analytická trasa A ze stavu s_i do s_{new}, která má lepší ohodnocení než původní trasa, pak se stavu s_{new} změní předchůdce na s_i, tím se mu změní ohodnocení g a změní se mu $maneuvreType = smooth$. Zároveň je nutné zajistit nejrychlejší možnou trasu. Proto jestliže existuje analytická trasa A, minimalizuje se její doba trvání, tím se získá nový koncový stav, pro který se postupuje od bodu 10 s tím, že se už nehledá nejoptimálnější cesta.

14. Vloží se do seznamu *Open*.

15. Skočí se na 5.

16. Je-li s_{cur} nenadefinovaný stav, znamená to, že se cestu neporařilo najít. Jinak se podle předchůdců stavu s_{cur} vytvoří letový plán.

    (a) Označme $s_n = s_{cur}$ a s_{i-1} je předchůdce stavu s_i, kde $i = \{0, 1, ..., n\}$.

    (b) Pro k od 1 do n se provede:

    (c) Podle *maneuvreType* stavu s_k se vytvoří požadovaný typ manévru, který je začíná ve stavu s_{k-1} a končí ve stavu s_k.

    (d) Manévr i s jeho parametry se vloží do letového plánu, který patří k segmentu, ke kterému je přiřazen stav s_k.

17. Celý algoritmus se spustí znova na zbylé segmenty.

3.3.3.1 Pseudokód prohledávání

Vstup: startovní bod w_0 a posloupnost cílových bodů $W = (w_0, w_1, ..., w_n)$.
Výstup: letový plán.
Kód ukazuje pouze vytvoření plánu pro segmenty, pro které se najednou plánuje. Na zbylé segmenty se musí algoritmus spustit znova.

```
segments ← countSegments(W);
s_0 ← getStateFromWaypoint(w_0);
p ← smoothPath(s_0, getEndWaypoint(s_0));
if not IsIntersecting(p, zone) then
    s_0.smoothToEndUsable ← true;
end
OPEN ← {s_0};
CLOSED ← {};
while not isEmpty(OPEN) do
    s_cur ← removeTheBest(OPEN);
    insert s_cur into CLOSED;
    if s_cur.smoothToEndUsable then
        w_segment ← getEndWaypoint(s_cur);
        s_tmp ← s_cur;
        p ← smoothPath(s_cur, w_segment);
        s_cur ← getEndState(p);
        updateHeuristic(s_cur);
```

```
s_cur.lastInSegment ← true;
if s_cur.segment = segments then
    return ReconstructPath(s_cur);
else
    w_segment+1 ← getEndWaypoint(s_cur);
    h ← smoothPath(s_cur, w_segment+1);
    if not IsIntersecting(h, zone) then
        s_cur.smoothToEndIsUsable ← true;
        updateHeuristic(s_cur);
        s_cur.lastInSegment ← true;
        if not contains(s_cur, CLOSED) then
            insert s_cur into OPEN;
        end
    end
    if isTimeConstrained(w_segment) then
        p ← minimalizeDuration(p);
        t_min ← getEndTime(p);
        p ← maximizeDuration(p);
        t_max ← getEndTime(p);
    else
        t_min ← getMinTime(w_segment);
        t_max ← getMaxTime(w_segment);
    end
    for t from t_min to t_max step 2Δtime do
        p ← smoothPath(s_tmp, w_segment, t);
        s_cur ← getEndState(p);
        h ← smoothPath(s_cur, w_segment+1);
        if not IsIntersecting(h, zone) then
            s_cur.smoothToEndIsUsable ← true;
            updateHeuristic(s_cur);
            s_cur.lastInSegment ← true;
            if not contains(s_cur, CLOSED) then
                insert s_cur into OPEN;
            end
        end
```

```
            end
            continue;
        end
    end
    foreach m ∈ Expand(s_cur) do
        quickest ← false;
        do
            s_new ← getEndState(m);
            if not contains(s_cur, CLOSED) then
                continue;
            if IsIntersecting(m, zone) then
                continue;
            h ← smoothPath(s_new, getEndWaypoint(s_new));
            if not IsIntersecting(h, zone) then
                s_new.smoothToEndIsUsable ← true;
            updateHeuristic(s_new);
            if not quickest
                p ← findSmoothPath(s_new);
                if p ≠ null then
                    s_new ← getEndState(p);
                    p ← minimalizeDuration(p);
                    quickest ← true;
                end
            else
                quickest ← false;
            InsertOrReplaceIfBetter(s_new, OPEN);
        while quickest
    end
end
return null;
```

Zde jsou použity funkce, jejichž význam je následující:

- **countSegments** - vrací, pro kolik segmentů se najednou plánuje.
- **createStateFromSegment** - vytvoří počáteční stav, podle startovního bodu.

- **smoothPath** - vrací analytickou trasu ze stavu do cílového bodu.
- **getEndWaypoint** - vrátí ke stavu cílový bod segmentu, ke kterému stav náleží.
- **IsIntersecting** - vrací, zda manévr prochází bezletovou zónou.
- **removeTheBest** - odejme a vrací ze seznamu stav, který má minimální ohodnocení f. V případě, že je jich více, vrací vybíra jenom jeden.
- **getEndState** - vrací koncový stav manévru.
- **updateHeuristic** - vypočítá heuristiku daného stavu a nastaví tak parametr h stavu.
- **ReconstructPath** - podle cílového stavu se výtvoří letový plan, který se vrací v podobě sekvence letových manévrů.
- **contains** - vrací, zda je stav v seznamu již obsažen.
- **isTimeConstrained** - zjišťije, jestli je daný cílový bod časově omezen.
- **minimalizeDuration** - nastaví zrychlení analytické trasy tak, aby byla doba jejího trvání minimální.
- **maximizeDuration** - nastaví zrychlení analytické trasy tak, aby byla doba jejího trvání maximální a neobsahovala čekací smyčky.
- **getEndTime** - vrací čas na konci manévru.
- **getMinTime** - je-li cílový bod časově intervalem $\langle t_{min}, t_{max} \rangle$, vrací hodnotu t_{min}. Jinak vrací přímo hodnotu časového omezení.
- **getMaxTime** - je-li cílový bod časově intervalem $\langle t_{min}, t_{max} \rangle$, vrací hodnotu t_{max}. Jinak vrací přímo hodnotu časového omezení.
- **Expand** - generuje všechny možné manévry z daného stavu. V nastavením plánovače je definováno, jaké manévry se generují.
- **findSmoothPath** - zkusí najít kratší trasy do daného stavu, jestliže se trasa dá zkrátit, vrátí se analytická trasa, kterou se do stavu dostane, jinak se vrátí null.
- **InsertOrReplaceIfBetter** - vkládá do seznamu stav. Je-li stav v seznamu obsažen, ale má větší ohodnocení f, stav se nahradí novým. V opačném případě se vkládaný stav zahodí.

Kapitola 4

Implementace

Systém AgentFly je psaný v prostředí Java. Tato kapitola popisuje třídy využívané k časo-prostorovýmu plánování. Je zde ukázáno, kde probíhá plánování, jak jsou reprezentovány stavy a letové manévry.

4.1 Třída FlightPlanWrapper4D

Třída FlightPlanWrapper4D je hlavním prvkem časo-prostorového plánovače, který řeší, jak se pro jednotlivé segmenty bude plánovat. Je rozšířením třídy FlightPlanWrapperWithZones a jejím ekvivalentem původního plánování je třída FlightPlanWrapperFromManeuvers. Nachází se v balíku *atc.plane.pilot.planning*. Její součástí jsou dvě důležité funkce, *planProcess* a *createElementsOnSegment*.

4.1.1 Funkce planProcess

Jejím vstupem je první segment, který zároveň nese ukazatele na následující segmenty. Nejprve se vypočítají směrové vektory k cílovým bodům segmentů, viz [4] a poté se spustí plánovací procedura, která pro každý segment volá funkci *createElementsOnSegment*.

4.1.2 Funkce createElementsOnSegment

Tato funkce má za úkol ke každému vytvořenému segmentu, který je vstupem do funkce, vytvořit letový plán. Volá se zde funkce *planPath* třídy PathFinder4D, která vytvoří

letový plán pro daný segment, ale i také pro následující segmenty v případě, že se pro ně musí najednou plánovat. Pro tyto následující segmenty se už znova funce *planPath* nevolá.

4.2 Reprezentace stavů

Pro stavy je nadefinovaná třída State4D, která vznikla ze třídy State rozšířením o čas a rychlost. Celý stav se tak rozlišuje podle veličin:

- **Poloha** - umístění stavu.
- **Směrový vektor** - směr, jakým se do stavu dostalo.
- **Čas** - v jaký časový okamžik se stavem proletí, měřeno absolutně od počátku.
- **Rychlost** - jakou bude mít letadlo v dánem stavu rychlost.

Dále obsahuje parametry popsáné v sekci 3.3.2. Kvůli optimalizaci vyhledávání má stav definované tzv. *hash* kódy, které urychlují hledání stavů se zdiskretizovaně stejnou pozicí. To urychluje ověření, zda je stav nachází v seznamu.

4.2.1 Funkce getState

Hlavní význam této statické funkce je vytvořit stav a nastavit mu jeho parametry podle vstupních hodnod. Vždy nastavuje hlavní proměnné jako je poloha, směrový vektor, čas a rychlost. Je-li vstupem předchůdce a hodnosta *stepG*, nastaví hodnotu g jako součet hodnoty g předchůdce a *stepG*. Jako vstup pro *stepG* se používá *fitness* funkce manévrů. Nastaví se parametr *segment* na stejnou hodnotu jako má předchůdce a má-li předcházející stav nastaveno $lastIsSegment$, navýší se parametr *segment* o jedna.

4.2.2 Funkce detectPointLevelAndUpdateHash

Tato funkce počítá *hash* kódy, které se využívají k rychlemu vyhledání stavů s blízkou pozicí. Kódy jsou vytvářeny na základě vzdálenosti stavu od nejbližší překážky a na rozměrech letového prostoru.

4.3 Reprezentace manévrů

Každý z letových manévrů je reprezentován třídou. Základem pro všechny manévry je třída Maneuver4D, která se nachází v balíku *atc.planner4D*. Její ekvivalent v původním plánování je třída Maneuver. Liší se od ní tím, že využívá rozšířené stavy o čas a rychlost a má nadefinované funkce pro práci s časem a rychlostí.

- **getStartTime** - vrací čas zahájení manévru.
- **getDuration** - vrací dobu trvání manévru.
- **getEndTime** - vrací čas na konci manévru.
- **getStartVelocity** - vrací rychlost na začátku manévru.
- **getFinalVelocity** - vrací rychlost na konci manévru.
- **getFitness** - vrací ohodnocení manévru podle rovnice 3.1.
- **getEndState** - vrací koncový stav manévru, který má předchůdce počáteční stav, navýšenou cenu g o hodnotu $fitness$ funkce a nastavenou informaci o typu manévru, kterým se do koncového stavu dostalo.

V této třídě jsou také definované váhy w_{length}, $w_{penalty}$, w_{time} a $w_{velocity}$ pro stanovení *fitness* funkce.

Každý manévr obsahuje dvě důležité funkce, *initialize* a *initializeForFinalState*.

4.3.1 Funkce initialize

Účelem této funkce je nastavit manévr podle vstupních parametrů. Vstupním paramentrem je vždy počateční stav. Tím je pak určen cílový stav manévru.

4.3.2 Funkce initializeForFinalState

Vstupem je počáteční a koncový stav. Tato funkce je zvláštním případem inicializace, kde se manévr nastaví tak, aby začínal v počátečním stavu a končil v koncovém stavu. To je důležité při vytváření plánu z posloupnosti stavů, kde je jeden předchůdce druhého.

4.3.3 Třída StraightManeuver4D

Třída StraightManeuver4D reprezentuje přímý let letadla. Pro nastavení tohoto manévru je důležitý počáteční stav a délka manévru. Během přímého letu je možno zrychlovat. Zrychlování se provádí funkcí *setAcceleration*, která má vstup velikost zrychlení a dráhu po kterou se bude zrychlovat. Opakovaným voláním se nastavuje úsek za předchozím úsekem, čili se celý manévr může rozdělit na části s různým zrychlením. Zbylý úsek, pro který není nadefinované zrychlení, má zrychlení nulové. Postupně se po jednotlivých úsecích vypočítává koncová rychlost až pro poslední úsek se zjistí koncová rychlost celého manévru. Stejným způsobem se určí doba trvání jednotlivých úseků a tím i celková doba trvání manévru a koncový čas.

4.3.4 Třída TurnManeuver4D

Manévr zatočení je reprezentován touto třídou. Modeluje jak horizontální zatáčení, tak vertikální stoupání nebo klesání. Jelikož je rychlost konstantní během manévru, je jeho koncová rychlost stejná jako počáteční. Pro definování manévru je nutný počáteční stav, typ změny směru, směr změny a úhel. Typ změny může být horizontální, nebo vertikální. V případě horizontální změny je směr doprava, nebo doleva a u vertikální změny je směr nahoru, nebo dolu.

4.3.4.1 Třída PitchToLevelManeuver4D

Speciálním případem vertikální změny směru je vyrovnání letadla do horizontální roviny. Tento letový manévr reprezentuje třída PitchToLevelManeuver4D.

4.3.5 Třída SpiralManeuver4D

Touto třídou je charakterizováno spirálové stoupání (klesání). Manévr je určen počátečním stavem, počtem spirál a směrem zatáčení. Speciálním případem je spirálový manévr, který se odehrává v horizontální rovině a má význam čekací smyčky.

4.3.6 Třída ToEndLevelManeuver4D

Manévr, který vyrovná letadlo do horizontální roviny ve výšce cílového bodu, je implementován ve třídě ToEndLevelManeuver4D. Skládá se ze dvou částí, z rovného úseku

a vyrovnání do horizontální roviny. Jelikož se skládá pouze z těchto dvou částí, nemusí být pokaždé realizovatelný. Proto při inicializaci manévru se vrací, zda je možné tento manévr uskutečnit.

4.3.7 Třída SmoothManeuver4D

Třída SmoothManeuver4D reprezentuje analytickou trasu. Analytická trasa se hledá mezi počátečním stavem a koncovým bodem, který může mít omezení času a rychlosti. Skládá se z předchozích manévrů, které se hledají původním algoritmem, který je ve třídě SmoothManeuver. Pro rozvržení rychlosti během této trasy jsou k dispozici funkce *flyWithVelocity* a *timePlanning*.

4.3.7.1 Funkce flyWithVelocity

Vstupem funkce je požadovaná rychlost, na kterou se zrychlí a poté se letí už touto rychlostí. Je-li počáteční rychlost menší než požadovaná rychlost, použije se maximální zrychlení, jinak se použije maximální zpomalení. Ve funkci je ošetřen případ, kdy se v průběhu rovného úseku nestihne zrychlit na požadovanou rychlost.

4.3.7.2 Funkce minimalDuration

Nastaví zrychlení tak, aby celková doba trvání manévru byla minimální. Používá funkci *flyWithVelocity*, kde je vstupní parametr roven maximální rychlosti letadla. Jak funkce funguje je popsáno v sekci 3.2.5.

4.3.7.3 Funkce maximalDuration

Nastaví zrychlení tak, aby celková doba trvání manévru byla maximální tak, aby trasa neobsahovala čekací smyčky. Používá funkci *flyWithVelocity*, kde je vstupním parametrem minimální rychlost letadla. Jak funkce funguje je popsáno v sekci 3.2.6.

4.3.7.4 Funkce timePlanning

Cílem této funkce je nastavení zrychlení tak, aby letadlo proletělo koncovým bodem přesně v požadovaný čas, který je vstupem do funkce. Během plánování se uvažuje omezení na koncovou rychlost. V případě, že se to nepodaří omezení dodržet, penalizuje se to ve *fitness* funkci. Rozvržení zrychlení je popsáno v sekci 3.2.

4.3.7.5 Funkce getFitness

Ohodnocovací funkce je oproti ostatním manévrům penalizována i nesplněním omezujících podmínek podle rovnice 3.50.

4.4 Třída PathFinder4D

Prostorové prohledávání popsané v sekci 3.3 je implementováno ve funkci *planPath*. Vstupem funkce je segment, který ukazuje na následující segmenty a objekt reprezentující bezletové zóny. V prohledávání se používá funkce *getHeuristics*, která spočítá heuristiku daného stavu a funkce *getEndWaypoint*, která vrací aktuální cílový bod ke stavu.

4.4.1 Třída SearchQueue4D

V prohledávání se využívá seznam *Open* a *Closed*. Tyto seznamy jsou reprezentovány třídou SearchQueue4D, která vznikla podle původní třídy SearchQueue. Navíc se zde stavy rozlišují podle času a podle rychlosti. Pro rozlišení jsou v této třídě nadefinované parametry $\Delta time$ a $\Delta velocity$. Seznam je uchováván v podobě binárního stromu, tím se zvyšuje efektivita prohledávání. Třída obsahuje důležité funkce pro práci se seznamem.

4.4.1.1 Funkce contains

Vstupem do funkce je stav a funkce vrací, zda je v seznamu či nikoliv. Hledání probíhá podle *hash* kódu stavu, tím se rychle najdou stavy, které se nachazejí přibližně ve stejné lokaci. Pak se jenom ověří, zda odchylky vzdáleností poloh, směrových vektorů, časů a rychlostí nepřesahujou toleranční meze.

4.4.1.2 Funkce insert

Tato funkce vkládá do seznamu stav S. V případě, že se již stejný stav v seznamu s lepším ohodnocením f, vrátí stav S a do seznamu se nevloží. Jestliže je stav v seznamu obsažen, ale nemá lepší ohodnocení f, nahradí se stavem S a vrátí se původní stav. Jestliže se v seznamu nenachází, vloží se a vrátí se hodnota *null*.

4.4.1.3 Funkce removeTheBest

Účelem funkce removeTheBest je odejmutí stavu, který má nejlepší ohodnocení. To je takový stav, který má nejmenší hodnotu f. Tento stav je zároveň návratovou hodnotou funkce. Nejlepší stav je uchováván v kořenu binárního stromu.

Kapitola 5

Testování a porovnání plánovačů

Tato kapitola zkoumá výkon časo-prostorového plánovače a testuje, jak je schopný zvládnout složitější situace. Je zde testováno, jak se hledá analytická trasa včetně zrychlením při časovém a rychlostním omezení, dále jak funguje plánování přes více segmentů a jak je plánovač schopen hledat trasu kolem překážek. Ke konci kapitoly je pro zkoumáno, co může ovlinit dobu hledání plánu. Veškeré testování je prováděno na procesoru Intel Core 2 Quad 2,4GHz s operační pamětí 6GB DDR2 800MHz pod 64 bitovým operačním systémem Windows Vista. Při testování je využívám vizualizační prvek systému AgentFly, který zobrazuje průběh letu. Zde je popsán význam následujicích symbolů:

- **Červený puntík** ● - cílový bod.
- **Tmavě modrá trajektorie** — - přímý let bez zrychlení.
- **Silná světle modrá trajektorie** — - přímý let s kladným zrychlením.
- **Tenká světle modrá trajektorie** — - přímý let se záporným zrychlením.
- **Světle zelená trajektorie** — - zatáčecí manévr.
- **Červená kružnice** — - spirálový manévr.
- **Modrá čára** — - překážka, rovinná bezletová zóna.
- **Šedá plocha kolem letadla** - vymezuje prostor kolem letadla, do kterého se nesmí dostat jiný objekt.

5.1 Nastavení plánovače

Plánovač je testován na letadle, které má následující parametry:

- **Optimální rychlost** - $v_{opt} = 0.3$.
- **Minimální rychlost** - $v_{min} = 0.1$.
- **Maximální rychlost** - $v_{max} = 0.5$.
- **Maximální rychlení** - $a_{max} = 0.01$.
- **Maximální zpomalení** - $a_{min} = -0.01$.

Nastavení vah pro *fitness* funkci:

- **Váha délky** - $w_{length} = 1$.
- **Váha penalizace za neoptimální rychlost** - $w_{penalty} = 0.8$.
- **Váha odchylky od požadovaného času** - $w_{time} = 10$.
- **Váha odchylky od požadované rychlost** - $w_{velocity} = 10$.

Rozlišení stavů diskretizací:

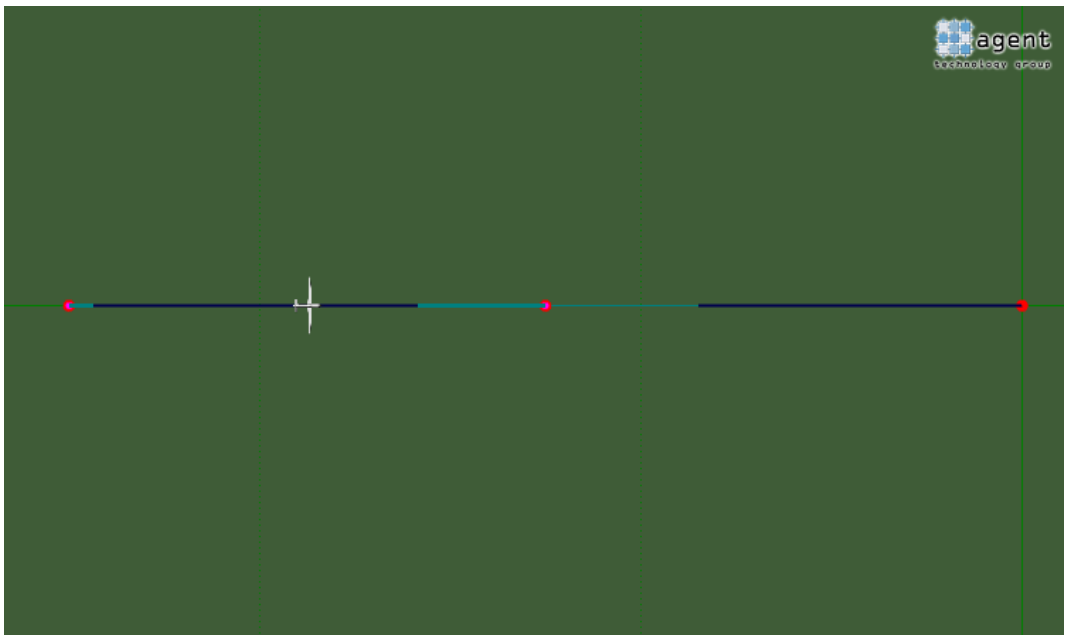
- **Rozlišení podle času** - $\Delta time = 5$.
- **Rozlišení podle rychlosti** - $\Delta velocity = 0.1$.

5.2 Testování analytické trasy

Tato sekce je zaměřená na testování analytické trasy, to znamená, že se zde nevyskytují překážky.

5.2.1 Plánování zvlášť po segmentech

První situace zkouší hledání analytické trasy na dvou segmentech. První segment je na konci časově a rychlostně omezen a druhý segment je bez omezení. Startovní bod má pozici $(300, 750, 500)$ a startuje se v čase 0 s. Prvním cílovým bodem musí proletět v

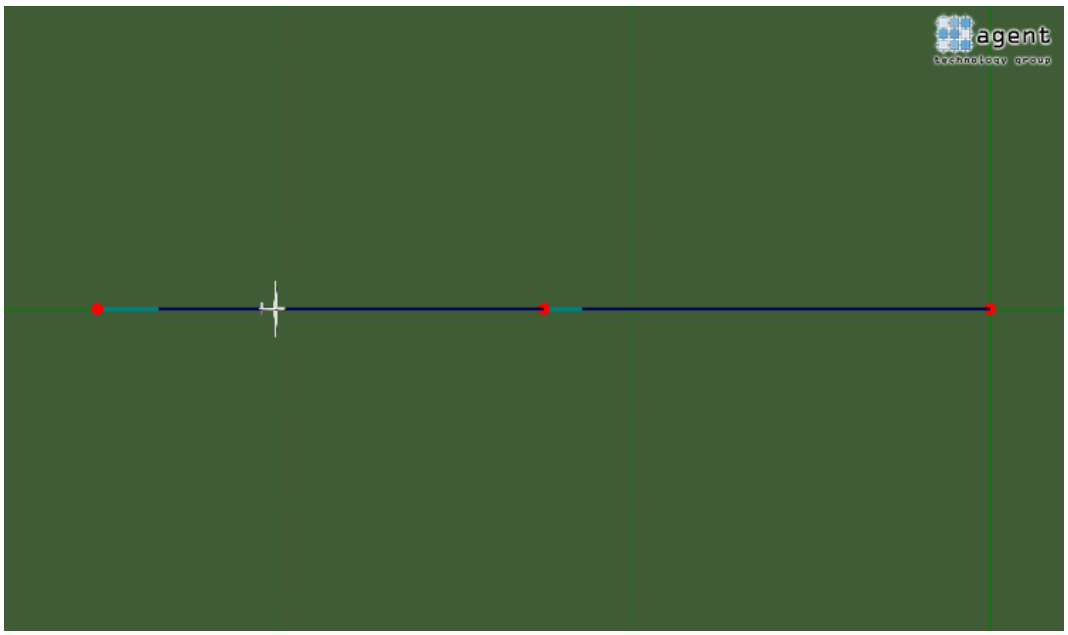


Obrázek 5.1: Analytická trasa pro jeden segment časově a rychostně omezený a pro druhý segment bez omezení.

dobu 70s maximální rychlostí. Cílový bod má pozici $(325, 750, 500)$. Poslední cílový bod je bez omezení a je umístěn v $(350, 750, 500)$. Obrázek 5.1 zobrazuje tuto situaci s již naplánovanou cestou. Je zde vidět, že letadlo rychluje, aby stihlo časové omezení a před koncem prvního cílového bodu zrychlí na maximální rychlost. Jelikož je další cílový bod bez omezení, letadlo zase zpomalí na optimální rychlost.

Výpis letového plánu:

```
StraightElement: p1=(300.0, 750.0, 500.0)
                 p2=(301.28, 750.0, 500.0)
                 accel=0.01
                 length=1.2799999999999727
StraightElement: p1=(301.28, 750.0, 500.0)
                 p2=(318.28, 750.0, 500.0)
                 accel=0.0
                 length=17.0
StraightElement: p1=(318.28, 750.0, 500.0)
                 p2=(325.0, 750.0, 500.0)
                 accel=0.01
```



Obrázek 5.2: Plánování přes dva segmenty analytickou trasou.

```
                  length=6.720000000000027
StraightElement: p1=(325.0, 750.0, 500.0)
                  p2=(333.0, 750.0, 500.0)
                  accel=-0.01
                  length=8.0
StraightElement: p1=(333.0, 750.0, 500.0)
                  p2=(350.0, 750.0, 500.0)
                  accel=0.0
                  length=17.0
```

5.2.2 Plánování přes více segmentů

Následující situace je navržena tak, aby plánovač musel plánovat přes více segmentů najednou. Startovní bod a cílové body jsou stejné jako v předchozím případě s tím rozdílem, že první cílový bod je bez omezení a druhý cílový bod se musí proletět v čase 120 s. Tato situace je zachycena na obrázku 5.2, kde je vidět, že zrychlení dochází už v prvním segmentu, ikdyž zde není časové omezení.

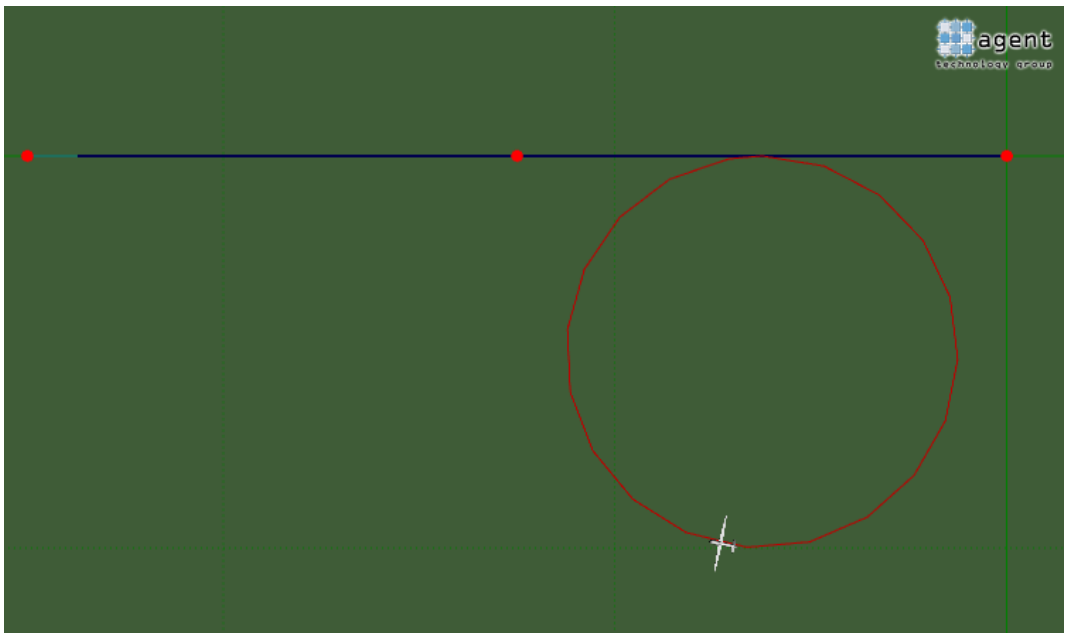
Výpis letového plánu:

```
StraightElement: p1=(300.0, 750.0, 500.0)
                 p2=(303.4262233876183, 750.0, 500.0)
                 accel=0.01
                 length=3.426223387618279
StraightElement: p1=(303.4262233876183, 750.0, 500.0)
                 p2=(325.0, 750.0, 500.0)
                 accel=0.0
                 length=21.57377661238172
StraightElement: p1=(325.0, 750.0, 500.0)
                 p2=(327.14085301433335, 750.0, 500.0)
                 accel=0.01
                 length=2.140853014333345
StraightElement: p1=(327.14085301433335, 750.0, 500.0)
                 p2=(350.0, 750.0, 500.0)
                 accel=0.0
                 length=22.859146985666655
```

Změní-li se požadavek u posledního cílového bodu tak, aby jím letadlo proletělo v čase 550 s, je nutné do trasy vložit časovou smyčku. Na obrázku 5.3 je zobrazena tato naplánovaná trasa. Je zde patrné, že hned v prvním segmentu dochází ke snížení rychlosti a v dalším segmentu je vložen jeden okruh smyčky.

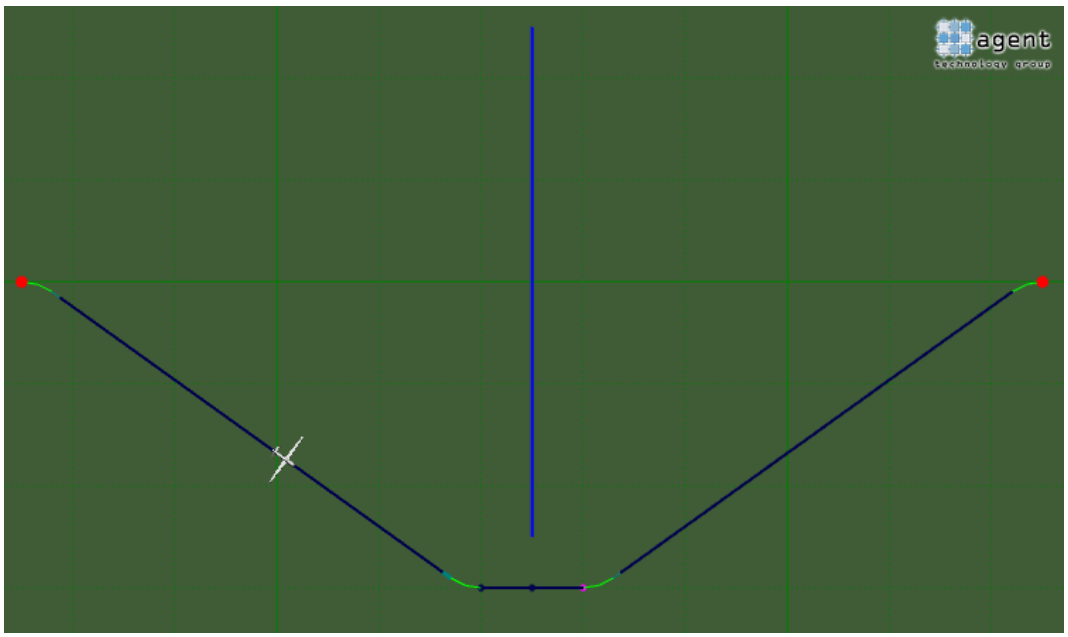
Výpis letového plánu:

```
StraightElement: p1=(300.0, 750.0, 500.0)
                 p2=(302.55234540500925, 750.0, 500.0)
                 accel=-0.01
                 length=2.5523454050092482
StraightElement: p1=(302.55234540500925, 750.0, 500.0)
                 p2=(325.0, 750.0, 500.0)
                 accel=0.0
                 length=22.44765459499075
StraightElement: p1=(325.0, 750.0, 500.0)
                 p2=(325.17800333655543, 750.0, 500.0)
                 accel=0.01
                 length=0.17800333655543454
StraightElement: p1=(325.17800333655543, 750.0, 500.0)
```



Obrázek 5.3: Plánování přes dva segmenty analytickou trasou s nutnou časovou smyčkou.

```
                p2=(337.5, 750.0, 500.0)
                accel=0.0
                length=12.321996663444565
SpiralElement: center=(337.5, 740.0, 500.0)
               u=(-0.0, 10.0, 0.0)
               v=(10.0, 0.0, 0.0)
               radius=10.0
               angle=6.283185307179586
               elevation=0.0
StraightElement: p1=(337.5, 750.0, 500.0)
                 p2=(350.0, 750.0, 500.0)
                 accel=0.0
                 length=12.5
```



Obrázek 5.4: Plánování trasy kolem překážky.

5.3 Hledání trasy kolem překážky

Tato část zkoumá hledání trasy kolem překážek. Zároveň se testuje rychlost plánovače při plánování přes více překážek.

5.3.1 Plánování v jednom segmentu s překážkou

Tento scénář je nastaven tak, aby se musela trasa hledat okolo překážky. Letadlo letí z bodu (300, 750, 500) do bodu (500, 750, 500), kde uprostřed je překážka o délce 100. V cílovém bodě má proletět v čase 1000 s. Obrázek 5.4 ukazuje danou situaci s naplánovanou trasou.

Trasa byla nalezena za 191 ms a její délka je 287,6. Celkem bylo zpracováno 110 stavů ze seznamu *Open* a 200 stavů v něm zůstalo.

Na ukázku je výpsán letový plán této složitější situace:

```
TurnElement: center=(300.0, 740.0, 500.0)
             u=(-0.0, 10.0, 0.0)
             v=(10.0, 0.0, 0.0)
             radius=10.0
             angle=0.6227157804858658
```

```
            isHorizontal=true
StraightElement: p1=(305.8324333035022, 748.1229749328801, 500.0)
                 p2=(307.4688152612176, 746.9480250276678, 500.0)
                 accel=-0.01
                 length=2.0145106580246517
StraightElement: p1=(307.4688152612176, 746.9480250276678, 500.0)
                 p2=(382.5311847387824, 693.0519749723323, 500.0)
                 accel=0.0
                 length=92.40748629388024
StraightElement: p1=(382.5311847387824, 693.0519749723323, 500.0)
                 p2=(384.1675666964978, 691.87702506712, 500.0)
                 accel=0.01
                 length=2.0145106580246517
TurnElement: center=(390.0, 700.0, 500.0)
             u=(-5.832433303502249, -8.122974932880062, 0.0)
             v=(8.122974932880062, -5.832433303502249, 0.0)
             radius=10.0
             angle=0.6227157804858658
             isHorizontal=true
StraightElement: p1=(390.0, 690.0, 500.0)
                 p2=(400.0, 690.0, 500.0)
                 accel=0.0
                 length=10.0
StraightElement: p1=(400.0, 690.0, 500.0)
                 p2=(410.0, 690.0, 500.0)
                 accel=0.0
                 length=10.0
TurnElement: center=(410.0, 700.0, 500.0)
             u=(0.0, -10.0, 0.0)
             v=(10.0, 0.0, 0.0)
             radius=10.0
             angle=0.6227157804858658
             isHorizontal=true
StraightElement: p1=(415.8324333035022, 691.8770250671199, 500.0)
                 p2=(417.2983991301455, 692.9296133096985, 500.0)
```

```
                accel=-0.01
                length=1.804715438317277
StraightElement: p1=(417.2983991301455, 692.9296133096985, 500.0)
                p2=(494.1675666964977, 748.12297493288, 500.0)
                accel=0.0
                length=94.63179217161222
TurnElement: center=(500.0, 740.0, 500.0)
             u=(-5.832433303502249, 8.122974932880062, 0.0)
             v=(8.122974932880062, 5.832433303502249, 0.0)
             radius=10.0
             angle=0.6227157804858658
             isHorizontal=true
```

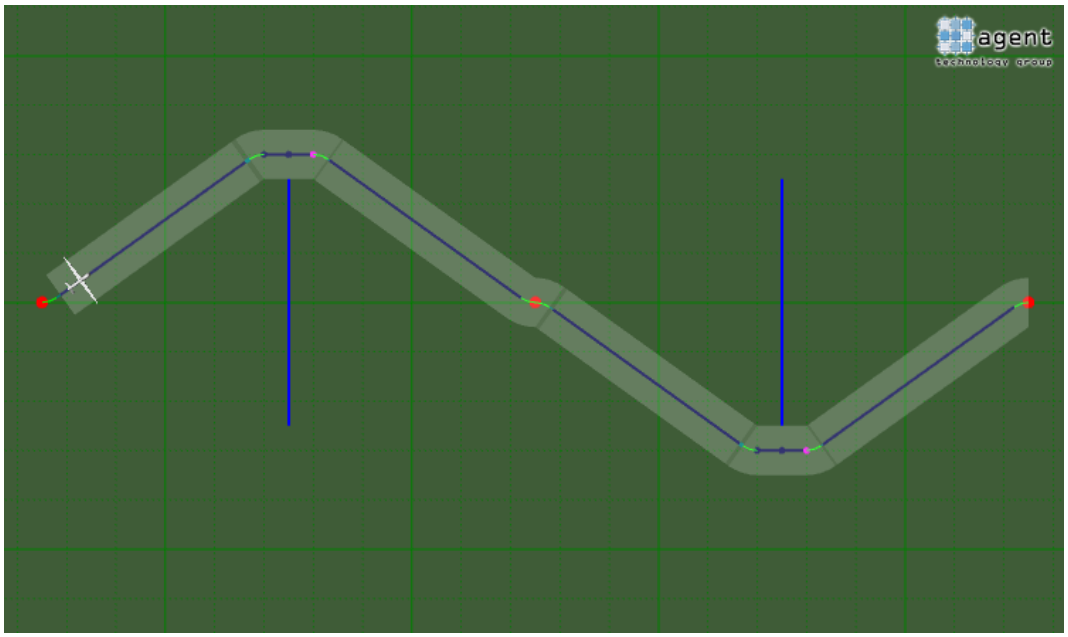
5.3.2 Plánování přes více segmentů s překážkou

Uvažujme předchozí situaci, kde cílový bod bude bez omezení času a přidá se další cílový bod umístěným v (700, 750, 500) a o další překážka umístěná podle obrázku 5.5. Posledním cílovým bodem se má proletět v čase 2000 s. Na obrázku je zároveň vidět zóna kolem letadla, která nesmí zasahovat do bezletových zón. Plán byl nalezen v čase 1646 ms a délka trasy je 575,11. Celkem bylo zpracováno 3458 stavů ze seznamu *Open* a 3982 stavů v něm zůstalo.

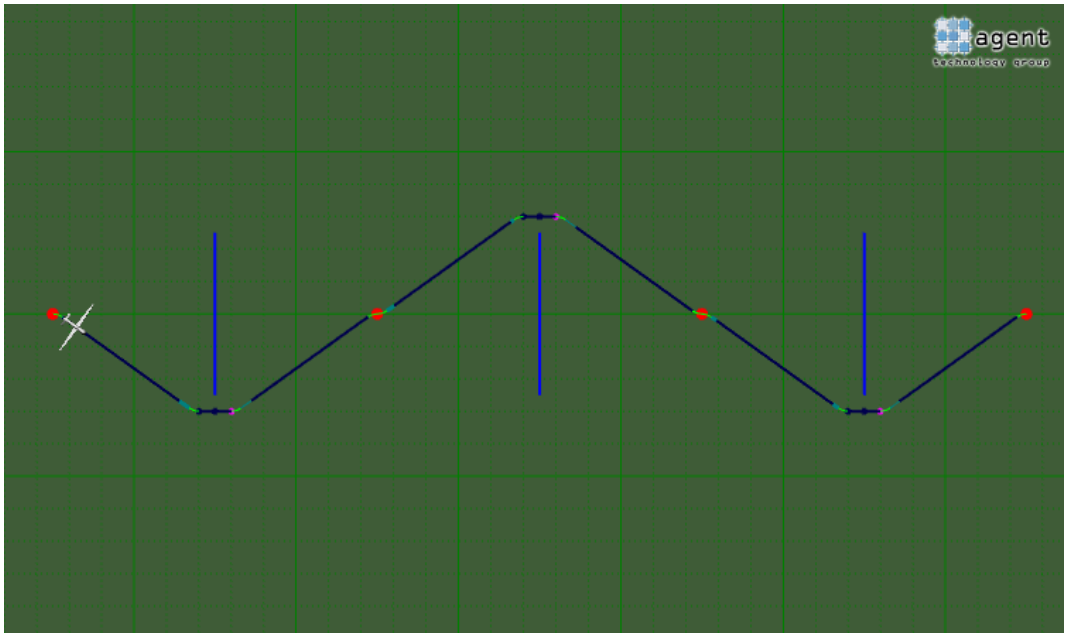
Přidáním třetího cílového bodu umístěného v (900, 750, 500) se získá scénař, který je na obrázku 5.6. Opět je ve středu posledního segmentu vložena překážka. V cílovém bodě má letadlo být v čase 3000 s a předchozí cílové body jsou bez omezení, proto se plánuje přes všechny tři segmenty najednou. Pro tuto složitější situaci byl plán nalezen za dobu 11032 ms. Délka nalezené trasy je 804.02. Celkem bylo zpracováno 23028 stavů ze seznamu *Open* a 7533 stavů v něm zůstalo.

5.4 Rychlost hledání trasy v závislosti na velikosti překážky

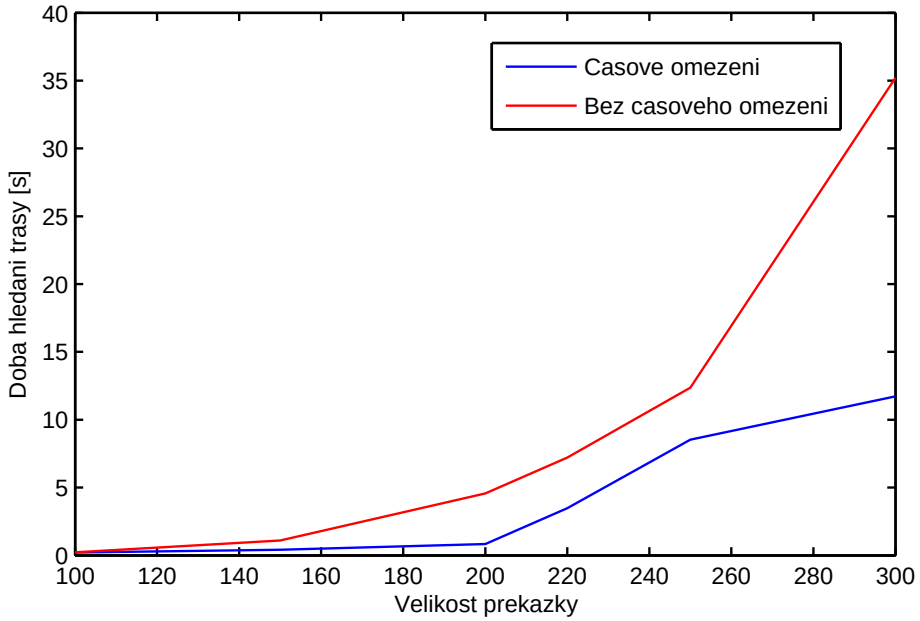
Tato sekce zkoumá jak ovlivňuje velikost překážky rychlost hledání plánu. Letadlo má v plánu letět z bodu (300, 750, 500) do bodu (500, 750, 500). Do středu trasy je vložena



Obrázek 5.5: Plánování trasy kolem překážkek přes dva segmenty.



Obrázek 5.6: Plánování trasy kolem překážkek přes tři segmenty.

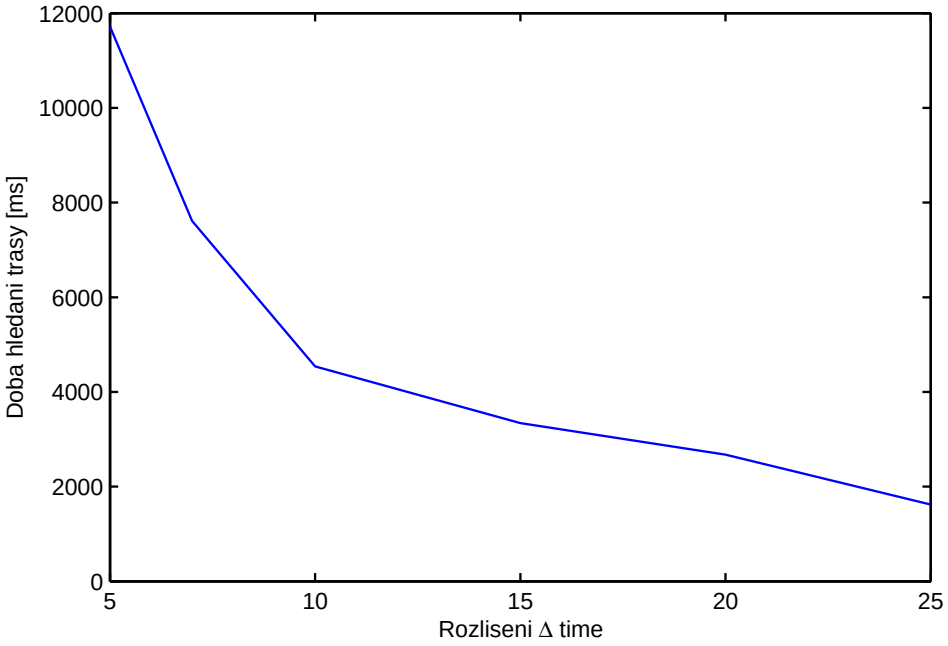


Obrázek 5.7: Graf závislosti doby plánování na velikosti překážky.

překážka a zkoumá se rychlost plánovače v závislosti na velikosti této překážky. Na obrázku 5.7 je zobrazen graf zkoumané závislosti. Jsou testovány případy, kde je cílový bod bez omezení a s časovým omezením. Pro omezení platí, že má letadlo proletět cílovým bodem v čase 1000 s. Je vidět, že se zvětšující se překážkou prudce stoupá doba plánování. Je-li cílový bod bez omezení, plánování trvá delší dobu než s omezením. Původní algoritmus dvoufázevého plánování je podstatně rychlejší. Najde trasu přibližně kolem 200 ms pro všechny velikosti překážek.

5.5 Rychlost hledání trasy v závislosti na diskretizaci stavu podle času

Parametr rozlišení stavu podle času $\Delta time$ má významnou roli při prohledávání celého prostoru. Snižováním se přibližuje k optimálnějšímu řešení na úkor výpočetní doby. Využitím přechozí situace, kde je velikost překážky 300 testuji vliv $\Delta time$ na výpočetní době. Obrázek 5.8 ukazuje graf závislosti doby plánování na rozlišení stavu podle času. Této závislosti se dá využít v optimalizaci výpočtu tak, že by se rozlišení dynamicky



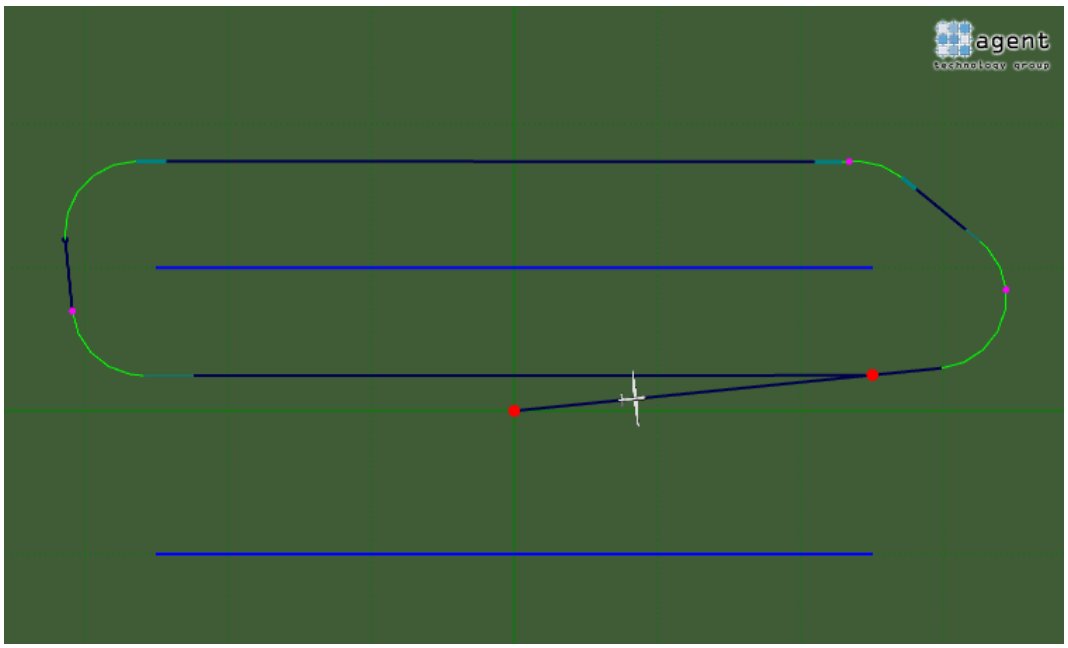
Obrázek 5.8: Graf závislosti doby plánování na rozlišení stavu podle času.

měnilo například v závislosti vzálenosti počátečního a cílového bodu, nebo v závislosti na časovém rozdílu obou bodů.

5.6 Nesplnitelná situace původním plánovačem

Zkoušel jsem jednoduchou situaci, kde letadlo letí z bodu (300, 750, 500) do bodu (400, 755, 500). Mezi bodama není žádná překážka, ale oba body jsou umístěné mezi překážkama. Není-li cílový bod časově omezen, oba plánovače fungují správně. Problém nastává když se cílový časově omezí tak, že se musí vkládat časová smyčka, která už prochází překážkama. V tomto připadě původní plánování selže a letadlo letí přímo do cíle, kde bude předčasně. Nový plánovač dokáže naplánovat trasu tak, aby letadlo proletělo cílovým bodem v požadovanou dobu. Obrázek 5.9 zobrazuje náplanovanou trasu časo-prostorovým plánovačem. Z obrázku je patrné, že je algoritmus schopen najít takovou časovou smyčku, která se vyhne překážkám a zároveň letadlo dorazí do cíle ve správnou dobu.

Doba hledání této trasy trvala 8996 ms. Celkem bylo zpracováno 20234 stavů ze



Obrázek 5.9: Situace, kde funguje pouze časo-prostorový plánovač. Plánovač si vytvoří čekací smyčku kolem překážky, aby letadlo nedorazilo předčasně.

seznamu *Open* a 5550 stavů v něm zůstalo. Odtud je vidět, že se prohledává relativně velká část stavového prostoru, přestože do cílového bodu vede cesta přímo. Tato situace je pro časo-prostorový plánovač poměrně náročná vzhledem k době nalezení letového plánu, nicméně se oproti původnímu plánovači splní požadavek časového omezení.

Kapitola 6

Závěr

Cílem práce bylo prozkoumat původní dvoufázový přístup k plánování v prostředí AgentFly a navrhnout jeho změnu tak, aby plánování probíhalo najednou. Původní plánovač byl prozkoumán a na jeho základě byl vyvinut nový časo-prostorový plánovač.

Byly identifikovány nedostatky původního řešení, které nejprve hledá nejkratší trasu a poté rozvrhuje rychlost letadla. Problém tohoto přístupu nastává v případě, že pro splnění časového omezení je nejkratší trasa nevhodná a tento plánovač si s tím nedokáže poradit. Naopak časo-prostorový plánovač tyto situace zvládne, protože trasu hledá tak, aby rovnou bylo splněno časové a rychlostí omezení. Původní plánovač si nedokáže poradit s veličinama zavislých na čase. Například kdyby maximální rychlost letadla závisela na nadmořské výšce, algoritmus by s tim neuměl pracovat, protože během časového plánování se vůbec neuvažuje pozice. V případě, že by se v letovém prostoru vyskytovaly bezletové zóny, které by platily pouze v určitou dobu, původní algoritmus by opět selhal. Časo-prostorový plánovač by úpravou s těmoto zónami uměl přirozeně zacházet, jelikož ví v jakém čase je v daném bodě. Další výhodnou nové metody je, že hledaná trasa je přímo penalizována za neoptimální rychlost, tím se trasa optimalizuje tak, aby letadlo mělo minimální spotřebu. Penalizace se násobí parametrem plánovače, takže se může hledat taková trasa, kde se poletí rychlostí co nejbližší k optimální rychlosti.

Jelikož časo-prostorový plánovač vznikl rozšířením původního o dimenzi času, ale i rychlosti, několika násobně se zvětšuje jeho prohledávácí prostor. Tím podává o poznání horší výsledky, co se týče doby hledání letového plánu. Vypočetní doba záleží na rozlišení stavů podle času, které mají stejnou pozici. Tato rozlišovací schopnost je nastavitelným parametrem plánovače, který se během plánování nemění Toho by se dalo využít v budoucí optimalizaci výpočtu tak, že by se tento parametr dynamicky měnil například podle časové vzdálenosti startovního a cílového bodu.

Literatura

[1] *David Šišlák and Martin Rehák and Michal Pěchouček:* **A-globe: Multi-Agent Platform with Advanced Simulation and Visualization Support.** In Web Intelligence. IEEE Computer Society, 2005. ISBN 0-7695-2415-X.

[2] *David Šišlák and Přemysl Volf and Michal Pěchouček:* **Accelerated A* Path Planning.** In Proceedings of 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2009). 2009.

[3] *David Šišlák and Michal Pěchouček:* **Agent-Based Approach to Free-Flight Planning, Control, and Simulation.** IEEE Intelligent Systems. 2009, vol. 24, p. 14-17.

[4] *David Šišlák and Přemysl Volf and Michal Pěchouček:* **Agent-Based Cooperative Decentralized Airplane Collision Avoidance.** Technical Report, 2008.

[5] *Antonín Komenda:* **Použití multi-agentního systému pro nekooperativní vyhýbání letadel.** Diplomova prace, ČVUT FEL, 2007.

Příloha A

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové AgentFly, potřebné pro navržení a testování nového plánovače. Výpis souborů obsahující zdrojové kódy časo-prostorového plánovače:

- AgentFly/atc/src/atc/planner4D/Maneuver4D.java
- AgentFly/atc/src/atc/planner4D/PathFinder4D.java
- AgentFly/atc/src/atc/planner4D/PathFinder4D2.java
- AgentFly/atc/src/atc/planner4D/PitchToLevelManeuver4D.java
- AgentFly/atc/src/atc/planner4D/SearchQueue4D.java
- AgentFly/atc/src/atc/planner4D/SmoothManeuver4D.java
- AgentFly/atc/src/atc/planner4D/SpiralManeuver4D.java
- AgentFly/atc/src/atc/planner4D/State4D.java
- AgentFly/atc/src/atc/planner4D/StraightManeuver4D.java
- AgentFly/atc/src/atc/planner4D/ToEndLevelManeuver4D.java
- AgentFly/atc/src/atc/planner4D/TurnManeuver4D.java
- AgentFly/atc/src/atc/planner/test/Planner4DTestSetup.java
- AgentFly/atc/src/atc/planner/test/WaypointAndZoneSetup.java

Nastavení plánovače je v souborech:

- AgentFly/atc/xml/__develop/planner_tests/modules4D.xml
- AgentFly/atc/xml/__develop/planner_tests/planner4D.xml
- AgentFly/atc/xml/__develop/planner_tests/scenario4D.xml
- AgentFly/atc/xml/__develop/planner_tests/setup4D.xml