FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Obstacle Avoidance Framework based on Reach Sets

Alojz Gomola


Doutoramento em Matemática Aplicada

Supervisor: Dr. João Tasso de Figueiredo Borges de Sousa
Co-supervisors: Dr. Fernando Manuel Ferreira Lobo Pereira
Dr. Milan Hrusecky

February 14,2019

I declare that I carried out this research plan independently, and only with the cited sources, literature and other professional sources.

In Porto on .....

Alojz Gomola

# Contents

# List of Figures

# List of Tables

# (C)Nomenclature

This chapter summarize used symbols (tab. 3), acronyms (tab. 1), terminology (tab. 4) and, organizations (tab. 2) mentioned in work.

| Acronym | Meaning |
|---|---|
| UAV | Unmanned Aerial Vehicle |
| UAS | Unmanned Autonomous System(including naval vehicles) |
| RPAS | Remotely Piloted Aerial System(lesser degree of autonomy) |
| OPA | Optionally Piloted Aircraft |
| RPV | Remotely Piloted Vehicle |
| PIC | Pilot-in-Command |
| LOS | Line Of Sight |
| VLOS | Visual Line Of Sight |
| BLOS | Behind Line Of Sight |
| SAA | Sense And Avoid |
| DAA | Detect And Avoid |
| MAC | Mid-Air Collision |
| OAC | Off-Air Collision |
| ABSAA | Airborne Sense and Avoid |
| GBSAA | Ground Based Sense and Avoid |
| POA | Preemptive Obstacle Avoidance |
| ROA | Reactive Obstacle Avoidance |
| TCAS | Traffic Alert and Collision Avoidance System |
| ACAS X | Airborne Collision Avoidance System X |
| ACAS XU | Airborne Collision Avoidance System X for UAS |
| CD&R | Collision Detection and Resolution |
| RTK | Real-Time Kinematik |
| GPS | Global Positioning System |
| IMU | Internal Measurement Unit |
| LiDAR | Light Detection and Ranging |
| ADS-B | Automatic Dependent Surveillance – Broadcast |
| GSE | Ground Support Equipment |
| ATC | Air Traffic Control |
| ATO | Air Traffic Organization |
| C2 | Control and Communications |
| MASPS | Minimum Aviation System Performance Standard |
| MOPS | Minimum Operational Performance Standard |
| RVSM | Reduced Vertical Separation Minimum |
| RHSM | Reduced Horizontal Separation Minimum |
| SM | Safety Margin |

Table 1: List of Acronyms

| Acronym | Organization name |
|---|---|
| ICAO | International Civil Aviation Organization (UN) |
| ITU | International Telecommunication Union (UN) |
| EASA | European Aviation Safety Agency (EU) |
| JARUS | Joint Authorities for Regulation of Unmanned Systems (EU) |
| FAA | Federal Aviation Administration (USA) |
| FCC | Federal Communications Commission (USA) |
| LSTS | Laboratório de Sistemas e Tecnologia Subaquática (PT) |
| FEUP | Faculdade de Engenharia da Universidade do Porto (PT) |
| ITK | Institutt for teknisk kybernetikk NTNU (NO) |
| NTNU | Norges teknisk-naturvitenskapelige universitet (NO) |
| IST | Instituto Superior Técnico - Universidade de Lisboa (PT) |
| HWI | Honeywell International (CZ/USA) |

Table 2: List of Organizations

| Symbol | Explanation |
|---|---|
| $A, B, C, D, \dots$ | Capital letters are used for matrices |
| $A(\dots), B(\dots), \dots$ | Functional matrices, $(\dots)$ denotes parameters |
| $f(\dots), g(\dots), \dots$ | Vector or scalar functions $(\dots)$ denotes parameters |
| $\vec{f}(\dots), \vec{g}(\dots), \dots$ | Explicit vector functions, when equation contains both types of scalar and vector functions |
| $t, x, y, z, \dots$ | Vectors or scalar coefficients |
| $\vec{x}, \vec{o}, \vec{g}, \dots$ | Explicit vectors, when function contains both types of scalar and vector parameters. |
| $\theta, \varphi$ | Greek letters denoting angles in radians |

Table 3: List of symbols

| Terminology | Definition |
|---|---|
| Air Traffic Control | A service operated by appropriate authority to promote the safe, orderly, and expeditious flow of air traffic |
| Aircraft | A device that is used or intended to be used for flight in the air |
| Airspace | Any portion of the atmosphere sustaining aircraft flight and which has defined boundaries and specified dimensions. Airspace may be classified as to the specific types of flight allowed, rules of operation, and restrictions in accordance with International Civil Aviation Organization standards or State regulation |
| Civil Aircraft | Aircraft other than public aircraft. |
| Collision Avoidance | The Sense and Avoid system function where the UAS takes appropriate action to prevent an intruder from penetrating the collision volume. Action is expected to be initiated within a relatively short time horizon before closest point of approach. The collision avoidance function engages when all other modes of separation fail. |

| Terminology | Definition |
|---|---|
| Communication Link | The voice or data relay of instructions or information between the UAS pilot and the air traffic controller and other NAS users. |
| Control Station | The equipment used to maintain control, communicate with, guide, or otherwise pilot an unmanned aircraft. |
| Crewmember (UAS) | In addition to the crewmembers identified in 14 CFR Part 1, a UAS flightcrew member includes pilots, sensor/payload operators, and visual observers, but may include other persons as appropriate or required to ensure safe operation of the aircraft. |
| Data Link | A ground-to-air communications system which transmits information via digital coded pulses. |
| Detect and Avoid | Term used instead of Sense and Avoid in the Terms of Reference for RTCA Special Committee 228. This new term has not been defined by RTCA and may be considered to have the same definition as Sense and Avoid when used in this document. |
| ICAO | International Civil Aviation Organization is a specialized agency of the United Nations whose objective is to develop the principles and techniques of international air navigation and to foster planning and development of international civil air transport. |
| Manned Aircraft | Aircraft piloted by a human onboard. |
| Model Aircraft | An unmanned aircraft that is capable of sustained flight in the atmosphere; flown within visual line-of-sight of the person operating the aircraft and flown for hobby or recreational purposes. |
| Optionally Piloted Aircraft | An aircraft that is integrated with UAS technology and still retains the capability of being flown by an onboard pilot using conventional control methods. |
| Pilot-in-Command | Pilot-in-command means the person who:<br>1. has final authority and responsibility for the operation and safety of the flight;<br>2. has been designated as pilot-in-command before or during the flight; and<br>3. holds the appropriate category, class, and type rating, if appropriate, for the conduct of the flight. |
| Public Aircraft | An aircraft operated by a governmental entity (including federal, state, or local governments, and the U.S. Department of Defense and its military branches) for certain purposes |
| RTCA | RTCA, Inc. is a private, not-for-profit corporation that develops consensus-based recommendations regarding communications, navigation, surveillance, and air traffic management system issues. RTCA functions as a Federal Advisory Committee. Its recommendations are used by the FAA as the basis for policy, program, and regulatory decisions and by the private sector as the basis for development, investment and other business decisions (`www.rtca.org`) |
| See and Avoid | When weather conditions permit, pilots operating instrument flight rules or visual flight rules are required to observe and maneuver to avoid another aircraft. |

| Terminology | Definition |
|---|---|
| Self-Separation | Sense and Avoid system function where the UAS maneuvers within a sufficient time-frame to remain well clear of other airborne traffic. |
| Sense and Avoid | The capability of a UAS to remain well clear from and avoid collisions with other airborne traffic. Sense and Avoid provides the functions of self-separation and collision avoidance to establish an analogous capability to "see and avoid" required by manned aircraft. |
| Unmanned Aircraft | 1. A device used or intended to be used for flight in the air that has no onboard pilot. This devise excludes missiles, weapons, or exploding warheads, but includes all classes of airplanes, helicopters, airships, and powered-lift aircraft without an onboard pilot.<br>2. An aircraft that is operated without the possibility of direct human intervention from within or on the aircraft. |
| Unmanned Aircraft System | An unmanned aircraft and its associated elements related to safe operations, which may include control stations (ground, ship, or air-based), control links, support equipment, payloads, flight termination systems, and launch/recovery equipment.<br>An unmanned aircraft and associated elements (including communications links and the components that control the unmanned aircraft) that are required for the pilot-in-command to operate safely and efficiently in the national airspace system. |
| Visual Line of Sight | Unaided (corrective lenses and/or sunglasses exempted) visual contact between a pilot-incommand or a visual observer and a UAS sufficient to maintain safe operational control of the aircraft, know its location, and be able to scan the airspace in which it is operating to see and avoid other air traffic or objects aloft or on the ground. |

Table 4: Terminology

*Note. Acronyms* (tab. 1) and *Terminology* (tab. 4) are in compliance with *ICAO*, *FAA*, and, *EASA* definitions, refer to [13] for more detailed information.

# Chapter 1

# (W) Introduction

*To be done here:*

- Introduce UAV problem of integration into non segregated airspace.
- Open JARUS/NASA requirements and discuss legal framework a little
- Introduce Honeywell interest into topic.

## 1.1 (W) Related Work

*To be done here:*

- Ramasy work, Sabatiny work on LiDAR and obstacle avoidance, introduce movement automaton etc...
- Lattice search related work, the problem of lattice search above 4th dimension
- Reach set approximation related work

## 1.2 (W) Goals

*To be done here:*

- Propose abstract obstacle avoidance framework able to avoid obstacles in real time and guaranteeing safe path independent of controlled platform
- Define guarantee of safety margin concept.
- Define requirements for avoidance set.

## 1.3 (W) Assumptions

*To be done here:*

- Guarantee feasible safe trajectory in open world space at low altitude of the flight. Manage information resources about real obstacles, weather obstacles, ATM restrictions.
- create previously mentioned points as assumptions, xxx is accessible to feasible extent, etc ...

## 1.4 (W) Overview

*To be done here:*

- Chapter overview
- Notation of key concepts

## 1.5 (W) Contributions

*To be done here:*

- Notable contributions
- List of concepts and articles with references

# Chapter 2

# (W) Collision avoidance

The context of Collision Avoidance is introduced in table 2.1, the structure was taken from Gardi [12]and modified to reflect actual state of art.

| *Function* | *Equipment/Task* |
|---|---|
| Communication | Telecommunication datalinks,<br>Controller Pilot Data Link-Control (CPDLC),<br>Voice Communication |
| Navigation | Navigation sensors including GNSS, INS, etc.<br>providing 3D/4D navigation capabilities. |
| Surveillance | Cooperative Systems (TCAS, ACAS, etc.)<br>Non-cooperative Sensors (LiDAR,Cameras, etc.) |
| Situation<br>Awareness | Early Warning Systems,<br>CDTI Display |
| Autonomous<br>Decision<br>Making | Strategic, Tactical, Emergency Flight Planning,<br>Intelligent Collision Detection,<br>Conflict Resolution and Prevention,<br>Weather/Terrain/Constraints Avoidance |

Table 2.1: Collision Avoidance Systems Context Overview [12].

*Communication* overview elaboration on capability, reliability, security, architecture have been summarized by Johansen et. al. in [14].

*Navigation* overview is given by Nex [15] *Waypoint planning in 3D enviroment* is elaborated in [16]. *Waypoint Tracking and Test Environments* are thoughtfully discussed in [17, 18, 19, 20].

*Surveillance* cooperative surveillance is covered by TCAS and ACAS systems, interesting aspect of these systems are *Resolution Advisories* [21] for TCAS [22]. For ACAS. The visualization of surroundings has been introduced in [23]. *LiDAR* based *SAA* system have been introduced by Sabatini [24] further enhanced by Ramasay [25]. Other *Non-Cooperative* sensors and their feasibility have been outlined in Ramasay work [26].

*Situation awareness* is implemented mainly as *human-based* systems, *Early Warning System* has been proposed by Lee [27] and adaptive version by Miller [28]. Effects of *CDTI Display* visualization and human decision impact have been examined by Thomas [29]. *Self Separation* aspect have been examined by Williams [30].

*Autonomous Decision Making* –*TBD:*This will be added later, because its main part...

*Note.* The purpose following sections is to introduce current state in:

1. Introduce *Airspace Classification* in 2.1, from current ICAO/FAA/EASA viewpoint, emphasizing common viewpoint among regulations.

2. Provide background for *manned aircraft operation rules* in 2.2,restriction imposed by Visual(2.2.1)/Instrumental(2.2.2) Flight rules.

3. Define *"Well Clear"* (sec.2.3) state of aircraft in airspace. Introducing the roles of *Air Traffic Control* (2.3.1) and current CAS systems TCAS (2.3.2) and ACAS-X (2.3.3).

The purpose of following sections is to introduce ATM extension for UAS systems:

1. *UAS Traffic Management* functionality is analysed in 2.4, two major movements EU USPACE (2.4.1) and US NASA UTM (2.4.2).

2. *Event Based Avoidance* (2.5) defines basic event based control invoked by *UTM*, two major categories are analyzed in *MACP* (2.5.1) section and *Weather Impact* (2.5.2)

3. *UTM Functional decomposition* is given in section 2.6.

## 2.1 (W) Airspace classification

- Airspace classes,
- Controlled airspace (Risk analysis, Actors)
- Uncontrolled airspace (Risk analysis, Actors)
- Aircraft categorization
- MOPS for different airspace

## 2.2 (W) Aircraft operational rules

- Introduction to rules and classification and regulations.
- Analysis of impact.

### 2.2.1 (W) Visual Flight Rules (VFR)

- Rules of the Air
- Visual separation

### 2.2.2 (W) Instrumental Flight Rules (IFR)

- Instrumental separation
- Alert/Notice definition
- Prioritization IFR/VFR

## 2.3 (W) Remaining "Well Clear"

- Well clear definition/Near miss definition
- Well clear zones (Crash margin (near miss), warning margin, alert margin)

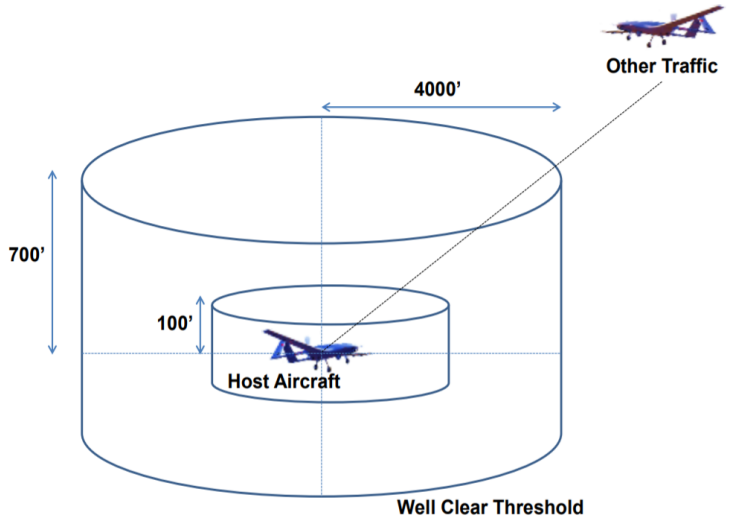



Figure 2.1: Well Clear Threshold [1, 2].

### 2.3.1 (W) Air Traffic Control(ATC)

- Keeping Aircraft↔ UTM ↔ Aircraft well clear
- ATC situations - Climb/Descent/Leveled flight (presentation), put emphasis on horizontal/vertical separation
- warning/notice/resolution notation and definitions
- standard ATM functionality
- Standard procedures convergence/divergence
- Standard procedures for maneuver restrictions

Real-time *Airspace Management* approach have been presented in [31]. Following *Dynamic Airspace Management* [3].

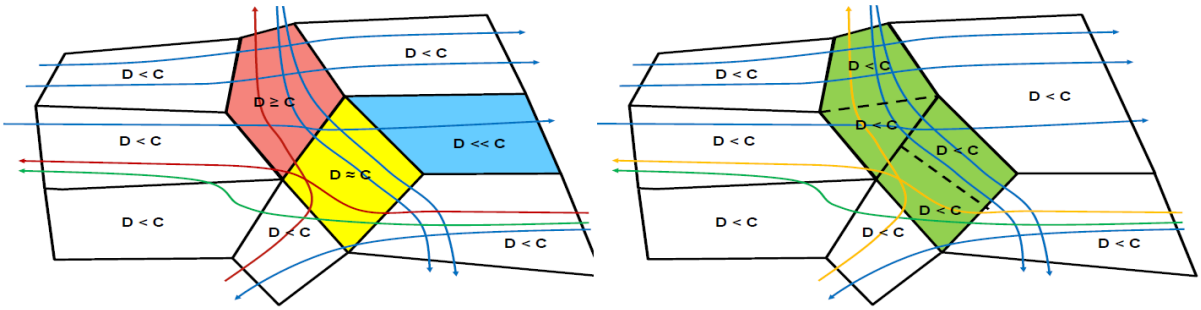



Figure 2.2: Example of DAM flight rerouting to homogenize traffic density [3].

### 2.3.2 (W) Traffic Collision Avoidance System (TCAS)

- Keeping Aircraft $\leftrightarrow$ Aircraft well clear,
- Traffic advisories and alerts messaging,
- Basic principle dissemination,

### 2.3.3 (W) Airborne Collision Avoidance System X (ACAS-X)

*ACAS-X*$_U$ concept have been implemented as *Deep Neural Network Solver* in [32]

- ACAS principples
- AXAS XA addition TCAS

## 2.4 (W) UAS Traffic Management (UTM)

- Introduce the role of future UTM system
- Discuss crucial requirements for UTM systems in sense of DAA topic
- Strongly refer to DAA UTM CONOPS (Appendix B) already sent to Joao
- focus on controlled airspace
- outline challenges for G class airspaces

### 2.4.1 (W) U-Space (EU)

- Introduce U-Space project in terms of EUROControl, EASA, ICAO, JARUS intention and involvement
- Outline supported DAA functionality for Phase 1 - Phase 4
- Refer to USPACE leaflet, already sent to Joao

### 2.4.2 (W) NASA UTM (US)

- Focus on DAA aspect and level flight in controlled airspace
- Refer to materials at: `https://utm.arc.nasa.gov/index.shtml`

## 2.5 (W) Event Based Avoidance

- Introduce the context for Event based avoidance:
  - Preventive DAA - before flight preparation
  - Event-based DAA - reaction window in minutes, usually cooperative via notifications and other methods

  - Reactive DAA - you know the drill boys ..., reaction window in 10th of seconds ...

- Introduce the process of notification and Event Resolution
- Outline hierarchy in U(A)TM controlled space
- Outline expected minimal set of data sources

### 2.5.1 (W) Mid-Air Collision Prevention

- Identify and dispute shortcomings of Aircraft classification in ICAO
- List of MAC events
- For each event describe trigger and resolution
- cite barrel [33]

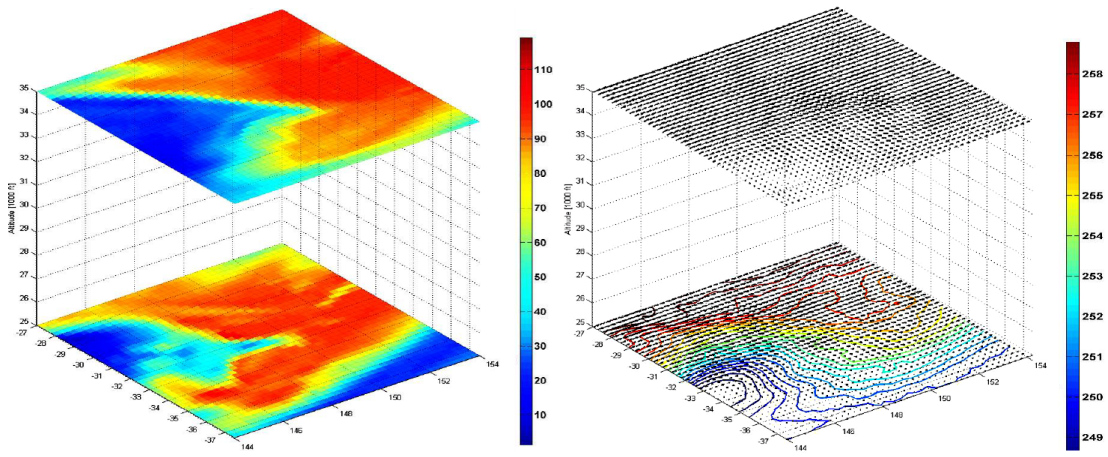
### 2.5.2 (W) Weather Impact

- do not use term "climate change"
- critical events are getting more localized and their magnitude is increased
- List of Weather event
- Discuss weather mitigation

*Weather Impact* on transportation in general has been introduced by Koetse in study [34].

*Weather-based* preemptive planning have been introduced into manned aviation in 2015 [35]. –*TBD:*This one shows global model impact

*Severe Weather Condition Detection Capabilities* for current level of standard aviation equipment have been reviewed in [36] –*TBD:*This one shows intermediate weather detection



(a) Relative Humidity.

(b) Localized Wind Vectors.

Figure 2.3: Localized Weather Model [4].

*Icing risk* in localized environment have been predicted based on numerical model [37]. It is shown that icing can be prevented by *planned* and *reactive* avoidance.

*Weather Models* can be extracted from *Climate and weather model archive at the National Oceanic and Atmospheric Administration* [5].

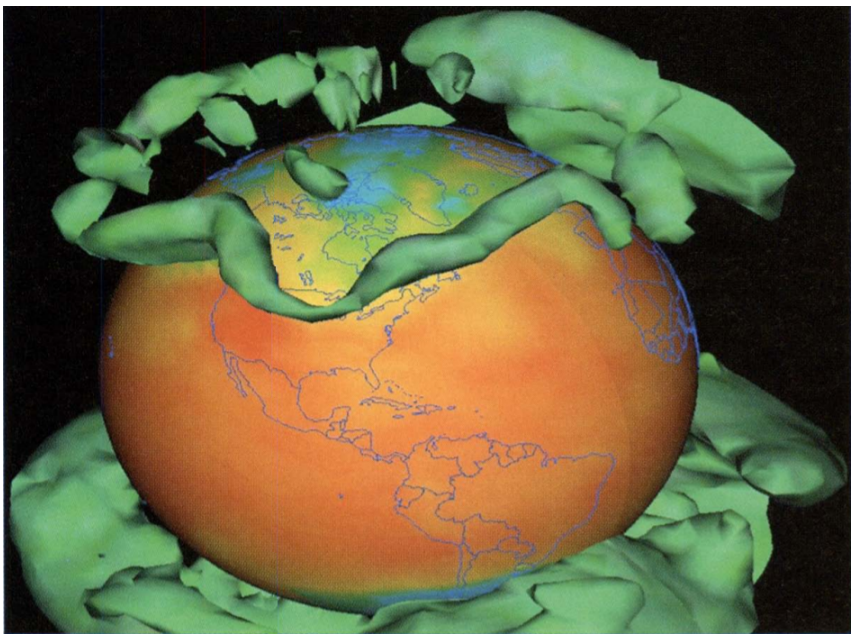


Figure 2.4: Example of upper troposphere winds [5].

## 2.6 (W) UTM Functional Decomposition

- general thinking, approach
- notification run
- resolution run

### 2.6.1 (W) Information Exchange Principles

- Information Exchange schematics in general, describe data flow and actors

**Collision Case**

- what collision case contains and how is calculated
- define conditions and role prioritization
- define blank spaces in VFR procedures

**Weather Case**

- This section needs to be discussed with subject matter expert.

### 2.6.2 (W) Resolution Advisories

- just describe the resolution format from ATM
- discuss manned/unmanned interaction and reaction delays, reaction uncertainity, etc ...

**Movement Restriction**

**Convergence/Divergence**

# Chapter 3

# (R) Background Theory

**Motivation:** Cooperative and Non-Cooperative *Sense and Avoid* (SAA) systems are key enablers for the *Unmanned Aerial Systems* (UAS) to routinely access non-segregated airspace [38]. Both cooperative and non-cooperative SAA systems are being developed to address this integration requirement.

The *SAA capability* is defined as the automatic detection of possible conflicts by the UAS platform under consideration and performing avoidance maneuvers to prevent the identified collisions.

An analysis of the available SAA candidate technologies and the associated sensors for both cooperative and non-cooperative SAA systems is presented in [39].

Non-cooperative *Collision Detection and Resolution* (CD&R) for UAV is considered as one of the major challenges that needs to be addressed [40] for the insertion of UAVs in non-segregated air space. As a result, a number of non-cooperative sensors for the SAA system have been adopted. Light Detection and Ranging (LIDAR)is used for detecting, warning and avoiding obstacles for low-level flying [24].

An approach to the definition of encounter models and their applications to SAA strategies is presented in [41] for both cooperative and non-cooperative scenarios.

Since 2014, there is a visible strong political support for developing rules on drones but regulations are not harmonized yet. The European Aviation Safety Agency (EASA) has been tasked to develop a regulatory framework for drone operations and proposals for the regulation of "low-risk" UAV operations. In achieving this, EASA is working closely with the Joint Authorities for Regulation of Unmanned Systes (JARUS) [42].

**Background Areas:** Following Areas are introduced in this chapter:

1. *UAS System Model* (sec. 3.1) - continuous and discrete mathematical models.
2. *Reach Sets* (sec. 3.2) - introduction to *Reach set* representation and calculation methods.
3. *Movement Automaton* (sec. 3.3) - intuitive definition and establishment of *hybrid automaton* control and modeling.
4. *LiDAR* (sec. 3.4) - short summary of *LiDAR* technology and terminology introduction.
5. *Complements of Algebra* (sec. 3.5) - necessary algebra complements for *Local* and *Global Coordinate Systems*.

## 3.1 (R) UAS System Model

This section strongly follows [43].

### 3.1.1 Continuous-time systems

Consider a class of systems given by functions:

$$\begin{aligned} &StateEvolution : input(time) \rightarrow state(state_0, time) \\ &input(time) : [0, FinalTime] \rightarrow \mathbb{R}^p \\ &input(time) \in \mathbb{R}^p, state(time) \in \mathbb{R}^n \end{aligned} \tag{3.1}$$

Where $input(time)$ and $state(state_0, time)$ are a sets of continuous-time signals. These are often called continuous-time systems because they operate on continuous-time signals.

Frequently, such systems can be defined by differential equations that relate the input signal to the output signal.

A prototypical description of a controlled (there is a control input signal) continuous-time system is:

$$\begin{aligned} &\partial/\partial \text{t state}(time) = \\ &\qquad f(time, state(time), input(time)), input(time) \in Inputs(time) \end{aligned} \tag{3.2}$$

Where $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^p \rightarrow \mathbb{R}^n$ satisfies the conditions for existence and uniqueness of the ordinary differential equation and $u$ is our control [44].

### 3.1.2 Discrete-time systems

Consider another class of systems given by functions

$$\begin{aligned} &StateEvolution :input(k) \rightarrow state(k), \\ &\qquad k \in \{0, t_s, 2.t_s, 3.t_s, \ldots i.t_s\}, i \in \mathbb{N}^+ \\ &input(k) \in \mathbb{R}^p, state(k) \in \mathbb{R}^n \end{aligned} \tag{3.3}$$

Where $input(k)$, $state(k)$ is a set of discrete-time signals. They can be represented by a function $f$ like $f : \{0, t_s, 2.t_s, 3.t_s, \ldots i.t_s\} \rightarrow \mathbb{R}^n, i \in \mathbb{N}^+$ where $t_s$ is sampling time and $i$ is discrete step [45].

### 3.1.3 Adversarial behaviour in continuous systems

Consider a subclass of continuous time systems where are two sets of control signals $uas(time)$ and $adversary(time)$ which are accommodated in following system:

$$\begin{aligned} &\partial/\partial \text{t state}(time) = f(t, state(time), uas(time), adversary(time)), \\ &\qquad uas(time) \in UASInputs(time) \subset \mathbb{R}^u, \\ &adversary(time) \in AdversaryInputs(time) \subset \mathbb{R}^v \end{aligned} \tag{3.4}$$

This system representation is often used in definition of problem of pursuit/evasion problem. Krasovskii developed a solution approach to this problem in [46]. A complex example of can be found in article [47].

## 3.2 Reach Sets

Informally, the *Reach Set* of a UAS system described by a differential equation is the *set of all states that can be reached from an initial state within a given time interval.*

### 3.2.1 Incremental Definition

**For following definitions** consider *nonlinear UAS system* described in (sec. 3.1).

**Definition 1** (Reach set starting at a given point)**.** *Suppose the initial position and time* $(state_0, time_0)$ *are given. The reach set* $ReachSet[\tau, time_0, state_0]$ *of* nonlinear system *at time* $\tau \geq time_0$*, starting at* $(state_0, time_0)$ *is given by:*

$$ReachSet[\tau, time_0, state_0] = \bigcup\{state(\tau) : input(s) \in Inputs(s), s \in (time_0, \tau]\} \quad (3.5)$$

**Reach set starting at given set** can be used to determine reach set in case of *hybrid system* input control switch and it is defined as follow:

**Definition 2.** *set starting at a given set] The reach set at time* $\tau > t_0$ *starting from set* $States_0$ *is defined as:*

$$ReachSet[\tau, time_0, States_0] = \bigcup\{ReachSet[\tau, time_0, state_0] : state_0 \in States_0\} \quad (3.6)$$

**Reach set for adversarial behavior** can be used to calculate possible escape routes from pursuer and it is defined as follow:

**Definition 3** (Reach set under adversarial behavior)**.** *Consider now the case of adversarial behavior([46, 47]). where* $input(t)$ *is our control and* $adversary(t)$ *is adversary control which is independent of* $input(t)$*, let* $differentialControl(t) = input(t) - \sup_{state \in state(t)} adversary(t)$*, which represents worst possible input change in given state and time, then* reach set for system *is represented as:*

$$ReachSet\begin{bmatrix}\tau, \\ time_0, \\ state_0\end{bmatrix} = \bigcup\left\{state(\tau) : \begin{matrix} differentialControl(s) \in \\ DifferentialControlSet(s)\end{matrix}, s \in (time_0, \tau]\right\} \quad (3.7)$$

**Reach set under constraints** are usable to define state constrained systems in terms of dynamics and technical capabilities.

**Definition 4** (Reach set under state constraints)**.** *Suppose the initial position and time* $(state_0, time_0)$ *and state constraints are given* $state(t) \in \mathbb{A} \subset \mathbb{R}^n, \dot{x}(t) \in \mathbb{B} \subset \mathbb{R}^n$*. The reach set* $ReachSet[\tau, time_0, \vec{state}_0]$ *of* nonlinear UAS system *at time* $\tau \geq time_0$*, starting at position and time* $(state_0, time_0)$ *is given by:*

$$ReachSet\begin{bmatrix}\tau, \\ time_0, \\ state_0\end{bmatrix} = \bigcup\left\{state(\tau) : \begin{matrix} \forall s \in (time_0, \tau], state(s) \in \mathbb{A}, \\ \dot{state}(s) \in \mathbb{B}, \\ \exists input(s) \in Inputs(s)\end{matrix}\right\} \quad (3.8)$$

### 3.2.2 Computation of Reach Sets

Several techniques for reachability analysis of systems have been proposed. They can be (roughly) classified into two kinds:

1. Purely symbolic methods based on:
   a. the existence of analytic solutions of the differential equations and
   b. the representation of the state space in a decidable theory of the real numbers.
2. Methods that combine
   a. numeric integration of the differential equations
   b. symbolic representations of approximations of state space typically using (unions of) polyhedra or ellipsoids.

These techniques provide the algorithmic foundations for the tools that are available for computer-aided verification of hybrid systems ([48], [49], [50]).

The set-valued Lebesgue integral provides a conceptual tool for the direct computation of the reach set. In what follows we describe techniques from dynamic optimization which are used to compute reach sets for dynamic systems.

The relation between dynamic optimization and reachability was first observed in [51]. A typical problem of optimal control can be formulated as follows:

$$\max \left( \begin{array}{c} \int_{initialTime}^{finalTime} cost(time, state(time), contro(time)) \mathrm{d}time + \ldots \\ \cdots + FinalCost(state(finalTime)) \end{array} \right) \tag{3.9}$$

For nonlinear system:

$$\dot{state}(t) = f(t, state(t), control(t)), control(t) \in ControlSet(t) \subset \mathbb{R}^p \tag{3.10}$$

Where *cost* is given as cost function of time,state and input and $FinalCost$ represents cost functional. There are two main techniques to solve this problem:

1. The maximum principle
2. Dynamic programming. The maximum principle gives necessary conditions of optimality. Dynamic programming may be used to derive sufficientconditions of optimality.

A good reference on the maximum principle is [52]. A less known reference with detailed geometric interpretations is [53]. A good reference on dynamic programming is [54].

## 3.3 (R) Movement Automaton

*Movement Automaton* is basic interface approach for discretization of *trajectory evolution* or *control input* for any *continuous or discrete system model*.

*Main function* of *Movement Automaton is* for system given by equation $\dot{state} = f(time, state, input)$ with initial state $state_0$ to generate *reference trajectory* $\hat{state}(t)$ or *control signal* $input(t)$.

Using *Movement Automaton* as *Control Proxy* will provide us with *discrete command chain* interface. This will reduce the *non deterministic* element from *Evasive trajectory* generation, by reducing infinite maneuver set to finite *movement set*.

*Non determinism* of *Avoidance Maneuver* have been discussed as an issue in following works:

1. Newton gradient method for evasive car maneuvers [55].
2. Non-holistic methods for trajectory generation [56].
3. Stochastic approach to elliptic trajectories generation [57].

*Examples* of *Movement Automaton Implementation* as *Control Element* can be mentioned as follows:

1. Control of traffic flow [58].
2. Complex air traffic collision situation resolution system [8, 59].
3. SAA/DAA capable avoidance system [7].

### 3.3.1 Hybrid Automaton

First the notion of *hybrid* automaton [48, 60, 61] needs to be introduced:

**Definition 5.** *Hybrid automaton (3.11) is given as structure:*

$$HybridAutomaton(States, SystemState, VectorField, DiscreteTransition, ResetMap) \tag{3.11}$$

States *(Q) is given as set of discrete states, for every time* $t \in$ *Domain hybrid automaton stays in exactly one of* states.

SystemState *(x), is given in domain* $x \in \mathbb{R}^n, n \in \mathbb{N}^+$, *representing the trajectory evolution.*

VectorField *(f) (3.12) is bounded to single State* $\in$ *States and represents local SystemState evolution, when given automaton State is Active.*

$$VectorField : State \times SystemState \to SystemState \tag{3.12}$$

DiscreteTransition *($\varphi$) (eq. 3.13) indicates changes of states in automaton, the changes are triggered by satisfying specific condition given by State and SystemState.*

$$DiscreteTransition : State \times SystemState \to State \tag{3.13}$$

ResetMap *($\rho$) (eq. 3.14) defines changes of State to some default value, this change is triggered by specific automaton State and SystemState.*

$$ResetMap : State \times SystemState \to SystemState \tag{3.14}$$

### 3.3.2 Building Blocks

**Definition 6.** *Movement Primitive:*
States *from* Hybrid automaton *can be taken as* Movements *in* Movement Automaton. MovementPrimitive *(eq. 3.15) is describing the* Movement *behaviour as transfer function* VectorField *enriched with parameters.*

$$\begin{aligned} &MovementPrimitive(vectorField, minimalDuration, parameters) \\ &VectorField : SystemState \times parameters \rightarrow SystemState \end{aligned} \tag{3.15}$$

**Example:** Let say that *UAS* system is given as $\dot{position} = velocity$, then let us have two *MovementPrimitives*:

1. *Stay* - $minimalTime = 1s$, $parameters = \{\}$, $VectorField : \dot{position} = 0$.
2. *Move* - $minimalTime = 1s$, $parameters = \{velocity\}$, $VectorField : \dot{position} = velocity$.

**Trajectory from Movement Primitives:** The *UAS* should *Move* for $5s$ with velocity $10m/s$, then *Stay* for $10s$, then move for $7s$ with velocity $4m/s$, with initial position $position_0 = 0$ and initial time $t_0 = 1$ The standard approach is to derive transfer function $position = \Theta(\dots)$

$$position(t) = \Theta(\dots) \begin{cases} t \in [0, 5] & : 10 \times t + position(0) \\ t \in (5, 15] & : 0 \times (t - 5) + position(5) \\ t \in (15, 22] & : 4 \times (t - 15) + position(15) \end{cases} \tag{3.16}$$

The *example* given by (eq. 3.16) is fairly primitive, but imagine UAS system given by nonlinear dynamics $\dot{x} = f(x, u, t)$ [62]. Then defining transfer function for given command chain can be impossible.

**Definition 7.** *Movement Transition:*
System state *can be different than intended movement application, the notion of* Transition *is therefore introduced as stabilizing element in movement chaining (eq. 3.17).*

$$Transition : MovementPrimitive \times SystemState \rightarrow MovementPrimitive \tag{3.17}$$

**Trajectory with Transitions:** Introducing two transitions $Transition(Move, Stay)$ and $Transition(Stay, Move)$ reflecting periods when vehicle stop moving or speed-up to desired velocity. The transfer function (eq. 3.16) can be rewritten as combination of *MovementPrimitives* (eq. 3.15) and *Transitions* (eq. 3.17):

$$\begin{aligned} &Transition(Stay, Move), Move(5s, 10m/s), \\ &\qquad Transition(Move, Stay), Stay(10s), \\ &\qquad\qquad Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{3.18}$$

.

*Note.* There are two types of *MovementPrimitives*:

1. *Stationary* - when system state is considered neutral and they are considered as entry point for automaton.
2. *Dynamic* - when the system state is considered evolving and they needs to be terminated with *stationary* transition.

**Movement Mapping Example:** Transition/MovementPrimitive pairs (eq. 3.17) can be mapped into movements (eq. 3.19).

$$\begin{aligned} Move(5s, 10m/s) &: Transition(Stay, Move), Move(5s, 10m/s), \\ Stay(10s) &: Transition(Move, Stay), Stay(10s), \\ Move(7s, 4m/s) &: Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{3.19}$$

**Definition 8.** *Movement:*
*Movement can consist from multiple* Transitions *(eq. 3.17) and one* MovementPrimitive *(eq. 3.15), the duration of* MovementPrimitive *can be shortened by* Transitions *duration.* Movement *is defined as follows:*

$$Movement \begin{pmatrix} initialState, \\ initialTime[0..1], \\ duration, \\ parameters[0..1] \end{pmatrix} = Chain \left( \begin{array}{c} InitialTransition(\dots)[0..*], \\ MovementPrimitive \begin{pmatrix} transitionState, \\ remainingDuration, \\ parameters \end{pmatrix} \\ LeaveTransition(\dots)[0..*], \end{array} \right) \tag{3.20}$$

Chain function *connects multiple* initial Transitions *which are appliead at* initialState *at* initialTime. *Then own* MovementPrimitive *(eq. 3.15) is invoked with* transitionnsState. Transitions state *is state changed by* Initial Transitions. *After* Movement Primitive *there can be* Leave Transitions Movement

**Minimal Movement Time:** Given by (eq. 3.21) for *movement* is given as sum of *MovementPrimitive* (eq. 3.15) minimal time, and *Transition* (eq. 3.17) in/out combined minimal time.

$$\begin{aligned} minimalTime(Movement) = &\ minimalTime(MovementPrimitive) + \\ &\max_{in/out} \{time(Transition)\} \end{aligned} \tag{3.21}$$

**Movement Chaining:** *Movements* can be *chained* and applied to initial *system state* to generate *system trajectory*. Example of trajectory is given by (eq. 3.16). Movements are reversibly obtained by participation such *trajectory* into *Movement primitives* and *Transitions*. Then sample *Trajectory* for $n \in \mathbb{N}^+$ movements looks like (eq. 3.22).

$$\begin{aligned}
&Trajectory(t_0) = State(t_0)\\
&Trajectory(t_0, t_1] = Movement_1(Trajectory(t_0), t_0, duration_1, parameters_1)\\
&Trajectory(t_1, t_2] = Movement_2(Trajectory(t_1), t_1, duration_2, parameters_2)\\
&Trajectory(t_2, t_3] = Movement_3(Trajectory(t_2), t_2, duration_3, parameters_3)\\
&\vdots\\
&Trajectory(t_{n-1}, t_n] = Movement_n(Trajectory(t_{n-1}), t_{n-1}, duration_n, parameters_n)
\end{aligned} \tag{3.22}$$

Given *Trajectory* at time $t_0$ is given as initial *State* of *System*. For time interval $(t_0, t_1)$, which length is equal to $duration_1$, the *State* is given by $Movement_1$ with $parameters_1$ and base time $t_0$. This behaviour continues for movements $2, \ldots, n$.

**Definition 9.** *Movement Buffer:*
Movements *can be chained into* Buffer *with assumption of* continuous movement execution. Continuous movement executions *each movement in chain (eq. 3.22) is executed in time interval* $\tau_i = (t_{i-1}, t_i]$ *where* $i$ *is movement order and* $\forall\ Movement_i$ *starting time is* $t_0$ *or* $t_{i-1}$ *from previous movement. With given assumption* Buffer *is given as (eq. 3.23) with parameters* $t_{i-1}, t_i$ *omitted, due* $t_0$ *and* $duration_i$ *dependency.*

$$Buffer = \{Movement_i(duration_i, parameters_i)\}\, i \in \mathbb{N}^+ \tag{3.23}$$

**Definition 10.** *Movement Automaton Trajectory:*
*Let say system* State$\in \mathbb{R}^n$ *which* Trajectory *is defined by movement chaining (eq. 3.22), applied on some* initial time $t_0 \in \mathbb{R}^+$ *and final time* $t_f = t_0 + \sum_{i=1}^{I} duration_i$*, with movements contained in* Buffer *(eq. 3.23) is given as* Trajectory *(eq. 3.24).*

$$Trajectory(t_0, State(t_0), Buffer)\ or\ Trajectory(State_0, Buffer)\ if\ t_0 = 0 \tag{3.24}$$

*Note.* The space dimension of *Trajectories* is $\mathbb{R}^{n+1}$ if the space dimension of state *Space* is $R^n$, because *Trajectory space* contains evolution of *Space* in time interval $T[t_0, t_f]$.

The transformation from *transfer function* (eq. 3.16) to *trajectory* (eq. 3.24) is natural, only set of *Movement primitives* (eq. 3.15) and set of *Transitions* (eq. 3.17) is required.

**State Projection:** *Trajectory* (eq. 3.24)is naturally evolution of space over time, then there exists *StateProjection* function (eq. 3.25) which returns *State* for specific *Time*.

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{3.25}$$

### 3.3.3 Definition

**Definition 11.** *Movement Automaton is given as follow:*

$$InitialState :\in \mathbb{R}^h, h \in \mathbb{N}^+ \tag{3.26}$$

$$System : \dot{State} = f(Time, State, Input) \text{ or } vectorField \tag{3.27}$$

$$Primitives = \left\{ MovementPrimitive_i \begin{pmatrix} vectorField, \\ minimalDuration, \\ parameters \end{pmatrix} \right\} i \in \mathbb{N}^+ \tag{3.28}$$

$$Transitions = \left\{ Transition_j \begin{pmatrix} MovementPrimitive_l, \\ MovementPrimitive_k \end{pmatrix}_{k \neq l} \right\} j \in N^+ \tag{3.29}$$

$$Movements = \left\{ Movement_m \begin{bmatrix} Transition_o[0..*], \\ MovementPrimitive_p \\ Transition_r[0..*], \end{bmatrix}_{o \neq r} \right\} m \in N^+ \tag{3.30}$$

$$Buffer = \{Movement_s(duration_s, parameters_s)\}\, s \in \mathbb{N}^+ \tag{3.31}$$

$$Executed = \{Movement_s(duration_s, parameters_t)\}\, t \in \mathbb{N}^+ \tag{3.32}$$

$$Builder : Movement \times MovementPrimitive \rightarrow Movement \tag{3.33}$$

$$Trajectory : InitialState \times Movement^u \rightarrow State \times Time, u \in N^+ \tag{3.34}$$

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{3.35}$$

System *(eq. 3.27) is given in form of* differential equations $\dot{x} = f(t, x, u)$ *or* other transformable equivalent*, with* initial state *(eq. 3.26).*

Movements *(eq. 3.22) are defined as sequence of necessary* initial transitions *(eq. 3.29),* movement primitive *(eq. 3.28), and,* leave transitions *(3.29).*

Buffer *contains a set of* movement primitives *(eq. 3.28) to be executed in order to achieve desired goal.* Builder *(eq. 3.33) assures that first* movement primitive *(eq. 3.15)from* Buffer *(eq. 3.31) is transformed into* next movement *(eq. 3.30) based on* current movement *(eq.3.30).*

*System* trajectory *(eq. 3.34) is defined in (eq. 3.24).* State projection *(eqs. 3.25,3.35) is giving* State *variable for time* $t \in [t_0, t_{max}]$ *where* $t_max$ *is given by:*

$$t_{max} = t_0 + \sum_{i=1,u} Buffer.Movement(i).movementDuration \tag{3.36}$$

*Note.* From Continuous Reach set to Movement Automaton Control Reach Set:

*The reach set* $R$ (3.37) for system $\partial/\partial$t state $= model(state, input)$ with initial state $state_0 = state(t_i)$ in time interval $[t_i, t_{i+1}[$ is with existing control strategy $input(t) \in ControlStrategy(t)$. The reach set $R(state_0, t_0, t_1)$ where $t_1 > t_0$.

$$R(state_0, t_0, t_1) = \bigcup \{state(s) : input(s) \in ControlStrategy(s), s \in (t_0, t_1]\} \tag{3.37}$$

*The reach set* $\mathcal{R}$ (3.38) of the system under the control of the *movement automation* consist from the set of trajectories $Trajectory(initialState, buffer)$, which are executed in constrained time period $[t_i, t_{i+1}[$.

$$ReachSet(state_0, t_i, t_{i+1}) = \{Trajectory(state_0, buffer) : \text{duration}(buffer) \leq (t_{i+1} - t_i)\} \tag{3.38}$$

*Note.* Weak Invariance:

When the UAS is under the control of the movement automaton for the obstacle avoidance problem, by design of the avoidance algorithm, the trajectories of the UAV will not intersect any threat. This means that the controlled system $\partial/\partial$t state $= model(state, input)$ is *weakly invariant* with respect to the complement of the threats, and with respect to the free space. A pair $(state, SafeSpace)$, where $\partial/\partial$t state $= model(state, input)$ and $SafeSpace$ is a closed set, is weakly invariant if there exist controls such that a trajectory starting inside $State_0 \in SafeSpace$ remains inside $State(t) \in SafeSpace$ [63].

## 3.4 (R) LiDAR

**LiDAR(Light Detection And Ranging)** is active form of remote sensing: information is obtained from a signal which is sent from a transmitter and reflected by a target, and detected by a receiver back at the source. Following types of information can be obtained:

1. *Range to target* - topographic LiDAR or laser altimeter.
2. *Chemical properties of target* - differential absorption LiDAR.
3. *Velocity of target* - Doppler LiDAR.

Chemical properties of target are out of scope. Velocity of target seems as interesting property to investigate, but this type of LiDAR is usually used for meteorological measurements of wind currents [64]. Extended research in LiDAR as obstacle detection sensor has been executed by research group around Sabatini [24] and Ramasy [25].

LiDAR output is represented as point cloud it is described by following definition.

**Definition 12** (Scanned point and Point-cloud)**.** *Consider viewpoint as origin of* $\mathbb{R}^3$ *space, Let point* $\in PolarCoordinates$ *be defined as:*

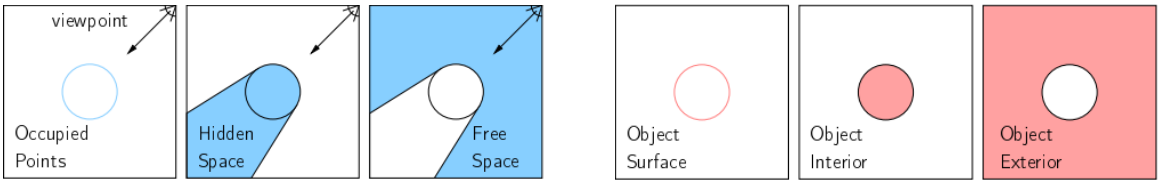
$$point = [distance, horizontal^\circ, vertical^\circ, time]^T \tag{3.39}$$

*Where* $horizontal^\circ$ *is horizontal angle from origin,* $vertical^\circ$ *is vertical angle to origin, and, time is time of retrieval.*

*Point-cloud is set of points scanned in small enough time-frame, based on processing raw point data it can have following representations:*

1. *Local point-cloud - position of sensor is used as origin of space and points can be represented in orthogonal or planar representation.*
2. *Global point-cloud -global position of sensor is used as reference to calculate global position of points.*

**Point-cloud** is usually addressed as *raw point-cloud* in case if its represented in Local planar coordinates. Other forms of point cloud require further processing and they are not feasible for real-time obstacle detection and avoidance [65].




(a) Space type definitions (b) Object properties definitions

Figure 3.1: Six space classifications [6].

Because of real-time obstacle avoidance it is necessary to introduce following terminology:

1. *Occupied points* - points which have been detected by LiDAR (also addressed as visible points).
2. *Hidden space* - space which is hidden behind occupied points, from viewpoint it is uncertain what is in that space.
3. *Free space* - space which is visible from viewpoint and it is not occupied by known objects.
4. *Object surface* - detected and undetected object surface
5. *Object interior* - occupied space by object.
6. *Object exterior* - free space around known objects.

Existing method for space segregation [6] leads to following definition:

**Definition 13** (Accessible space)**.** *Consider known space as space explored by sensor (it can have different viewpoint along previous 3D trajectory). Intersection between object exterior (Exterior) and free space Free gives us Accessible space (Accessible).*

$$Accessible = Exterior(object) \cap Free(object) \tag{3.40}$$

Accessible space $S_A$ (def. 13) is our bordering limitation for reachable space of system $ReachSet[\tau, time_0, state_0]$ (def. 1.).

## 3.5 (R) Complements of Algebra

**Cartesian Space:** 3D Cartesian space defined by an X, Y, and Z axes (describing position based on horizontal placement, vertical placement, and depth respectively). The coordinates for any point within this space are shown as a vector $[x, y, z]$. *Coordinate system* used this work is the right-handed system (thumbs points at positive direction of x-axis, index finger is pointing to positive direction of y-axis, the positive of z axis given by remaining fingers).

**Base Works:** *Euler* outlined *universal rotation theorem* which was presented in [66]. Rigid body dynamics and rotation matrices defined by Schaub [67].

**UAS Coordinate System:** *Local Coordinate frame* is defined by UAS mass center as space center, $Z-$ in direction of gravitational force, $X+$ in direction of UAS heading. This local coordinate system is called Euler Normalized Unit-frame (ENU).

**Rotation Matrices:** Following *Rotation Matrices* are used to transform between two displaced coordinate systems. Roll angle rotation is defined around X-axis by matrix (eq. 3.41) on YZ-plane. Pitch angle rotation matrix is defined around Y-axis by matrix (eq. 3.42) on XZ-plane. Yaw angle rotation matrix is defined around Z axis by matrix (eq. 3.43) on XY-plane.

$$R_{YZ} = R_{roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & \sin(roll) & \cos(roll) \end{bmatrix} \tag{3.41}$$

$$R_{XZ} = R_{pitch} = \begin{bmatrix} \cos(pitch) & 0 & \sin(pitch) \\ 0 & 1 & 0 \\ -\sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \tag{3.42}$$

$$R_{XY} = R_{yaw} = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.43}$$

The full rotation matrix in X,Y,Z is given by (eq. 3.44).

$$R_{XYZ} = R_{roll,pitch,yaw} = R_{XY} * R_{XZ} * R_{YZ} = R_{yaw} * R_{pitch} * R_{roll} \tag{3.44}$$

*Note.* The rotation matrix $R_{XYZ}$ (eq. 3.44) and its inverse $R^{-1}_{XYZ}$ which gives identity $R_{XYZ} \times R^{-1}_{XYZ} = I$ are used all over this work in transformation.

**Gimbal Lock Prevention:** To keep solution numerically stable and rotations numerically stable gimbal lock prevention is necessary [68]. Gimbal lock occurs when one of matrices (eq. 3.41, 3.42, 3.43) is singular or final matrix for X,Y,Z rotation is singular (eq. 3.44). Gimbal lock leads to loose of one or more degree of freedom, depending on rank and space dimension of singular matrix. To prevent gimbal lock it is necessary to introduce mechanism to check if rotation matrix is regular. For this purpose normative reset function is introduced:

$$[roll, pitch, yaw]^T = f(t, roll^-, pitch^-, yaw^-), \quad \text{norm}(R_{roll,pitch,yaw}) = 3 \tag{3.45}$$

Function resets yaw or roll angle to initial position to keep degree of rotation matrix. Simpler but not fault tolerant solution is to keep angles $roll, pitch, yaw \in (-\pi, \pi]$ range.

**Polar coordinates:** A *polar coordinate system* represents point in form of vector:

$$point_{polar} = [distance, horizontalDislocationAngle, verticalDislocationAngle]^T$$

which is ideal for representation of LiDAR scanned point, because usually total point distance and pair of dislocation angles are returned. Using most common LiDAR with horizontal rotation $horizontal^\circ$ and vertical mirror inclination $vertical^\circ$, one can define polar coordinate $point_{polar} = [distance_{x,y,z}, horizontal^\circ, vertical^\circ]$ which is dual to Cartesian coordinate $point_{cartesian} = [x, y, z]$. If rotation angle ranges are $horizontal^\circ, vertical^\circ \in (-\pi, \pi]$ transformation function is bijection.

**Polar → Cartesian:** Transformation from polar to Cartesian representation is defined by following series of functions (eq. 3.46, 3.47,3.48, 3.49).

$$distance_{xy} = \cos(horizontal^{\circ}) \times distance_{xyz} \tag{3.46}$$

$$z = \sin(hirizontal^{\circ}) \times distance_{xyz} \tag{3.47}$$

$$y = \sin(horizontal^{\circ}) \times distance_{xy} \tag{3.48}$$

$$x = \cos(vertical^{\circ}) \times distance_{xy} \tag{3.49}$$

**Cartesian → Polar:** Transformation from Cartesian to polar representation is defined by following series of functions (eq 3.50, 3.51,3.52, 3.53).

$$distance_{xyz} = \sqrt{x^2 + y^2 + z^2} \tag{3.50}$$

$$distance_{xy} = \sqrt{x^2 + y^2} \tag{3.51}$$

$$horizontal^{\circ} = \arctan\left(\frac{y}{x}\right) \tag{3.52}$$

$$vertical^{\circ} = \arctan\left(\frac{z}{d_{xy}}\right) \tag{3.53}$$

**Definition 14.** *Global Coordinate System (GCS) $\mathcal{X}_{\mathcal{G}}$ takes as center $c_{\mathcal{G}0}$ well known point (for example center of geo-reference model in GNSS systems) every reference distance, plane or angle is calculated taking this center to mind.*

**Definition 15.** *Local Coordinate System (LCS) $\mathcal{X}_{\mathcal{L}}$ takes as center $c_{\mathcal{L}0}$ frame of vehicle and can be changing position and orientation in global coordinate frame $\mathcal{X}_{\mathcal{G}}$.*

**Definition 16.** *Global position of polar obstacle $o_i \in \mathcal{O}_{3D}$. Let $o_i = [d_o, \theta_o, \varphi_o]^T$ be polar position of obstacle $o_i$ in local coordinate frame of vehicle with global Cartesian position $[x_{,v}, y_v, z_v]^T$ and normalized orientation angles $[roll_v, pitch_v, yaw_v]$.*
*Then Cartesian position of obstacle oi, $[x_o, y_o, z_o]^T$ in local coordinate frame is given by transformation functions $x_o$ (eq. 3.49), $y_o$ (eq. 3.48), $z_o$ (eq. 3.47).*
*Global position of polar obstacle oi, $[x_g, y_g, z_g]^T$ is given by following equation:*

$$\begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} = \begin{bmatrix} R_{XYZ}(roll_v, pitch_v, yaw_v) \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} \end{bmatrix} \tag{3.54}$$

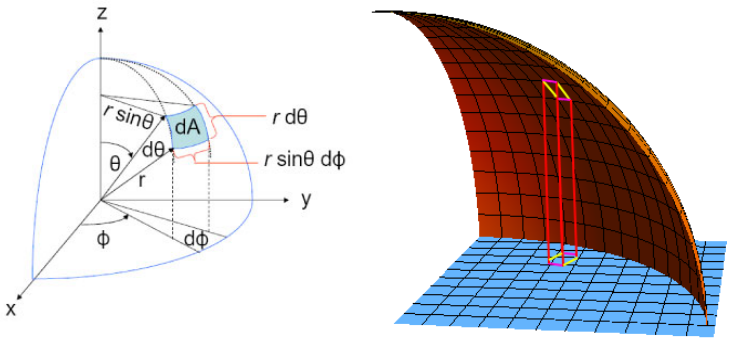
**Definition 17.** *Local position of global coordinate $[x_g, y_g, z_g]^T \in \mathbb{R}^3$. Let there be vehicle with global Cartesian position $[x_{,v}, y_v, z_v]^T$ and normalized orientation angles $[roll_v$, $pitch_v$, $yaw_v]$. in global coordinate frame $\mathcal{X}_{\mathcal{X}}$.*

*Then local Cartesian coordinate position* $[x_l, y_l, z_l]^T$ *of point* $[x_g, y_g, z_g]^T$ *is given by following equation:*

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \left[ R_{XYZ}(-roll_v, -pitch_v, -yaw_v) \left( \begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} - \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} \right) \right] \tag{3.55}$$

*Local polar position is given as* $[distance_l, horizontal_l^\circ, horizontal_l^\circ]$*, where* $distance_l$ *is given by (eq. 3.50),* $vertical_l^\circ$ *is given by (eq. 3.52).* $vertical_l^\circ$ *is given by (eq. 3.53), where* $[x_l, y_l, z_l]$ *are used as local coordinates.*

**Polar surface calculation:** The problem is to calculate intersected surface $dA$ of ball subsurface defined by radius $r$, horizontal span $\phi$ and vertical span $\theta$. From classical mechanics one can formulate problem as given by (fig. 3.2a). The intersection plot is in (fig. 3.2b).




(a) Notation (b) Plot

Figure 3.2: Polar surface calculation notation and plot

One can use first fundamental form to determine the surface area element. Recall that this is the metric tensor, whose components are obtained by taking the inner product of two tangent vectors in polar space $g_{i,j} = X_i \cdot X_j$, for tangent vectors $X_i$, $X_j$. Following identification for the components of metric tensor will be used:

$$g_{ij} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} \tag{3.56}$$

Where $E =< X_u, X_u >$, $F =< X_u, X_v >$, and $G =< X_v, X_v >$. Lagrange's identity can be used , which tells us that the squared area of a parallelogram in space is equal to the sum of the squares of its projections onto the Cartesian plane:

$$|X_u \times X_v|^2 = |X_u|^2 |X_v|^2 - (X_u \cdot X_v)^2 \tag{3.57}$$

Given example is displayed in (fig. 3.2b). The area element is given as:

$$\begin{aligned} \mathrm{d}A &= |X_u \times X_v| \quad \mathrm{d}u\mathrm{d}v \\ &= \sqrt{\left||X_u|^2 |X_v|^2 - (X_u \cdot X_v)^2\right|} \quad \mathrm{d}u\mathrm{d}v \\ &= \sqrt{EG - F^2} \quad \mathrm{d}u\mathrm{d}v \end{aligned} \tag{3.58}$$

We will find tangent vectors via the usual parametrization which give, $X(\phi, \theta) = [r\cos\phi\sin\theta,\, r\sin\phi\sin\theta,\, r\cos\theta]$, so that tangent vectors are simply defined as:

$$\begin{aligned} X_\phi &= [-r\sin\phi\cos\theta, r\cos\phi\sin\theta, 0] \\ X_\theta &= [-r\cos\phi\sin\theta, r\sin\phi\cos\theta, -r\sin\theta] \end{aligned} \tag{3.59}$$

Computing the elements of the first fundamental form gives us:

$$E = r^2\cos^2\theta, \quad F = 0, \quad G = r^2 \tag{3.60}$$

Thus final difference is given as:

$$\mathrm{d}A = \sqrt{r^4\cos^2} \quad \mathrm{d}\theta\mathrm{d}\phi = r^2\cos\theta \quad \mathrm{d}\theta\mathrm{d}\phi \tag{3.61}$$

*Note. Polar Surface* is used in *Detected Obstacle Rating Calculation* (sec. 6.5.1). The final formula used in *surface integral calculation* in *non-compact notation* is given as follow:

$$\mathrm{d}A = r^2\cos(horizontal^\circ) \quad \mathrm{d}horizontal^\circ\mathrm{d}vertical^\circ \tag{3.62}$$

# Chapter 4

# (R) Problem Statement

An *UAV* equipped with several types of sensors is tasked to fly several types of *Missions* in a 3-dimensional space. There is a *Terrain* map, an *Object map*, and a *Weather* forecast for target region that are known a priori. The map of the *Terrain* may not be up to date and *Uncharted obstacles* may affect flight safety. The *UAV* has to comply to a set of *Flight Rules* specifying *Flight Constraints*. The performance of the *UAV*, including sensor performance, is affected by the *Weather*.

Several difficulties must be faced. First, the design space of is large and complex. Second, *Trajectory calculation* Third, *Navigation*.

## 4.1 Basic Definitions

A few definitions are needed.

### 4.1.1 World

The world of interest consists of:

*Space* has a Global Reference Frame with three axes $X^+, Y^+, Z^+$.

$$Space \subseteq \mathbb{R}^3 : \text{main axes:} X^+, Y^+, Z^+, \text{ center}[x_0, y_0, z_0] \tag{4.1}$$

*Object* (4.2) is generic subset of *Space* which has *Boundary*.

$$Object \subset Space : \forall point \in Object, point \text{ is solid} \tag{4.2}$$

The dynamic model of the *UAV* is given by a nonlinear first order state-space model:

$$\dot{x} = f(t, x, u) \tag{4.3}$$

Where $x \in \mathbb{R}^{6+n}, n \in \mathbb{N}^{0+}$ is *system state* containing minimal information about vehicle position $[x, y, z]$ and orientation $[\theta, \varphi, \rho]$ (roll, pitch yaw), and $u(t)$ is *control signal* belonging to $R^k, k \in \mathbb{N}^+$, bounded by control set $u(t) \in U$.

The map of the terrain is given by *TerrainMap* mapping $(x, y)$ to the terrain elevation

$$TerrainMap : (x, y) \rightarrow z \tag{4.4}$$

*Weather* (4.5) can impact the performance of the *UAV*. *wind* can decrease flying capabilities and maneuverability, *visibility* can impair the performance of sensors such as LiDAR and cameras, *humidity* can be serious danger to electronic equipment and in combination low *temperature* it can cause icing on vehicle, and *air pressure* can impact ranging sensor and flight performance.

$$Weather : (position \in Space) \times time \rightarrow \begin{bmatrix} \text{wind } (w_x, w_y, w_z) \\ \qquad\quad [m/s, m/s, m/s] \\ \text{visibility } v\,[m] \\ \text{humidity } h\,[0-100\%] \\ \text{air pressure } a\,[hPa] \\ \text{temperature: } \tau\,[C] \end{bmatrix} \tag{4.5}$$

### 4.1.2 Mission

*Mission* (4.6) which UAV should fly is given as set of *ordered, feasible in terms of vehicle dynamic (4.3), waypoints* in subspace of *Space* (4.1).

$$Mission = \left\{ \begin{matrix} waypoint_1, waypoint_2, \ldots, waypoint_m : \\ \forall_{i=1\ldots m} waypoint_i \in Space \end{matrix} \right\}, \quad m \in \mathbb{N}^+, m \geq 2 \tag{4.6}$$

*Waypoint Passing* (4.7) function maps system (4.3) trajectory *projection in Space* and *Mission* waypoints (4.6) to a vector of *passing times*.

$$WaypointPassing : TrajectoryProjection \times Mission \rightarrow Time^m \tag{4.7}$$

*Note.* The *Mission* (4.6) is considered as successfully completed if and only if $\forall$ waypoints are reached and in given order (check output of 4.7).

### 4.1.3 Flight constraints

Flight constraints arise from aviation rules and from ATM.

There are $H$ sets of hard flight constraints expressed as subsets of *Space*:

$$HFlightConstraints = \{HFlightConstraint_1, \ldots, HFlightConstraint_H\} \tag{4.8}$$

The union of all $HFlightConstraint_i$ is the set $HFlightConstraint$. *Example:* turning radius of UAV, climb rate etc.

There are $S$ sets of soft flight constraints expressed as subsets of *Space*:

$$SFlightConstraints = \{SFlightConstraint_1, \ldots, SFlightConstraint_S\} \tag{4.9}$$

The union of all $SFlightConstraint_i$ is the set *SFlightConstraint*. *Example:* sharp maneuvers, restricted flight areas etc.

The *Weather* may impose either soft or hard flight constraints.

### 4.1.4 Partition of Space

In what follows it is convenient at each time $t$ to partition the *Space* into four disjoint subsets *Free(t)*, *Restricted(t)*, *Occupied(t)* and *Uncertain(t)*.

There is a *SpaceClassification* function (4.10)

$$SpaceClassification : y \in Space \mapsto s \in \{Free, Restricted, Occupied, Uncertain\} \tag{4.10}$$

*Note. SpaceClassification* (4.10) is *total* function mapping each element of *Space*. Other followup *classifications* are considered as *total* functions.

There is set of *Space* Hard constraints, where one *HardConstraint* (4.11) is given as:

$$HardConstraint = \{point \in Space : SpaceClassification(point) = Occupied\} \tag{4.11}$$

There is set of *Space* Soft constraints, where one *SoftConstraint* (4.12) is given as:

$$SoftConstraint = \{point \in Space : SpaceClassification(point) = Restricted\} \tag{4.12}$$

### 4.1.5 Information source

**Definition 18.** *Information source (4.13) represents a priori information for each* point *in* Space *at time prior mission execution mapping it into one of distinctive categories* Free, Occupied, Restriced*, and* Uncertain*.*

$$InformationSource = SpaceClassification(PriorTime) \tag{4.13}$$

**Definition 19.** *Landmark is partition of* Space *(4.1) which is notable in context of* society *or* law *given properties. Landmark can be: notable buildings, notable natural structures or crucial infrastructure. Landmark is part of terrain map, but it special status can induce additional properties.*

For obstacle avoidance problem following *Information sources* are available:

1. *Terrain map* - map of terrain with notable landmarks.
2. *Object map* - map of notable structures with *protection zones* considered as *occupied* or *restricted* space.
3. *Fly zones restriction map* - map of restricted flight areas considered as *restricted* constraints.

### 4.1.6 Single Observation by single sensor classification

*Vehicle* is equiped with single *Sensor*, which returns *Space* classification for given *observation position* and observation time.

Observation (4.14) at given UAV *position*, *time*, and for given *sensor* sorts points from *sensor reading* into following distinguish sets (4.14):

1. $Free_O(position, sensor, time)$ - observable space by sensor and considered as $Free$ by sensor reading,
2. $Occupied_O(position, sensor, time)$ - observable space by sensor and considered as $Occupied$ by sensor reading,
3. $Uncertain_O(position, sensor, time)$ - all other points.

$$Observation(position, sensor, time) \rightarrow \begin{cases} Free_O(position, sensor, time) \\ Occupied_O(position, sensor, time) \\ Uncertain_O(position, sensor, time) \end{cases} \tag{4.14}$$

### 4.1.7 Multiple observations by single sensor classification

Let Let add pairs of *observation position* and *observation time* in manner $\{(position_1, t_1), (position_2, t_2), \ldots (position_k, t_k)\}, k \in \mathbb{N}^+$ that point position is independent and time is ordered in fashion $t_1 < t_2 < \cdots < t_k$.

*Free space* from multiple sensor *reading* over multiple *positions* is inclusive space, because we are obtaining additional information regarding to space reachability by sensor reading independent on sensor space and orientation limitation. Therefore the union of single instances of observations are used to represent *Combined free space* $Free_O(sensor)$ (4.15).

$$Free_O(sensor) = \bigcup_{i=\{1,\ldots,k\}} Free_O(sensor, position_i, time_i) \tag{4.15}$$

*Occupied space* (4.16) from multiple sensor *reading* over multiple *positions* is inclusive space, because of increased information, therefore it has similar handling to *Free space*.

$$Occupied_O(sensor) = \bigcup_{i=\{1,\ldots,k\}} Occupied_O(sensor, position_i, time_i) \tag{4.16}$$

*Uncertain space* (4.17) from multiple sensor *reading* over multiple *positions* is exclusive space, because of decrease in uncertainty.

$$Uncertain_O(sensor) = \bigcap_{i=\{1,\ldots,k\}} Uncertain_O(sensor, position_i, time_i) \tag{4.17}$$

### 4.1.8 Sensor fusion

The observation at fixed time $t_{fix}$ can be made by multiple sensors $sensor_1$, $sensor_2$, ..., $sensor_k$, $k \in \mathbb{N}^k$, for $k$ sensors execute observations $Observation_1$ ($sensor_1$, $position_1$, $t_{fix}$), $Observation_2$ ($sensor_2$, $position_2$, $t_{fix}$), ..., $Observation_k$( $sensor_k$, $position_k$, $t_{fix}$).

*Sensor Fusion* (4.18) function with *data fusion parameters* is introduced which combines *Observations* for each *sensor* and *Weather*, then uniquely maps each point into four distinguish sets: $Free(t_{fix})$, $Occupied(t_{fix})$, $Restricted(t_{fix})$, *and* $Uncertain(t_{fix})$ (special case of (4.10)).

$$SensorFusion: \begin{bmatrix} Observation_1(sensor_1, position_1, t_{fix})\times \\ Observation_2(sensor_2, position_2, t_{fix})\times \\ \times\cdots\times \\ Observation_k(sensor_k, position_k, t_{fix})\times \\ Weather(\dots) \\ \textit{fixed time } t_{fix}\times \\ sensorFusionParameters \end{bmatrix} \rightarrow \begin{cases} Free(t_{fix}) \\ Occupied(t_{fix}) \\ Restricted(t_{fix}) \\ Uncertain(t_{fix}) \end{cases} \tag{4.18}$$

### 4.1.9 Data fusion

Multiple *Information Sources*: $InformationSource_1$, $InformationSource_2$, ..., $InformationSource_l$, $l \in \mathbb{N}$, where $l$ is count of information sources needs to be fused with *Sensor Fusion* function (4.18) outcome at *fixed time* $t_{fix}$.

*Data fusion* function (4.19) combines the classification from various *Information Sources* with classification from *Sensor Fusion* function (4.18) for vehicle position at *fixed time* $t_{fix}$, under given *Data Fusion Parameters*.

$$DataFusion: \begin{bmatrix} InformationSource_1\times \\ InformationSource_2\times \\ \times\cdots\times \\ InformationSource_l\times \\ SensorFusion(\dots)\times \\ Weather(\dots) \\ position\times \\ \textit{fixed time } t_{fix}\times \\ dataFusionParameters \end{bmatrix} \rightarrow \begin{cases} Free(t_{fix}) \\ Occupied(t_{fix}) \\ Restricted(t_{fix}) \\ Uncertain(t_{fix}) \end{cases} \tag{4.19}$$

Each *point* in *Space* is uniquely classified into one of sets$Free(t_{fix})$, $Occupied(t_{fix})$, $Restricted(t_{fix})$, $Uncertain(t_{fix})$ (special case (4.10)).

*Note.* Moreover *Data Fusion* function is covering the case of *Multiple information sources, combined with multiple sensor readings over multiple times*, including *Weather* as *sensor impact factor* and *information source*.

### 4.1.10 Known world

*Known world* (4.20) for some *fixed time* $t_{fix}$ is given as joint set of points which belongs to one of *Data Fusion* (4.19) output sets $Free(t_{fix})$ or $Occupied(t_{fix})$ or $Restricted(t_{fix})$. The *Known World* is compact set with existing boundary.

$$KnownWorld(t_{fix}) = Free(t_{fix}) \cup Occupied(t_{fix}) \cup Restricted(t_{fix}) \tag{4.20}$$

### 4.1.11 Safety margin

Let say that *mission* was executed in *time* interval $t \in [missionStart, missionEnd]$ in *Known world* (4.20). For every *position*, extracted from *UAV model state* $x(t)$, keeps distance from any point in $Occupied(t)$ with greater or equal to $safetyMargin$ $(s_m)$ (4.21).

$$\forall t \in [missionStart, missionEnd] : \\ distance(x(t), Occupied(t), t) \geq safetyMargin \tag{4.21}$$

## 4.2 Basic obstacle avoidance problem

Given:

1. Initial system state $x_0$, for UAV model (4.3).
2. Mission (4.6) to be executed.
3. Space (4.1), with existing objects (4.2).
4. Sensor system $\{sensor_1, sensor_2, \ldots, sensor_k\}$ with existing sensor fusion function (4.18).
5. Weather information (4.5) during the flight is available.
6. Information sources $informationSource_1, informationSource_2, \ldots, informationSource_l$ containing hard (4.11) and soft space constraints (4.12),with existing data fusion function (4.19).
7. *Hard* (4.8) and *Soft* 4.9 flight constraints given by ATM and rules of the air.

Generate *Control input signal* $u(t)$ to complete mission (4.6) with satisfaction of following conditions:

1. Waypoint passing function condition (4.7).
2. Set safety margin $s_m$ to *Occupied* space in $KnownWorld(time)$ (4.20) is not breached at any time (4.21).

## 4.3 Initial assumptions

*Initial assumptions* are following:

**Assumption 1.** *Filtered sensor readings are available.*
SensorObservation *(4.14) for given* position, time *returns classification of* Space *which is corresponding with real situation.*

**Assumption 2.** *There are no moving obstacles.*
*The initial* Space Classification Function *(4.10) is static for all observation times* $t \in (-\infty, \infty)$. *Moreover there are no* intruders *or* adversaries *present.*

**Assumption 3.** *The movement takes place in the unrestricted airspace .*

**Assumption 4.** *Mission consists of set of reachable waypoints.*
*For specific* UAV system *(4.3) and* Mission *(4.6), there exists control signal* $u(t)$ *which satisfies* Waypoint passing *(4.7) criterion and* SafetyMargin *(4.21) condition.*

**Assumption 5.** *The UAV is moving with constant velocity.*
*For given* UAV system *(4.3) there is a subset of state* $velocity(t) \subset x(t)$ *which contains velocity parameters. Then there exist transformation function* $LinearVelocity(\circ)$ *which maps* $velocity(t)$ *to* linear velocity $\in \mathbb{R}^1$. *For time* $t$ *in* missionStart *and* missionEnd *in* Mission *(4.6) constraint (4.22) with some* $constantVelocity \in \mathbb{R}^+$ *holds.*

$$\forall t \in \begin{bmatrix} missionStart, \\ missionEnd \end{bmatrix} : LinearVelocity(velocity(t)) = constantVelocity \tag{4.22}$$

*Note. Initial assumptions* 1., 2., 3., 4, and 5. will be relaxed in *Incremental problem definition.*

## 4.4 (R) Incremental problem definition

This section contains *incremental problem definition* as increments of (sec. 4.2). Each problem contains definition and references to addressed issues.

**Problem 1.** *Basic Avoidance (sec. 4.2) is to navigate trough* KnownWorld *under assumption that every* waypoint *in* Mission *is reachable.The* KnownWorld *is fed trough* SensorFusion *function which is joining* LiDAR *scanning into* Free(t), Occupied(t), *and,* Unknown(t) *sets in* discrete scan times $t$.

$$\begin{aligned} KnownWorld &:= SensorFusion(t) \forall point \in KnownWorld(t) \\ &= Free(t) \cup Occupied(t) \cup Unknown(t) \\ Mission &:= \forall waypoint \in Mission \text{ are reachable} \\ Sensors &:= \{LiDAR\} \\ SensorFusion &:= \{Clasificaiton\ function\} \\ HFlightConstraints &:= \{vehicle\ dynamic\} \end{aligned} \tag{4.23}$$

Challenges for problem 1. :

1. Navigation Loop Implementation *(sec. 6.7.3).*
2. Avoidance Loop Implementation *(sec. 6.7.2).*

**Problem 2.** *Intruder Problem in addition to* Known world evolution *(pr.1) the* ADS-B *sensor is introduced into* Sensors *array, this imposes* HardConstraint *of* Flight corridor *for detected intruders impacting the evolution of* Free*(t) and* Occupied*(t) sets significantly.*

$$\begin{aligned} KnownWorld &:= SensorFusion(t) \forall point \in KnownWorld(t) \\ &= Free(t) \cup Occupied(t) \cup Unknown(t) \\ Mission &:= \forall waypoint \in Mission \text{ are reachable} \\ Sensors &:= \{LiDAR, ADS-B\} \\ SensorFusion &:= \{Advanced\ joint\ sets\} \\ HFlightConstraints &:= \{vehicle\ dynamic\} \\ HardConstraints &:= \{intruder\ corridors\} \end{aligned} \tag{4.24}$$

Challenges for problem 2. :

1. Intruder Intersection Models *(minimal operation requirements achieved):*
   a. Linear Intersection Model *(sec. 6.6.2).*
   b. Body-volume intersection *(sec. 6.6.3).*
   c. Maneuverability uncertainty intersection *(sec. 6.6.4).*
2. Flight Corridors *(sec. 6.5.3).*

Relaxed Assumption: *2., the* UAS *encountering cooperative and non-cooperative intruders.*

**Problem 3.** *Static restrictions, in addition to* Intruder *problem (pr. 2) the* Information-Sources *are expanded by* static restriction *sources:*

1. ObstacleMap - *database containing notable landmarks, buildings, structures, with well defined* protection zones.

2. FlightRestrictions - *database containing ATM flight restrictions in non-segregated airspace for UAV relevant airspace categories.*

*This change impacts* DataFusion *by splitting* Free(t) *set into* Free(t) *and* Restricted(t) *disjoint sets. In addition* SoftConstraints *are introduced which contain restricted areas from relevant information sources.*

$$\begin{aligned}
KnownWorld &:= DataFusion(t) \forall point \in KnownWorld(t) \\
&= Free(t) \cup Occupied(t) \cup Unknown(t) \cup Restricted(t) \\
Mission &:= \forall waypoint \in Mission \text{ are reachable} \\
Sensors &:= \{LiDAR, ADS-B\} \\
SensorFusion &:= \{Advanced\ joint\ sets\} \\
InformationSources &:= \{TerrainMap, ObstacleMap, FlightRestriction\} \\
DataFusion &:= \{Advanced\ data\ fusion\} \\
HFlightConstraints &:= \{vehicle\ dynamic\} \\
HardConstraints &:= \{intruder\ corridors, terrain, obstacles\} \\
Softconstraints &:= \{protection\ zones\}
\end{aligned} \tag{4.25}$$

Challenges for problem 3. :

1. Obstacle Map *(sec. 6.5.2).*

2. Visibility Rating Concept *(fig. 6.11).*

3. Static Constraints *(sec. 6.5.3).*

Relaxed Assumption: *3., the* UAS *is moving in* restricted space *now.*

**Problem 4.** *Dynamic restrictions in addition to* Static restrictions *(pr. 3), the* Weather *as information source is introduced.* Soft constraints *are extended by medium level dangerous zones from weather map.* Hard constraints *are expanded by protection zones where the* weather *conditions are harsh. Overall* Weather *constraints are dynamic and changing position and shape over mission time. Modern weather systems can provide streamline overview of weather situation.*

$$
\begin{aligned}
KnownWorld &:= DataFusion(t) \forall point \in KnownWorld(t) \\
&= Free(t) \cup Occupied(t) \cup Unknown(t) \cup Restricted(t) \\
Mission &:= \forall waypoint \in Mission \text{ are reachable} \\
Sensors &:= \{LiDAR, ADS-B\} \\
SensorFusion &:= \{\text{Advanced joint sets}\} \\
InformationSources &:= \{TerrainMap, ObstacleDatabase, \\
&\quad FlightRestriction, Weather\} \\
DataFusion &:= \{Advanceddatafusion\} \\
HFlightConstraints &:= \{\text{vehicle dynamic}\} \\
HardConstraints &:= \{\text{intruder corridors,terrain,obstacles, protection zones}\} \\
Softconstraints &:= \{\text{protection zones}\}
\end{aligned}
\tag{4.26}
$$

Challenges for problem 4. :

1. Moving Constraints including Weather *(sec. 6.6.5).*
2. Weather Avoidance Case *(sec. 6.8.9).*

**Problem 5.** *Rules of the air, in addition to* Dynamic restrictions *(pr. 4),* Rules of the air *framework introduction, inducing new* SFlightConstraints *including air-spaces and rules of air impact on control mechanism.*

$$\begin{aligned}
KnownWorld :&= DataFusion(t) \forall point \in KnownWorld(t) \\
&= Free(t) \cup Occupied(t) \cup Unknown(t) \cup Restricted(t) \\
Mission :&= \forall waypoint \in Mission \text{ are reachable} \\
Sensors :&= \{LiDAR, ADS-B\} \\
SensorFusion :&= \{Advanced\ joint\ sets\} \\
InformationSources :&= \{TerrainMap, ObstacleDatabase, \\
&\quad FlightRestriction, Weather\} \\
DataFusion :&= \{Advanceddatafusion\} \\
HFlightConstraints :&= \{vehicle\ dynamic\} \\
SFlightConstratins :&= \{airspaces,\ rules\ of\ the\ air\} \\
HardConstraints :&= \{intruder\ corridors, terrain, obstacles,\ protection\ zones\} \\
Softconstraints :&= \{protection\ zones\}
\end{aligned} \tag{4.27}$$

Challenges for problem 5. :

1. *UTM Implementation (sec. 6.8).*
2. *Rule Engine for UAS (sec. 6.9.1).*
3. *Rule Implementation (sec. 6.9.2).*

Relaxed Assumption: *5., the* UAS *is required to move with different velocity during* Overtake maneuver.

*Note.* The assumptions 1.for *filtered sensor output* and 4 *Reachable waypoints* holds for all *problem increments.*

## 4.5 Avoidance requirements

**SAA systems** have following conflicting performance criteria:

1. *Energy efficiency* - minimize energy consumption and flight time.
2. *Trajectory tracking* - stick to the proclaimed trajectory in a mission plan.
3. *Safety* - avoid harm sources during the mission execution.

*Note. Energy efficiency* and *Trajectory Tracking* are an optional criteria, while *the safety* is mandatory.

### 4.5.1 Energy efficiency

**Energy efficiency** can be measured by *cost function* (eq. 4.28), consisting from the *cost of flown trajectory* (eq. 4.29) and the *expected reach cost* (eq. 4.30) portions. There are optimalizaiton techniques based on *Reach sets* [69]. The inputs for the *cost function* are:

1. *Time* - current mission time.
2. *Initial state* - UAS state at the beginning of a *mission*.
3. *Applied movements* - list of already executed movements.
4. *Future movements* - list of movements to be applied in future.
5. *Current state* - UAS state at *Time*, with current position and orientation included.
6. *Waypoint* - current goal waypoint.

$$Cost(t, \dots) = costTrajectoryFlown(t, \dots) + expectedReachCost(t, waypoint, \dots) \tag{4.28}$$

**Cost of flown trajectory** (eq. 4.29) from *initial state* to *current state* is calculated as a sum of an energy consumed for each movement with following components:

1. *Direct cost* - cost of consumed energy to execute movement.
2. *Horizontal cost* - portion of direct cost which was used for horizontal steering multiplied by *horizontal penalization*.
3. *Vertical cost* - portion of direct cost which was used for ascending/descending multiplied by *vertical penalization*.

$$costTrajectoryFlown\begin{pmatrix} time, \\ initialState, \\ appliedMovements \end{pmatrix} = \sum_{\substack{movement \in \\ appliedMovements}} \begin{pmatrix} duration.directCost+ \\ horizontal(cost, penalization)+ \\ vertical(cost, penalization) \end{pmatrix} \tag{4.29}$$

**Expected reach cost** (eq. 4.29) is calculated for a *planned trajectory* portion and a *direct waypoint distance* to the *latest future UAS position*. *Cost of planned trajectory* is calculated by same formula as a *cost of flown trajectory* (eq. 4.29), the initial state is replaced with a *current state* and *executed movements* are replaced with *planned movements*.

$$expectedReachCost\begin{pmatrix} time, \\ currentState, \\ futureMovements, \\ waypoint \end{pmatrix} = \begin{pmatrix} distance(futureState, waypoint)+ \\ costTrajectoryFlown\begin{pmatrix} currectState, \\ futureMovements \end{pmatrix} \end{pmatrix} \tag{4.30}$$

*Note.* The tuning parameters of cost function are: *Horizontal penalization* $\in [0, \infty]$ and *Vertical penalization* $\in [0, \infty]$. which are used to enhance the outcome of *cost function*.

Following setup of tuning parameters are used in our simulations:

1. $horizontalPenalization \leq verticalPenalization < \infty$ - in an *uncontrolled airspace*, all kind of maneuvers are allowed. Horizontal maneuvers are cheaper for a plane UAS.

2. $horizontalPenalizaiton < vetricalPenalization = \infty$ - in a *controlled airspace* any kind of horizontal maneuvering must be allowed by UTM.

The tuning parameters are set up $verticalPenalizaiton \leq horizontalPenalizaiton < \infty$ for *copter* UAS in an *uncontrolled airspace*

### 4.5.2 Trajectory tracking

**Trajectory Tracking** is crucial parameter for *controlled airspace* and is expected to be important in *upcoming UTM systems*. There is *mission plan* which is compared with *real-time airspace situation* obtained from UAS *Position notifications*. The optimalization based on *Reach Set* is given in [70].

**Motivation:** *Situation awareness* for modern DAA systems is depending on planned trajectory tracking. The main conflict is between *navigaiton precision* and *situation evaluation*. If the planned trajectory is defined for *continuous domain* it takes a lot of effort to calculate collision points.

The discrete domain of *Movement Automaton* (def. 11) can be used as *tool for situation awareness* The main idea is to use *Movement automaton as predictor for trajectory intersection* [8, 59].

**Movement Automaton trajectory tracking:** There is a *reference trajectory* which is used as comparison by *aviation authority* (ATM,UTM) given as:

$$\begin{aligned} ReferenceTrajectory = \{&(point_1, time_1), (point_2, time_2), \dots, \\ &, \dots (point_n, time_n)\} \quad n \in \mathbb{N}^+, point_k \in \mathbb{R}^3 \end{aligned} \tag{4.31}$$

*Reference Trajectory* (eq. 4.31) is given as set of *points* in *Global Coordinate System* for given *UAS*, *operational time-frame* and other authority depending properties.

The *movement* automaton is executing *Trajectory(initialState,buffer)* where buffer is set of *Movements*. The buffer is changing according to following pattern during *mission* time frame:

$$
\begin{aligned}
&\texttt{Mission Start:}\\
&\quad buffer = \{plannedMovements\}, planned = m\\
&\texttt{During Mission:}\\
&\quad buffer = \{executedMovements, plannedMovements\},\\
&\qquad executed = n, planned = o\\
&\texttt{Mission End:}\\
&\quad buffer = \{executedMovements\}, executed = p;\\
&\texttt{Movement count constraints:}\\
&\quad m, n, o, p \in \mathbb{N}^{+}, n + o = m, m \leq p
\end{aligned}
\tag{4.32}
$$

At the beginning of the mission (eq. 4.32) the buffer is filled with $m$ movements, the *Trajectory* generated from this buffer and *initial state* is *predicted*.

During *mission execution phase*, buffer contains *executed movements* and *planned movements*. Trajectory created from *initial state* and this buffer can be split into:

1. *Executed part* - trajectory portion generated and executed from *executed movements*
2. *Predicted part* - trajectory portion generated as future reference from *planned movements*

After *mission* execution there is only *executed movements* The trajectory generated from *initial state* and buffer is *Executed Trajectory*

*Note.* The part of the trajectory bounded to the pas, the part of the trajectory lies in the future. The strong point of *Movement Automaton* is its ability to work as *predictor* and *trajectory memory* at the same time.

By selecting proper time series $t_1 \dots t_n$ one can compare future or past segments of trajectory (eq. 4.32) with reference (eq. 4.31)

**Reference Trajectory Deviation** for reference trajectory given by (eq. 4.31) and *Trajectory segment* (Executed/Predicted) (4.32) with existing *State projection function* (eq. 3.25) for *time series* is given as:

$$
Deviation \begin{pmatrix} timeSeries, \\ Trajectory, \\ Reference \end{pmatrix} = \sum_{\substack{time_i \in \\ timeSeries}} \begin{pmatrix} StateProjection(Trajectory, time_i) \\ - \\ Reference(time_i) \end{pmatrix}^2 \tag{4.33}
$$

*Reference Trajectory Deviation* (eq. 4.33) is designed as discrete *Mean Square Error* function, where the *timeSeries* is set of *times* from *reference trajectory* $(point_i, time_i)$ pair. The *state projection*.

**Trajectory tracking** is defined as *dual minimization problem* where the *primary objective* is depending on the *airspace type*:

1. *Reference trajectory deviation* (eq. 4.33) in *Controlled Airspace*.
2. *Cost of Flown Trajectory* (eq. 4.28) in *Non-controlled Airspace*.

*Trajectory tracking* can be defined as optimization problem (eq. 4.34).

$$
\begin{array}{lll}
\texttt{Minimize:} & costOfTrajectoryFlown & (4.28) \\
\texttt{Minimize:} & referenceTrajectoryDeviation & (4.33) \\
\texttt{Subject to:} & & \\
 & UAS\ Dynamics & (3.27) \\
 & & (3.26) \\
 & MovementAutomatonControl & \vdots \\
 & & (3.35) \\
 & Mission & (4.6) \\
 & KnownWorld(t) & (4.20) \\
 & SafetyMargin(t) & (4.21) \\
 & HardFlightConstraints & (4.8) \\
 & SoftFlightConstraints & (4.9) \\
 & HardSpaceConstraints & (4.11) \\
 & SoftSpaceConstraints & (4.12)
\end{array}
\tag{4.34}
$$

The *reference trajectory* is given by *mission* set of *waypoints*. The *UAS* dynamics with specific *Movement Automaton* goal is to fly in *Known World* to keep *Safety Margin* form *Obstacle Space*. The *Obstacle space* is result of *Data fusion* procedure (sec. 4.1.9) combining the *sensor reading*, information sources and constraints.

**Feasible Trajectory** for *tracking problem* (eq. 4.34) is a trajectory which in addition to *basic obstacle problem* (sec. 4.2) keeps deviation from *reference trajectory* under certain threshold:

$$Deviation(timeSeries, Trajectory, Reference) \leq performanceMargin \in \mathbb{R}^{+} \tag{4.35}$$

Feasible trajectory condition (eq. 4.35) is used as *margin* for airworthiness,s and *Deviation* is used as performance indicator further in this work.

### 4.5.3 Safety

*Safety* is very broad term there are following incidents which can occur and will be discussed in (sec. 6.7.5) *Safety margin* is broad term describing minimal distance to the center of intruder/adversary, surface of obstacle, boundary of protected area.

#### Controlled airspace safety

Safety for *controlled airspace* in given *flight level* is given as list of incidents:

1. *Soft constrained zone breach* - UAS fly to *soft constraint body* or *hard constraint protection zone*, these incidents can happen, and have least avoidance priority.

2. *Hard constrained zone breach* - UAS fly to *hard constraint protection zone*, typical geo-fencing, restricted airspace breaches.

3. *Well clear breach* - UAS fly to *well clear barrel* without impacting other aircraft, via the wake turbulence or other induced physical phenomenons and vice-versa. This type of breaches are allowed in case of inevitable *near miss situations* or *Clash incidents*

4. *Near miss situation*- UAS fly to *near miss* cone/barrel, inducing wake turbulence, or other kind of flight disturbance. This incidents are allowed in very low rate (near $1 : 10^6$).

5. *Clash incident* - UAS body impacts other aircraft hull/propulsion/steering systems and components. This kind of incidents are very severe and it should never happen.

*Note.* It is assumed that flight level in controlled airspace is free of terrain, static ground obstacles, the climb/descent maneuvers are not covered in this work and they are topic for multiple dissertation thesis. For more information refer to ICAO document 4444.

*Relation* for breach of *safety margin* and *body margin* for each object is given in (tab. 4.1):

| Violation of: | Safety Margin | Body Margin |
|---|---|---|
| Soft constraint | none | Soft constraint zone breach |
| Hard constraint | Soft constrained zone breach | Hard constrained zone breach |
| Intruder | Well clear breach,<br>Near Miss situation | Clash incident |

Table 4.1: Controlled airspace margins violations incidents.

#### Uncontrolled airspace safety

Safety for *uncontrolled airspace* is applied in $F/G$ class of airspace, which is given as airspace between the *ground* level and *other airspace prevalence.*

*Note.* Clarification of controlled/uncontrolled airspace:

- *Class F* airspace is given as space between the ground level or water surface and it is constrained up to the 500 feet above ground level.

- *Class C* airspace or *Controlled airspace* is considered starting at first flight level, which is given by Air traffic control zone starting at least at 300 feets from highest

ATC zone ground point. It is measured based on Above Sea Level altitude. *This is not a problem in Portugal, because of terrain diversity, but its a huge problem in Netherlands.*

- *Class A* airspace starts at ground level and covers majority of airport infrastructure - this is not a problem, because its modeled as hard constraint, which are unbreakable in non controlled airspace.

Safety for *uncontrolled airspace* is given as list of incidents:

1. *Soft constrained zone breach* - UAS fly into *soft constrained zone* or *hard constraint protection* zone, it is allowed to happen on very low rate.
2. *Hard constrained zone breach* - UAS fly into *hard constrained zone*, only airports and critical infrastructure are considered as hard constraints, it is not allowed to happen.
3. *Intruder near miss* - UAS fly into *other aircraft near miss zone*, it is allowed to happen on very low rate in case of other intermediate threats with higher priority.
4. *Intruder clash* - UAS has a contact with other man-made aircraft, it is not allowed to happen.
5. *Adversary clash* - UAS has a contact with other flying object who did not intentionally avoided UAS. (*Bird strike, Differential games,etc..* is out of scope of this thesis).
6. *Structure harm* - UAS fly close to structure and its propulsion can damage/harm structure.
7. *Structure crash* - UAS fly into natural/man-made ground structure (building, tree, human).
8. *Ground harm* - UAS fly close to the ground and its propulsion reflection can impact Ground or UAS.
9. *Ground collision* - UAS collides with ground.

*Relation* for breach of *safety margin* and *body margin* for each object is given in (tab. 4.2):

| Violation of: | Safety Margin | Body Margin |
|---|---|---|
| Soft constraint | none | Soft constraint zone breach |
| Hard constraint | Soft constrained zone breach | Hard constrained zone breach |
| Intruder | Intruder near miss | Intruder clash |
| Adversary | none | Adversary clash |
| Structure | Structure harm | Structure crash |
| Ground | Ground harm | Ground crash |

Table 4.2: Non-controlled airspace margins violations incidents.

## 4.6 (R) Navigation requirements

*Navigation requirements* are not main part of this work, they are used to show variability of the approach for *DAA* requirements:

1. *Contextual behaviour* - change navigation and decision behaviour based on context:
   a. *Airspace type* - Controlled/Uncontrolled,
   b. *Navigation mode* - Cooperative/Emergency.
2. *Determinism* - same result for same dataset in finite time.
3. *Threat prioritization* - threats prioritization based on *context*, (tab. 4.1) and (tab. 4.2).
4. *Rule compliance* - compliance with given set of rules based on context (focus on rules of the air).

| Requirement | Evaluation metrics |
|---|---|
| Constrained space navigation | Mission scenario does not have direct path between waypoints, additional borderline cases. |
| Contextual behaviour | Avoidance system changes behaviour based on the mission and vehicle context. |
| Determinism | Multiple runs of same non-borderline scenario returns same avoidance paths. |
| Rule compliance | Rules applied are in compliance with aviation standardization. |

Table 4.3: Navigation requirements evaluation metrics.

# Chapter 5

# (R) State of art

This chapter is introducing notable works and concepts of fellow *researchers* i the field of *control theory*, *software engineering* and other essential fields used in *Detect and Avoid* systems.

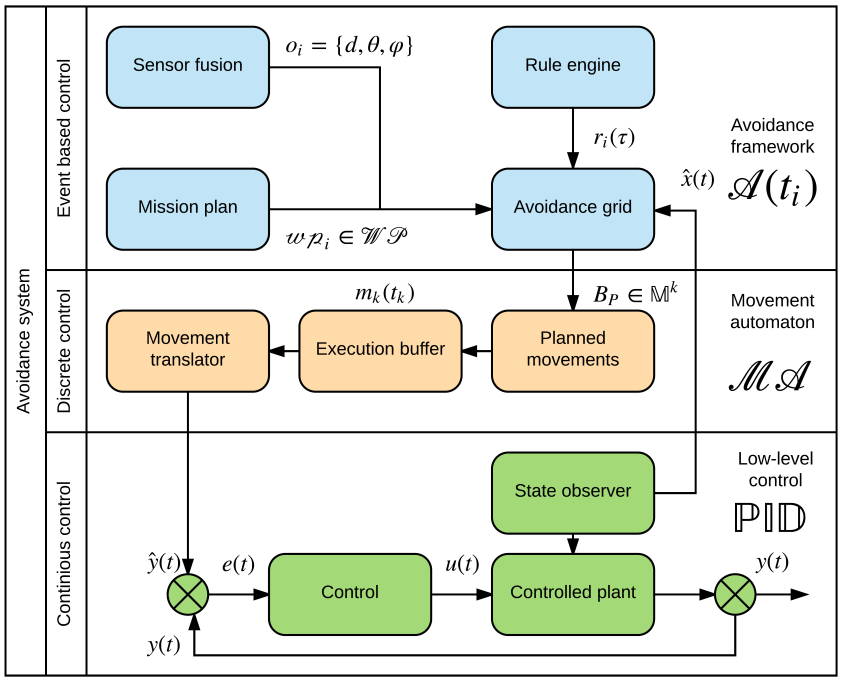



Figure 5.1: Obstacle avoidance based on Reach sets concept [7].

**Conceptual scheme:** The overall concept of *Detect and Avoid Framework* (fig. 5.1) is taking architecture from LSTS tool chain [71, 72]. The UAS part is based on *LSTS Dune* and it can be easily integrated in future.

1. *Continuous control* - is not solved in this work, its kept in scheme for reference.
2. *Discrete control* - it bridges event based *Detect and Avoid* core functionality with *Continuous control*. Its covered by *Movement Automaton* (sec. 5.1).

3. *Event based control* - covers major functionalists:

   a. *Sensor (Data) fusion* - the main feed of information, implementation of *sensor fusion* (sec. 4.1.8) and *data fusion* (sec. 4.1.9) contributing the avoidance events, introduced in (sec. 5.2).

   b. *Mission plan* - feeding actual goal and objectives to *Navigation Algorithm* (sec. 5.3) and obeying *UTM directives* (sec. 5.6).

   c. *Avoidance Grid* - using mainly *Approximation of Reachable Space* (sec. 5.4) in *Avoidance Maneuver Estimation.*

   d. *Rule engine* - enforcing UTM directives (sec. 5.6).

## 5.1 (R) Movement Automaton

**Idea:** The key idea is to create *interface* between *controlled plant* (UAV) and *Avoidance Algorithm* to ensure *Concept Reusability* at maximum degree. The concept is following:

1. *Interface consumes* discrete command chain and guides UAS along *desired trajectory.*

2. *Interface* can be used to *predict trajectory* based on *initial state* and future command chaining.

Frazolli provided the concept of *Movement Automaton* (def. 11) a specialized type of *Hybrid Automaton* (eq. 3.11), the concept is taken from his works [8, 59]. Other aspects and similarities are discussed over this chapter.

**Architecture:** The Movement Automaton can be seen as a consistent hierarchical abstraction of the continuous dynamics, in the sense outlined in [73]: *Any sequence of movement primitives generated by the Movement Automaton results by construction in a trajectory which is executable by the full continuous system. We will give a deeper meaning to hierarchical consistency.* The implementation of our movement automaton is given in (fig.5.1).

**Optimal Path Generation:** If the maneuvers are instantaneous (i.e. the UAS can transition instantaneously between two different trim trajectories), Reduction of stronger results obtained by Dubins [74] and Reeds [75] concerning optimal paths for kinematic cars on the plane (see also [76]).

**Controllability:** The systems controlled by Movement Automaton (as in [77]), is controllable according to our definition, even though it is not small-time controllable [78].

**Other Properties:** The other properties of movement automaton, like *Stability, Robustness* and other important control properties are proven in [8].

**Example:** The *example* is given in (fig. 5.2). The *States* (Barrels) are connected by *Transitions* (green arrows).

*Hover* is neutral and *initial* state, in this place the UAS stays on place and maintains altitude.

*Forward flight* is when *UAS* is flying in frontal direction with constant speed. The speed-up and slow-down is incorporated in *Transition* between *Hover* and *Forward flight* states and its takes some time to execute. *Transitions* between Turning states and *Flight forward* state are almost instant.

*Steady turn left/right* is when *UAS* is flying in frontal direction and starts steady turning left or right.

*Note.* UAS in (fig. 5.2) ignores the vertical maneuvering and it is expected to fly on horizontal plane.

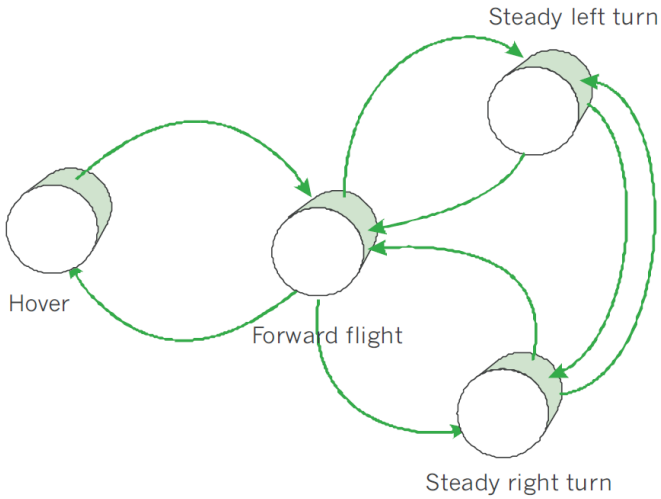



Figure 5.2: Movement Automaton for Copter UAS [8].

**Used concepts:** The movement automaton is essential part of this work. Supporting the idea, that *Obstacle avoidance framework* can be platform independent.

The *Movement Automaton* is used as is (sec. 3.3.3). The implementation is described in (sec. 6.2). The testing configuration was given in (tab. 7.2).

## 5.2 (R) Sensor (Data) Fusion - Input Interface

**Idea:** There is *Need for abstract representation* of *operational space*, which is independent of used sensors, technologies, information sources. The universal obstacle avoidance system should have *portability property*. Our previous work *Obstacle avoidance framework based on reach sets* [7] have introduced similar concept of *control interface*.

The original concept was using cell status interpretation, which was hardwired to LiDAR technology. The new demand is to incorporate concepts of *visibility*, *reachibility* and *obstacle probability*. The base methodst for *Statistical Sensor Fusion* were outlined in [79].

**Key Concept:** *Data fusion interface* (sec. 4.1.9) - interface to fuse sense data from various online, offline, cooperative, non-cooperative sources into single scalable space and trajectory evaluation procedure.

**Related work:** UAS specific sensor fusion has been proposed by Ramsay in [26]. *Next generation avoidance concept* [80] is introducing concept of higher level sensor fusion called *data fusion*.

The uncertainty and properties in *Remotely Piloted Systems* have been discussed in [81]. The work provided concept of various performance ratings like visibility and obstacle rating, more details have been given in [82]. This ratings were modeled only for operator decision making [83], results are usable for automated decision making and space assessment.

*Probabilistic trajectory assessment* has been firstly proposed in [84] where trajectory was tracking and predicting *safety properties* along.

*Game theory* viewpoint is firstly used in [85]. Probabilistic path planning using safety zones similar to cell classification of this work have been used in [86].

Probabilistic path search similar to our reach set representation using rapidly exploring path trees have been used in [87, 88]. Relationship between classic grid search and probabilistic lattice search have been established in [89]. A probabilistic approach for trajectory estimation via reduced lattice search is known from 1986 from work of Gessel [90] lattice paths were enumerated via movement sequences and similar technique is used in our reach set estimation method using movement automaton. Pruning methods comparison and complexity can be found in [91].

Overall concepts of probabilistic sets have been given by Hirota in [92]. Free flight safety rating similar to our reachability concept have been presented in [93].

**Shortcomings:**

1. *Hierarchical calculation* - there is need to calculate *avoidance trajectory* for incremental constraint applications. For example:
   a. Calculate *Minimal escape path* covering physical obstacles and intruders.
   b. Apply next level of constraints, like airspace restrictions and some virtual constraints. Then calculate path if exists, continue.
   c. Apply nice to have constraints, like non lethal weather, recalculate path.
2. *Source Reliability Evaluation* - reliability evaluation is empirical process usually done by hand. The result aggregation is not standardized. There can be multiple sources of same rating, for example visibility, which needs to be aggregated into one.
3. *Ambiguous rating definition* - There is multiple definitions especially for *Reachibility rating* in works [87, 88, 90].

**Improvements in Our Work:** *Hierarchical calculation* is addressed in *Mission Control run* (sec: 6.7.3) where threats are hierarchically applied based on *severity*.

*Source reliability evaluation* is addressed in *Static Obstacles* (sec. 6.5) and *Moving Obstacles* 6.6). The main rating for *Detected obstacle, Map Obstacle* and *Visibility* of space are established there.

*Clear rating definition* - the *Reachibility* of space portion and *Safety* rating for trajectory are established in *Avoidance Grid Run* (sec. 6.7.2)

## 5.3 (R) Navigation Algorithms

**Idea:** The basic idea is to provide hierarchical*navigation frame* with *some optimal path search capabilities.*

**Standard Navigation:** The standard navigation is given as *expected cost optimization problem* for *future cost function* (eq. 4.30). The key concept of navigation algorithm was fully taken from [94]. The decision was made based on navigation survey [95]. The *descent* for landing is out of scope in this work, can be found in [96]. The navigation principle is roughly described in (sec. 6.7.3).

**Maze Solving Capabilities:** The *maze solving capability* is usable in *controlled airspace* where 2D maze solving algorithms are applicable. The notable implementation was for *micro mouse robot* based on right hand rule [97]. Flood fill algorithm is partially usable for 3D environment [98]. The application of *maze solving* was given in case study [99].

**Hybrid Automaton Path Planning:** A hybrid automaton path planing based on $A*$ algorithm was given by Richards in [100]. The key idea was to use *hybrid automaton* (eq. 3.11) as a reference generator. This idea was taken and formulated as *Movement Automaton Predictor mode.*

The similar idea where *potential fields* were used as *intruder model* and path was re-planned based on events is given in [101].

**Mode Switch:** The *Mode Switch Control* idea has been presented in [102]. There were definition of behavioural switch between:

1. *Navigation Mode* - navigation control and behaviour was used.
2. *Task Specific Mode* - mode specific for tasks, authors were using modes for search and rescue.

This concept will be reused, the *Task specific mode* will be *Emergency Avoidance Mode* in Our case. The triggering events and switch conditions will be defined in (sec. 6.7.3).

**Used concepts:** The *Following concepts* were used in navigation loop:

1. *Standard navigation* taken from [94] minor implementation changes using offline optimization. The purpose of navigation loop is to bring us closest to the waypoint, if its reachable. Navigation example (sec. 7.4.3).
2. *Maze solving capabilities* partially taken as secondary functionality based on [98]. The purpose is the *looping prevention.* The example was given in (sec. 7.3.3) .
3. *Mode switch* partially taken as main feature from [102], the triggering events were identified and defined by author and can be found over (chapter 6).

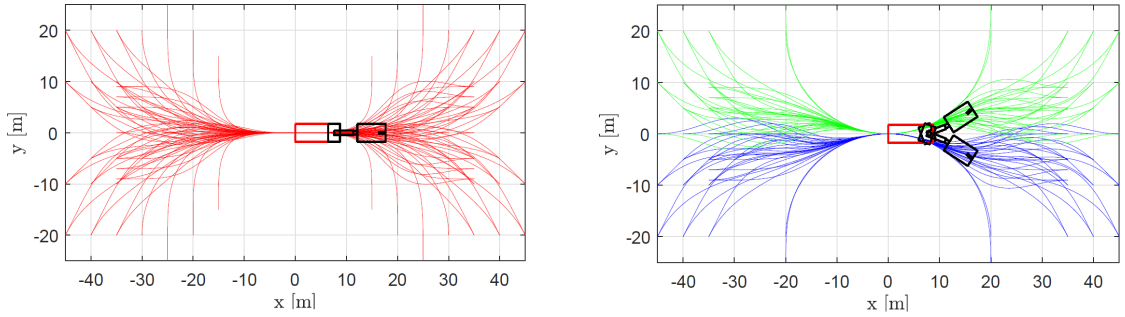
## 5.4 (R) Reach Set Estimation

**Idea:** The basic idea for *Discrete Reach set Estimation method* is taken from [9]. The focus of their work is to generate paths that are kinematically feasible. Path following controllers in order to find techniques to stabilize the system around these paths [103, 104].

Lattice-based planners have been deployed with great success on several robotic platforms [105, 106, 107, 108, 109]. However, a problem with lattice-based approaches is the exponential complexity in the dimension of the state space which can limit the use for more complicated models.

The optimization problem was solved real-time by *Avocado solver* [110].

**Example:** The *example of movement lattice* is given in (fig. 5.3). Truck (black rectangle) is towing Trailer (red rectangle). The *state* have only one *reach set impacting variable - trailer displacement*. When trailer displacement is 0° (fig. 5.3a) the lattice representation of *Reach Set* is different that in case of small left/right tilt (fig. 5.3b).




(a) Trailer displacement 0°.

(b) 12° (green) −12° (blue).

Figure 5.3: *Movement set primitives* for Lattice Based Movement Planning. [9].

**Benefits:** Presented method of *Lattice Search* is de-facto *Reduced Reach Set Approximation* in open space for *Truck-Trailer* system.

The idea of *Movement primitives* is very close to *Movement Automaton* (def. 11), which can be used as *control interface* effectively.

The *Constraints* of obstacle set in *Known world* (sec. 4.1.10) are supported to some degree.

**Shortcomings:** There are following shortcomings which were identified in this approach:

1. *Limited system dimension* - given method works in *real time* only if dimension of *system state space* does not exceed $4^{th}$ rank.

2. *Real time optimization* - real time optimization is main cause of *limited system dimension*. If the decision time can be discrete (which movement automaton enforces) then offline optimization can be used.

3. *Continuous space disparity* - the example (fig. 5.3) shows there are member variables of *State Space* which significantly impacts the shape of lattice (reach set estimation). This is not a problem in real-time environment. The discretization of *Time domain* raises this as a shortcoming.

4. *Trajectory Tracking* - approach generates *Continuous Domain* reference signal. For *Discrete Domain* it is necessary to address this issue.

**Improvements in Our Work:** *Limited system dimension* - the discretization due the higher system dimension and increased maneuver complexity goes hand-in-hand with *pre-calculation* of the *Reach Set*. This shortcoming is addressed in (sec. 6.4.2).

*Real time optimization* - replaced by *Discrete offline optimization problem*. The *general cost function* is given in (eq. 4.30). The optimization problem solved in this work is defined in (eq. 4.34).

*Continuous space disparity* - The *pre-calculated reach set estimation* can be valid with small *marginal error* for some region in *system state space*. The dynamic method for state space segmentation can be used [111]. This aspect is not addressed in this work, because it is strongly depending on the system behind movement automaton.

*Trajectory Tracking* - The *movement automaton* (def. 11) in Control Mode can be used to track reference trajectory in form of *Movement Buffer*(def. 9). Other option is to use *thick waypoint trajectory tracking for UAV* like in [112] or [113]. The work will use only *Movement Automaton* as controller/predictor.

## 5.5 (R) Testing Framework

**Idea:** *Reuse LSTS toolchain architecture* for DAA testing framework.

**LSTS Toolchain:** Software architecture used in modern unmanned aerial vehicles must be system independent and scalable. Writing own control software for unmanned aerial vehicle and ground station is unthinkable in current state of art. Most notable framework for unmanned aerial vehicle development is LSTS tool chain from University of Porto [114]. This tool chain is widespread in other universities and multiple independent applications are based on it [115].

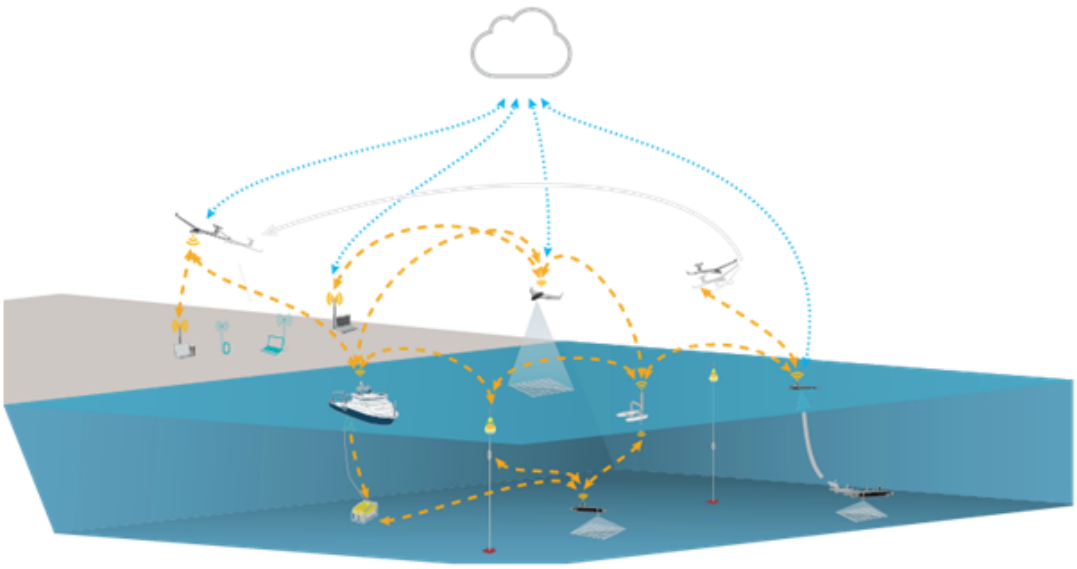


Figure 5.4: Example of LSTS toolchain deployment in real environment [10]

Example of software architecture implementation is shown in figure 5.4 LSTS HUB (cloud iconography) is collecting all important data from currently executing missions. Data are transferred via REST API (dotted blue lines) to HUB. Commanded vehicles can be unmanned copters, planes, ships, submarines or floating sensors compounds. Each vehicle has installed DUNE which is responsible for vehicle command and ground control station communication. Deployment, range of command and status messages can vary. Ground station can be implemented on personal computer (any platform) in NEPTUS environment or mobile platform (Android OS) in ACCU environment. Ground station environments are customizable and open source. Layout of ground station can be customized to need of current mission via plugins or console configuration. Vehicles and ground stations are communicating via IMC protocol (orange dashed lines). Communication channel is platform independent.

*Glued* is a Debian-based open source operating system which was initially released in 2010. Over the years it has become fairly widespread and been provided continuous updates. Meanwhile, a notable development community has emerged, where advice and support can be received as more applications are investigated. Some of the merits of the operating system are its reliability and customizability. Operation system is developed for unmanned autonomous vehicles. It is widely used in air, sea and land vehicles. Customization allows the operating system to be tailored to the specific usage. For this purpose, this includes stripping off functionality which is not required, thus releasing resources to be focused on the essential tasks. Glued is the operating system favored by the Beaglebone development community, and thus there exist considerable amounts of helpful documentation for this set-up.

*DUNE: Unified Navigational Environment* is an on-board software solution for unmanned vehicles. Multiple applications already run on this software. Almost any extension can be added to this to this environment. The software solution provides means for interacting with the connected components as well as control, navigation, supervision and plan execution. It is both CPU architecture independent and OS independent. It is written in C++ and developed by LSTS: Underwater Systems and Technology Laboratory.

*Neptus* [10, 116, 117] is a command and control software operated from a ground station. It is designed to operate well together with Dune and was also developed by LSTS. Neptus provides tools for remotely monitoring UAVs and assigning plans and commands in real-time missions, supporting multiple connections dynamically. Furthermore, it provides possibilities for both simulating missions and reviewing previously performed operations. This is presented in a customizable interface equipped with map layers and control panels. It is written in Java and available for both Windows and Linux systems.

The *Inter-Module Communication* [118] (IMC) protocol was developed by LSTS to provide reliable communication between the systems. The protocol is message-oriented, such that messages can be sent and received from a bus which connects independently run threads or systems. Thus it functions as a method of communication between tasks internally in Dune, and can also be passed to and from other vehicles or computers running Dune or Neptus. IMC is platform independent at multiple messages have been already developed and supported by both DUNE and Neptus (around 400 status/command messages).

*HUB* is communication hub for data dissemination and situation awareness. This module is responsible for complex mission execution, when cooperation of multiple pilots/vehicles is required. This module can be imagined as airport tower center, which is monitoring all flights in airport (operation site) area.

**Movement Automaton Control** Marconi used *hybrid automaton* with forced *State Switch* via buffer [119]. The key concept is that *Automata* state switch is forced as *external source command*. Our *Movement Automaton* implementation know only *forced state switch* like in [59].

**Used concepts:** Most of the architecture was re-used in our approach, the concept of *Rule engine* (sec. 6.9) was introduced to cover missing *UTM* related functionality. The implementation in Matlab was influenced by Alessandeti works [120, 121]. The other aircraft dynamic and control related concepts were taken from [122].

## 5.6 (R) UTM Services

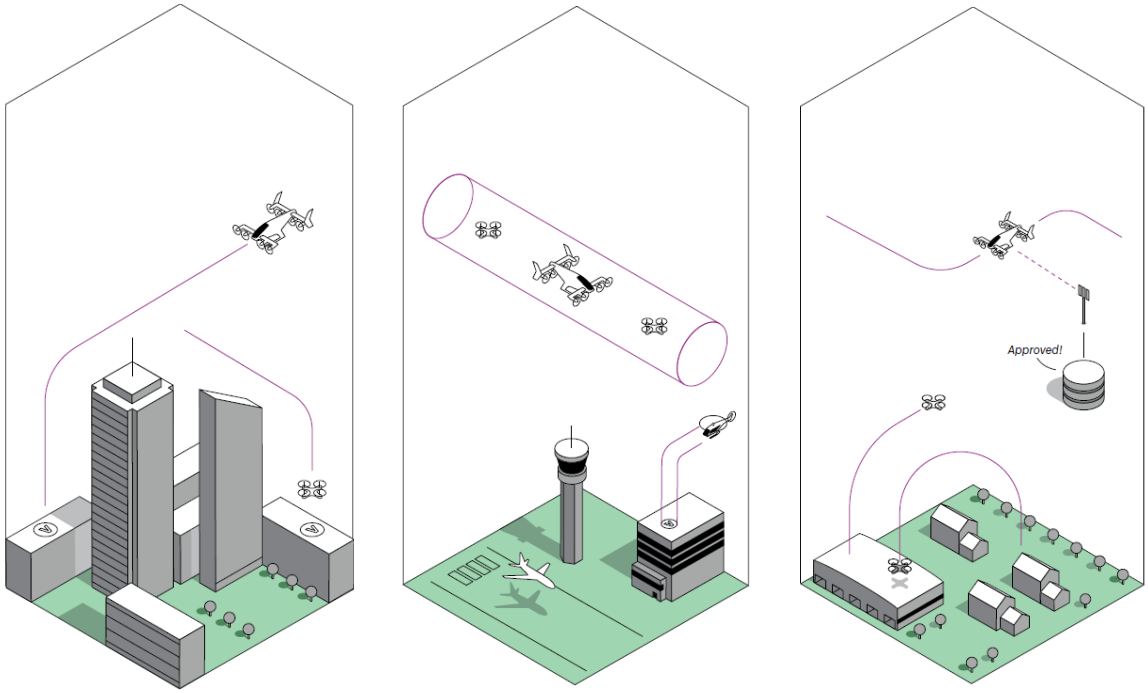
**Idea:** Take the Airbus UTM concept [123] combine it with *EUROCONTROL* concept [124] to obtain legal framework. *Provide conflict resolution functionality* for *Controlled Airspace*:

1. *Collision Detection* - define minimal required functionality for collision detection.
2. *Collision Resolution* - implement *Rules of the Air* [125].

The *implementation* of *UTM services* described in (sec. 2.4) is given in (sec. 6.8).

**UTM Operation Modes:** *defined in* [123] are following:

1. *Free route* (fig. 5.5a) is when aircraft can fly any path, so long as their planned path is coordinated with and de-conflicted from the paths of other aircraft by a traffic manager and approved based on calculated risk. Free routing is being introduced worldwide, such as free route airspace. This allows commercial flights to freely plan their route through participating sectors during cruise. There is less freedom for an aircraft in this situation than in basic flight, since its request may be rejected.
2. *Corridors* (fig. 5.5b) are defined volumes in space, useful for managing airspace in high demand or to manage traffic flow and separation. Coordination is necessary to ensure safety in this airspace. A corridor may take on many different shapes. Aircraft are often guided inside corridors using predetermined routes analogous to approach procedures used worldwide today.
3. *Fixed route* (fig. 5.5c) are used to ensure safety when there is high traffic density or in any location where structure is required to ensure safe operations. This could include locations such as airports or warehouses. These routes could be constructed or modified dynamically based on calculated risk. The most restrictive version is a predetermined path, where the only variable is when an aircraft is at a specific point in the path.




(a) Free route. (b) Corridors. (c) Fixed route.

Figure 5.5: UTM Operation Modes.

**Used Concepts:** The implementation of our UTM services is focused on *Free route mode* (fig. 5.5a). The *Corridors* and *Fixed routes* are just additional *space/time constraints*.

# Chapter 6

# (R/W) Approach

There are few levels of *Avoidance* based on the *remaining time to collision*. These levels are summarized in (fig. 6.1).

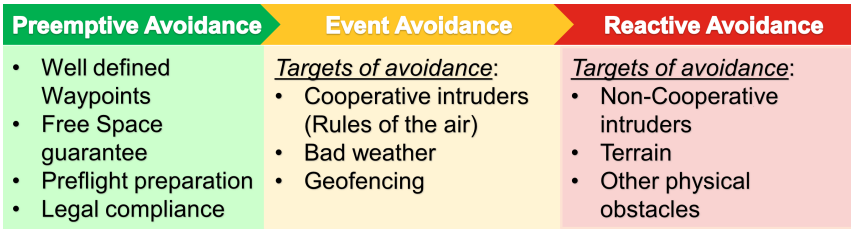



Figure 6.1: Avoidance levels based on reaction time.

This work will focus on handling *Event Avoidance* and *Reactive Avoidance* and the *Avoidance Path* will be calculated using *Reach set Based Methods*.

The *Preemptive Avoidance* is trying to remove any possible threat prior the flight. The risk mitigation is tedious and its done only when necessary. Even the best *preemptive* avoidance could fail.

The *Reactive Avoidance* is solving most urgent situations with very short reaction opportunity. This work focus on physical obstacles and terrain. Non cooperative intruders are considered partially. The adversary behaviour was omitted.

The *Event Avoidance* has more opportunity to react. Some threats are know prior the flight (geo-fenced areas, ...). The future UTM implementation is also considered as *Event Avoidance*, due the time horizon and authority enforcement.

**Basic Idea:** Create deterministic finite-time *Reactive Avoidance* based on *Reach sets* to assure *trajectory feasibility*. Enhance method with set of the rules to enable handling more complex situations.

The *Discretization* is the key to assure calculation in finite time. Finite *partition* of *operational space (Known World)* and finite representation of *Reach set* guarantees finite count of calculation steps. Aircraft conflict prediction mentioned in [126].

## 6.1 (R) Overview

The *Overview* is based on *Existing* Emergency avoidance framework [7] (fig. 5.1). To achieve goals defined in *Problem Definition* (sec. 4.2, 4.4) following *Avoidance Framework Concept* (fig. 6.2) is proposed:

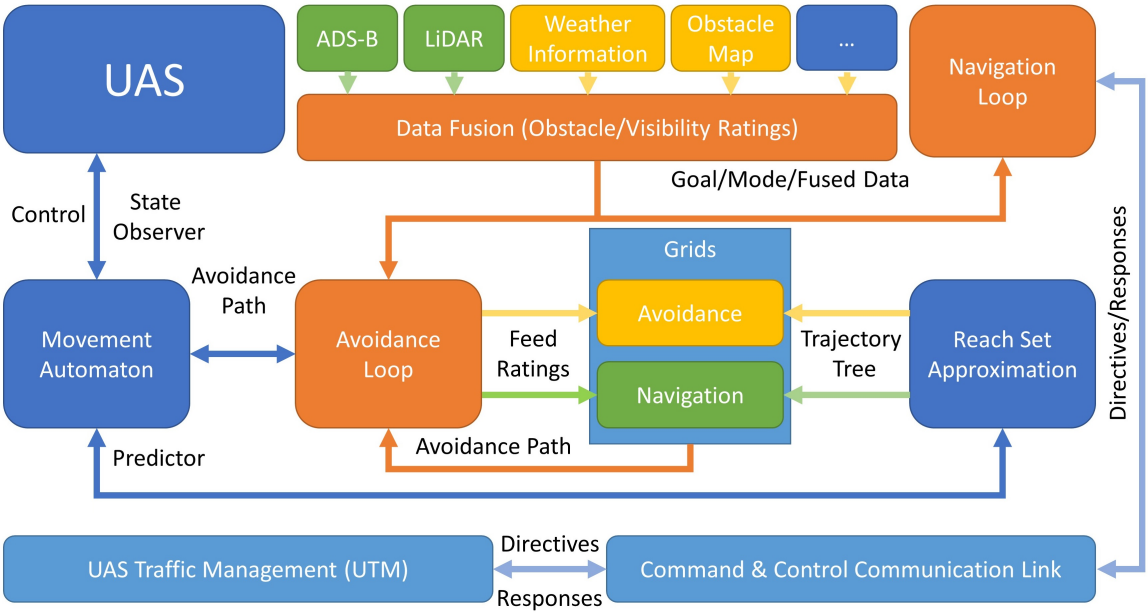



Figure 6.2: Avoidance Framework Concept.

**Structure of Avoidance Framework:**

1. *Unmanned Aircraft System* (UAS) (Role: Controlled Plant) - the *UAS* is controlled via *interface* implemented as *Movement Automaton*. The model used is described in (sec. 6.2.1).

2. *Movement Automaton* (Role: Control Interface/Predictor) - consumes *Discrete Command Chain* to generate discrete *reference trajectory*, it can be also used as a predictor of *future UAS states* (sec. 6.2.4). The movement Automaton used in this work is given in (sec. 6.2.2).

3. *Sensor Field* (Role: Surveillance Providers), following sensors were considered in this work:

   a. *LiDAR* (Static obstacle detection) - detection of physical obstacles (sec. 6.5.1)

   b. *ADS-B* (Intruder UAS/Plane detection) - detection of intruders whom are broadcasting their position and heading sometimes with future plans and additional parameters. The *intersection models* are given in (sec. 6.6.1, 6.6.2, 6.6.3, 6.6.4).

4. *Information Sources* (Role: Known World Information Enhancers):

    a. *Obstacle Map* (Static Restriction Source) - imposing static soft/hard constraints on *Known Word/Operational Space*. Static constraints are given in (sec. 6.5.3).

    b. *Weather Information* (Static/Dynamic Restriction Source) - imposing static/moving soft/hard constraints on *Known World/Operational Space*. Moving constraints are given in (sec. 6.6.5).

    c. *Other Airspace Restrictions* - like restricted airspace, geo-fencing and other future constraint sources, all of them are covered by *Static/Dynamic Constraints* for now.

5. *Data Fusion* (Role: Sensor Input Interface) - is the unifying interface to asses *Operational State Properties* mainly *Obstacle Rating*, *Visibility*, *Map Obstacle Rating*, *Intruder Rating* for portion of the space. The partial *ratings* are proposed in related sections. The data fusion procedure with *defuzzyfication* and final assessment into space sets is outlined in (sec. 6.7.1)

6. *Reach Set Approximation* (Role: Reachability Estimator) - as *data fusion* is providing the situation assessment, the *Reach set* is providing maneuvering capability assessment. The introduction is given in (sec. 6.4), the properties are defined in (sec. 6.4.1), the approximation methods with constrained expansion are outlined in (sec. 6.4.3, 6.4.4, 6.4.5, 6.4.6). The reach set estimation is main contribution of this work.

7. *Grids: Navigation/Avoidance* (Role: Operation Space Segmentation & Situation Evaluation) - space discretization in polar coordinates grid, different reach sets are used for different grid type, defined in (sec. 6.3).

8. *Avoidance loop* (Role: Short Term Decision Maker) - using data from *Sensor fusion* in *Avoidance/Navigation Grid* trimming *Reachable Space* approximated by *Reach Set* generating feasible *Avoidance Path*. *Avoidance Path* is fed to controlling *Movement Automaton*. The Goal is given by *Navigation Loop*. Avoidance loop is given in (sec. 6.7.2).

9. *Navigation loop* (Role: Long Term Decision Maker) - using data from *Avoidance Loop*, *Mission plan* and *UTM* directives defines the current long term navigation goal. Details given in (sec. 6.7.3).

10. *Command and Control Communication Link* (C2 Link) (Role: Communication Link) - standard communication link with sufficient reliability.

11. *UAS Traffic Management* (UTM) (Controlled Airspace Authority) - checking possible collisions and enforces counter-measurements. Details given in (sec. 6.8).

**Communication in Avoidance Framework:**

1. *UAS* $\leftrightarrow$ *Movement Automaton* - sharing *actual system state*, commanding the UAS platform.
2. *Reach Set* $\leftrightarrow$ *Movement Automaton* - predicting set of feasible trajectories for given situation.
3. *Reach Set* $\leftrightarrow$ *Grids* - providing trajectory set depending on active mode (Navigation/Emergency Avoidance).
4. *Avoidance Loop* $\leftrightarrow$ *Data Fusion* - assessing the situation in *operational space* based on sensor readings/information sources.
5. *Avoidance Loop* $\leftrightarrow$ *Navigation Loop* - determining long term goal based on situation assessment and UTM directives.
6. *Avoidance Loop* $\rightarrow$ *Grids* - feeding assessment data and constraints into selected operational space Grid.
7. *Grids* $\rightarrow$ *Avoidance Loop* - returning feasible and *cost effective* avoidance path after situation assessment and *Reach set* pruning.
8. *Avoidance Loop* $\rightarrow$ *Movement Automaton* - issuing and monitoring movement commands based on actual *avoidance strategy*.
9. *Navigation Loop* $\leftrightarrow$ *C2 Link* $\leftrightarrow$ *UTM* - communication to receive directives and send fulfillment.

## 6.2 (R) UAS Model and Control

The key feature of *Movement Automaton* is to interface *continuous-control signal* as the *discrete command chain*. Following topics are introduced in this section:

1. *UAS Nonlinear Model* (sec. 6.2.1) - simple plane model used in this work as *controlled plant*.
2. *Movement Automaton* (sec. 6.2.2) - movement automaton for *UAS Nonlinear Model* constructed from scratch.
3. *Segmented Movement Automaton* (sec. 6.2.3) - for more complex systems the *State Space* can be *separated into Segments* and *segment movement automaton* is used to generate *thick reference trajectory*.
4. *Reference Trajectory Generator* (sec. 6.2.4) - other use of *Movement Automaton* as predictor for *reference trajectory calculation*.

### 6.2.1 (R) UAS Nonlinear Model

**Motivation:** Simplified rigid body kinematic model will be used. This model have decoupled roll, yaw and pitch angles. The focus is on *reach set approximation methods*, therefore *UAS model* is simplified.

**State Vector** (eq. 6.1) defined as positional state in euclidean position in right-hand euclidean space, where *x, y, z* can be abstracted as latitude, longitude, altitude.

$$state = [x, y, z, roll, pitch, yaw]^T \tag{6.1}$$

**Input Vector** (eq. 6.2) is defined as linear velocity of UAS $v$ and angular speed of rigid body $\omega_{roll}, \omega_{pitch}, \omega_{yaw}$.

$$input = [v, \omega_{roll}, \omega_{pitch}, \omega_{yaw}]^T \tag{6.2}$$

Velocity distribution function (eq. 6.3) is is defined trough standard rotation matrix and linear velocity $v$, oriented velocity $[v_x, v_y, v_z]$ given by (eq. 6.4).

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v\cos(pitch)\cos(yaw) \\ v\cos(pitch)\sin(yaw) \\ -v\sin(pitch) \end{bmatrix} \tag{6.3}$$

**UAS Nonlinear Model** (eq. 6.4) is given by *first order equations:*

$$\begin{aligned} \frac{\partial x}{\partial time} &= v\cos(pitch)\cos(yaw); & \frac{\partial roll}{\partial time} &= \omega_{roll}; \\ \frac{\partial y}{\partial time} &= v\cos(pitch)\sin(yaw); & \frac{\partial pitch}{\partial time} &= \omega_{pitch}; \\ \frac{\partial z}{\partial time} &= -v\sin(pitch); & \frac{\partial yaw}{\partial time} &= \omega_{yaw}; \end{aligned} \tag{6.4}$$

### 6.2.2 (R) Movement Automaton for UAS Model

**Motivation:** An *UAS Nonlinear Model* (eq. 6.4) can be modeled by *Movement Automaton* (def. 11).

**Movement Primitives** by (def. 6) are given as (eq. 3.15). To define primitives the *minimal time* is $1s$. The *maximal duration* is also $1s$.

**Assumption 6.** *Let assume that* transition time *of* roll, pitch, yaw, linear velocity *is* $0s$.

Under the assumption (as. 6) the *movement transitions* (def. 7) have 0 duration.

*Note.* The assumption (as. 6) can be relaxed under condition that *path tracking controller exists.*

**Movements** (def. 8) for *fixed step* $k$ we start with discretization of the input variables.
The *linear velocity* in text step is given:

$$v(k+1) = v(k) + \delta v(k) \tag{6.5}$$

The *roll, pitch, yaw* for next step are given

$$\begin{aligned} roll(k+1) &= roll(k) + \delta roll(k) \\ pitch(k+1) &= pitch(k) + \delta pitch(k) \\ yaw(k+1) &= yaw(k) + \delta yaw(k) \end{aligned} \tag{6.6}$$

The $\delta v(k)$ is *velocity change*, $\delta roll(k)$, $\delta pitch(k)$, $\delta yaw(k)$, are *orientation changes* for current discrete step $k$. If the duration of *transition* is $0s$ (as. 6) then 3D trajectory evolution in discrete time is given as:

$$\begin{aligned} x(k+1) &= x(k) + v(k+1)\cos(pitch(k+1))\cos(yaw(k+1)) &&= \delta x(k) \\ y(k+1) &= y(k) + v(k+1)\cos(pitch(k+1))\sin(yaw(k+1)) &&= \delta y(k) \\ z(k+1) &= z(k) - v(k+1)\sin(pitch(k+1)) &&= \delta z(k) \\ time(k+1) &= time(k) + 1 &&= \delta time(k) \end{aligned} \tag{6.7}$$

The $\delta x(k)$, $\delta y(k)$, $\delta z(k)$ are positional differences depending on *input vector* for given discrete time $k$:

$$input(k) = \begin{bmatrix} \delta x(k), \delta y(k), \delta z(k), \delta v(k), \\ \delta roll(k), \delta pitch(k), \delta yaw(k), \delta time(k) \end{bmatrix}^T \tag{6.8}$$

The *state vector* for discrete time is given:

$$state(k) = \begin{bmatrix} x(k), y(k), z(k), v(k), \\ roll(k), pitch(k), yaw(k), time(k) \end{bmatrix}^T \tag{6.9}$$

The nonlinear model (eq. 6.4) is then reduced to *linear discrete model* (eq. 6.10) given by *apply movements* function (eq. 6.5, 6.6, 6.7).

$$state(k+1) = applyMovement(state(k), input(k)) \tag{6.10}$$

**Movement Set** for linear discrete model (eq. 6.10) is defined as set of extreme unitary movements on main axes (tab. 6.1) and diagonal axes (tab. 6.2).

| $input(movement)$ | Straight | Down | Up | Left | Right |
|---|---|---|---|---|---|
| $\delta x(k)[m]$ | 1.00 | 0.98 | 0.98 | 0.98 | 0.98 |
| $\delta y(k)[m]$ | 0 | 0 | 0 | 0.13 | -0.13 |
| $\delta z(k)[m]$ | 0 | -0.13 | 0.13 | 0 | 0 |
| $\delta roll(k)[°]$ | 0 | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[°]$ | 0 | 15° | -15° | 0 | 0 |
| $\delta yaw(k)[°]$ | 0 | 0 | 0 | 15° | -15° |

Table 6.1: Input values for main axes movements.

| $input(movement)$ | Down-Left | Down-Right | Up-Left | Up-Right |
|---|---|---|---|---|
| $\delta x(k)[m]$ | 0.76 | 0.76 | 0.76 | 0.76 |
| $\delta y(k)[m]$ | -0.13 | 0.13 | 0.13 | -0.13 |
| $\delta z(k)[m]$ | -0.13 | -0.13 | 0.13 | 0.13 |
| $\delta roll(k)[°]$ | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[°]$ | -15° | -15° | 15° | 15° |
| $\delta yaw(k)[°]$ | 15° | -15° | 15° | -15° |

Table 6.2: Input values for diagonal axes movements.

*Note. Movement set* in shorten form is given as

$$MovementSet = \left\{ \begin{array}{c} Straight, Left, Right, Up, Down, \\ DownLeft, DownRight, UpLeft, UpRight \end{array} \right\} \tag{6.11}$$

**Trajectory** by (def. 10) for initial time $time = 0$ , initial state $state(0)$ and *Movement Buffer* (from def. 9):

$$Buffer \in MovementSet^{*}(eq.6.11), \quad |Buffer| \in \mathbb{N} \tag{6.12}$$

Trajectory (eq. 6.13) is then given as the time-series of discrete states:

$$Trajectory(state(0), Buffer) = \left\{ \begin{array}{c} state(0) + \sum_{j=0}^{i-1} input(movement(j)) : \\ i \in \{1 \dots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{array} \right\} \tag{6.13}$$

Trajectory (eq. 6.13) is ordered set of states bounded to discrete time $0 \dots n$ , where $n$ is member count of *Buffer*. Trajectory set has $n+1$ members:

$$Trajectory(state(0), Buffer) = \left\{ \begin{array}{l} state(0) = state(0) + \{\} \\ state(1) = state(0) + input(movement(1)) \\ state(2) = state(0) + input(movement(1)) + input(movement(2)) \\ \qquad \vdots = \vdots \\ state(n) = state(0) + input(movement(1)) + \dots + input(movement(n)) \end{array} \right\} \tag{6.14}$$

**State Projection** (eq. 6.15) for the *Trajectory* (eq. 6.13) is given as follow:

$$StateProjection(Trajectory, time) = Trajectory.getMemberByIndex(time + 1) \tag{6.15}$$

*Note. Movement Automaton* for system (eq. 6.4) with given (as. 6) is established with all related properties (sec. 11).

### 6.2.3 (R) Segmented Movement Automaton

**Motivation:** Constructing *Movement Automaton* for more complex system can be tedious. Used *Movement Automaton* for *UAS system* (6.4) has decoupled control which is not true for most of the copters/planes [62].

**Partitioning UAS State Space:** Proposed movement automaton is defined by its Movement set (tab. 6.1,6.2). Those can be scaled depending on maneuverability in the *Initial state* $state(0)$:

1. *Climb/Descent Rate* $\delta pitch_{max}(k)$ - the maximal climb or descent rate for Up/Down movements.
2. *Turn Rate* $\delta yaw_{max}(k)$ - the maximal turn rate for Left/Right movement.
3. *Acceleration* $\delta v_{max}(k)$ - the maximal acceleration in cruising speed range.

**Definition 20.** *State Space partition* Maneuverability *is depending on* Initial State. *There can not be the infinite count of* Movement Automatons.

*The state space* $StateSpace \in \mathbb{R}^n$ *can be separated into two exclusive subsets:*

$$StateSpace = [ImpactStates, NonImpactingStates] \tag{6.16}$$

*The* Impacting states *are states which bounds the* Maneuverability*:* $\delta pitch_{max}(k)$, $\delta yaw_{max}(k)$, $\delta v_{max}(k)$. *For each* impact state *is possible to define upper and lower boundary:*

$$\forall impactState \in ImpactStates, \exists : \\ lower(impactState) \leq value(impactState) \leq upper(impactState) \tag{6.17}$$

*The bounded interval of impact state can be separated into distinctive* impact state segments *like follow:*

$$impactState \in [lower, upper] : \\ \{[lower, separator_1[\cdots \cup \ldots [separator_i, separator_{i+1}[\cdots \cup \ldots \\ \cdots \cup \ldots [separator_n, upper]\} = \\ = impactStateIntervals(impactState) \quad (6.18)$$

*Note.* The interval length depends on model dynamics. The rule of thumb is to keep maximal climb/descend/turn/acceleration rates near constant value.

*When partitioning of* all impact States *finishes, the count of partitions is given as product of* count of partitions *for each member of* Impact States*:*

$$partitionCount = \prod_{impactState \in}^{ImpactStates} |impactStateIntervals(impactState)| \quad (6.19)$$

*Note.* Try to keep the count of partitions to minimum, each new interval increases the count of partitions geometrically.

*There is finite number n of* Impacting States*, these are separated into impactState−Intervals$_i$ with respective index $i \in 1 \ldots n$. The* segment *with index defining position used* impacting state *intervals is given as* constrained space*:*

$$Segment(index) = \begin{bmatrix} impactState_1 \in impactStateIntervals_1[index_1], \\ \vdots \\ impactState_n \in impactStateIntervals_n[index_n], \\ \vdots \\ NonImpactingStates \end{bmatrix} \quad (6.20)$$

*Each* Segment *covers one of impacting state intervals combination, because the original intervals are exclusive, also* Segments *are exclusive. The* union *of all segments covers* State Space*:*

$$StateSpace = \bigcup_{\forall \quad index \in |impactStateIntervals|^n} Segment(index) \quad (6.21)$$

**Segmented Movement Automaton:** The segmentation of *state space* is done in (def. 20) any *state* belongs exactly to *Segment* of *State Space.* For each *Segment* in *State Space*it is possible to assess: *Climb/Descent Rate* $\delta pitch_{max}(k)$, *Turn Rate* $\delta yaw_{max}(k)$, and, *Acceleration* $\delta v_{max}(k)$.

**Definition 21.** *Movement Automaton for Segment(index)*

*For for Model(eq. 6.10) with State (eq. 6.9) the input vector (eq. 6.8) is for position* $[x, y, z]$ *and velocity defined like:*

$$\begin{aligned}
\delta x(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \cos(\delta yaw(k)) \\
\delta y(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \sin(\delta yaw(k)) \\
\delta z(k) &= -(v(k) + \delta v(k)) \cos(\delta pitch(k)) \\
\delta v(k) &\in [-\delta v(k)_{max}, \delta v(k)_{max}]
\end{aligned} \tag{6.22}$$

The acceleration $\delta v(k)$ is in interval $[-\delta v(k)_{max}, \delta v(k)_{max}]$, usually set to 0 $ms^{-1}$. The change of the orientation angles for *Movement Set* (eq. 6.11) is given in (tab. 6.3,6.4).

| $input(movement)$ | Straight | Down | Up | Left | Right |
|---|---|---|---|---|---|
| $\delta roll(k)[^\circ]$ | 0 | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[^\circ]$ | 0 | $\delta pitch_{max}$ | -$\delta pitch_{max}$ | 0 | 0 |
| $\delta yaw(k)[^\circ]$ | 0 | 0 | 0 | $\delta yaw_{max}$ | -$\delta yaw_{max}$ |

Table 6.3: Orientation input values for main axes movements.

| $input(movement)$ | Down-Left | Down-Right | Up-Left | Up-Right |
|---|---|---|---|---|
| $\delta roll(k)[^\circ]$ | 0 | 0 | 0 | 0 |
| $\delta pitch(k)[^\circ]$ | -$\delta pitch_{max}$ | -$\delta pitch_{max}$ | $\delta pitch_{max}$ | $\delta pitch_{max}$ |
| $\delta yaw(k)[^\circ]$ | $\delta yaw_{max}$ | -$\delta yaw_{max}$ | $\delta yaw_{max}$ | -$\delta yaw_{max}$ |

Table 6.4: Orientation input values for diagonal axes movements.

*Note.* The *Trajectory* is calculated same as in (eq. 6.13). The *State Projection* is given as in (eq. 6.15).

Then the *Movement Automaton* for *Segment* $\in$ *State Space* is defined.

**Definition 22.** *Segmented Movement Automaton For system with segmented state space (eq. 6.21) there is for each state(k) in StateSpace injection function:*

$$ActiveMovementAutomaton : StateSpace \rightarrow MovementAutomaton \tag{6.23}$$

*Selecting appropriate* movement automaton *implementation (def. 21) for* state(k) $\in$ Segment $\subset$ State Space. *The mapping function (eq. 6.23) is injection mapping every state(k) to Segment then* Movement Automaton Implementation. *The trajectory generated is then given:*

$$Trajectory \begin{pmatrix} state(0), \\ Buffer \end{pmatrix} = \left\{ \begin{array}{c} state(0) + \dots \\ \sum_{j=0}^{i-1} ActiveMovementAutomaton(state(j-1)). \\ .input(movement(j)) \\ i \in \{1 \dots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{array} : \right\} \tag{6.24}$$

### 6.2.4 (R) Reference Trajectory Generator

**Reference Trajectory Generator:** Segmented Movement Automaton (def. 22) with *trajectory function* (eq. 6.24) is used as *reference trajectory generator* for *complex systems*.

There is assumption that precise *path tracking* implementation exist for such system which with *thick reference trajectory* gives similar results to *plain movement automaton control*.

The *Reference trajectory* (eq. 6.25) for *Planned* movement set is given as projection of *Trajectory* time series to position time series $[x, y, z, t]$:

$$ReferenceTrajectory : Trajectory \begin{pmatrix} state(now), \\ Planned \end{pmatrix} \rightarrow \begin{bmatrix} x_{ref} \in \mathbb{R}^{|Planned|} \\ y_{ref} \in \mathbb{R}^{|Planned|} \\ z_{ref} \in \mathbb{R}^{|Planned|} \\ t_{ref} \in \mathbb{R}^{|Planned|} \end{bmatrix} \tag{6.25}$$

**Predictor:** The *Reference Trajectory Generator* (eq. 6.25) can be also used as predictor.

*Note.* The *Segmented Movement Automaton* (def. 22) is used in this work with one Segment equal to State space with input function given by (6.1, 6.2). The predictor used in *Reach set computation* is given by (eq. 6.25).

## 6.3 (R) Space Discretization - Avoidance Grid

**Operation Space:** The *Operation Space* is a space where UAS can effectively surveillance its surroundings.

The *Discrete Situation Evaluation* is bounded to *UAS* specific *position* and *orientation* in fixed *time* $t_i$. To enable *deterministic evaluation* the *operation space* needs to be segmented into *finite set* of space portions. The *finite operation space segmentation* is usually done by *Grid* segmentation, which distributes space into portions with solid boundary.

The *Main Sensor* is *LiDAR* (problems 4.23.-5.). The *effective occupancy computation* [127] is given by clustering *LiDAR* field of vision into *polar coordinates grid*.

The *point* scanned by *LiDAR*, where *UAS position* is center of *local coordinate frame* and *UAS heading is defining the main axes* is given as:

$$point = [distance, horizontal^\circ, vertical^\circ].$$

*Note.* For polar/euclidean transformations and local/global coordinate frames refer to background theory (sec. 3.5).

The *right side* of UAS $horizontal^\circ\ ]-\pi, 0[$, the *left side* of UAS $horizontal^\circ \in [0, \pi]$, the *down side* of UAS $vertical^\circ\ ]-\pi, 0[$, the *top side* of UAS $vertical^\circ \in [0, \pi]$

**LiDAR Reading Space Segmentation:** The *polar space* can be separated into cells, which bounds the portion the space, similar to *euclidean* space grid. The *reason* for this segmentation is *LiDAR* reading density[1]. The *polar space portions* state can be assessed directly, the *polar* $\rightarrow$ *euclidean* coordinate frame transformation is not time-effective. The *polar space* assessment of *Lidar Data* has minimal complexity and it is cost effective. [128].

**Definition 23.** *Space partition - cell The* cell *is a portion of space in UAS local polar coordinate frame, given by:*

1. Distance Range - *bounded by* $distance_{start} < distance_{end}$ *in* $\mathbb{R}^+$.
2. Horizontal Range - *bounded by* $horizontal^\circ_{start} < horizontal^\circ_{start} \in ]-\pi, \pi]$.
3. Vertical Range - *bounded by* $vertical^\circ_{start} < vertical^\circ_{start} \in ]-\pi, \pi]$.

*The* bounded space *for* cell *is defined as:*

$$BoundedSpace(cell) = \ldots$$
$$\left\{ point \in \mathbb{R}^3 \ where: \begin{pmatrix} cell.distance_{start} < & point.distance \leq & cell.distance_{end}, \\ cell.horizontal^\circ_{start} < & point.horizontal^\circ \leq & cell.horizontal^\circ_{end}, \\ cell.vertical^\circ_{start} < & point.vertical^\circ \leq & cell.vertical^\circ_{end} \end{pmatrix} \right\} \quad (6.26)$$

*For one* LiDAR Scan *the* hits set *is given as* set *of* all points *which lands in bounded cell space:*

$$LidarHits(cell) = \{point \in LidarScan : point \in BoundedSpace(cell)\} \quad (6.27)$$

[1]Example rotary LiDAR Velodyne VL-16 specs: `https://www.cadden.fr/wp-content/uploads/2017/02/Velodyne_VLP-16-Puck.pdf`

*The* passing hits *for cell are hits which are going trough the cell (passing), but it lands in distance greater than* $cell.distance_{end}$*, defined as:*

$$PassingHits(cell) = \ldots$$
$$\left\{ point \in LidarScan \ where: \left( \begin{array}{rcl} cell.distance_{end} < & point.distance & \\ cell.horizontal^{\circ}_{start} < & point.horizontal^{\circ} \leq & cell.horizontal^{\circ}_{end}, \\ cell.vertical^{\circ}_{start} < & point.vertical^{\circ} \leq & cell.vertical^{\circ}_{end} \end{array} \right) \right\} \quad (6.28)$$

*Note.* The *cells* with same *distance* range form *layers*. The *greater* the distance from *coordinate frame* origin the *greater* volume of the cell.

**Effective Operation Space - Avoidance Grid:** Let start with example, the UAS (fig. 6.3).

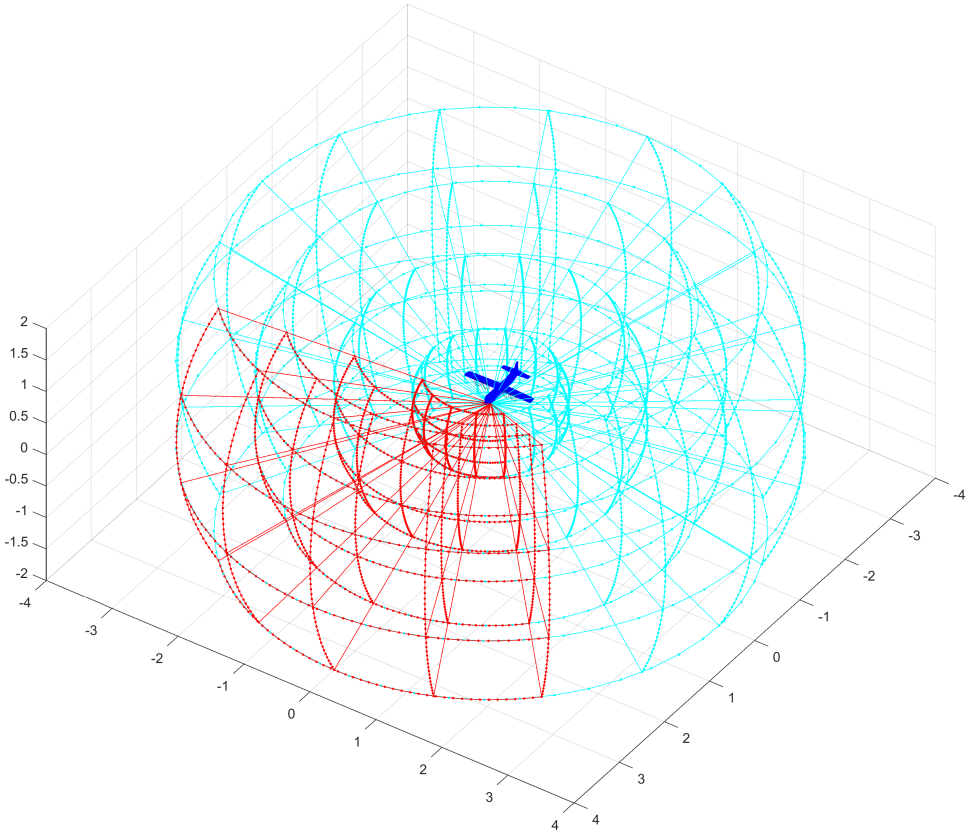



Figure 6.3: Example: The *LiDAR* reading segmentation - cells.

The *full LiDAR Swipe* (cyan and red lines) of *UAS* (blue plane) has *shape* of conical cylinder. Under *ideal circumstances* the *LiDAR swipe* would have *ball shape*, but in real cases the *craft body portion* where *LiDAR* is mounted is unused.

The *frontal portion* (red line) is a set of cells where *UAS* can make maneuver. The *red portion* size is determined by [129]:

1. *Sensors ranges* - the union of *effective sensor ranges* defines the maximal *effective space boundary*, because there is no reason to asses situation over effective *sensor range*.

2. *Information sources* impact - there is no real impact on *effective space boundary*.

3. *UAS maneuverability* - the *Reach Set* (sec. 3.2) gives optimal *effective space boundary*, because there is no need to assess the situation out of *reachable space* or its vicinity.

4. Computation power - the *Reach Set Evaluation* and *Intersection* algorithms are *scaling* with *effective space boundary*.

5. Airworthiness requirements - the *regulations* can impose some minimal requirements on *effective space boundary*.

**Definition 24.** *Avoidance Grid The* effective space boundary *(fig. 6.3 red lines) given by a portion of space in UAS local polar coordinate frame, bounded by:*

*1.* Distance Range - *bounded by* $distance_{start} < distance_{end}$ *in* $\mathbb{R}^+$.

*2.* Horizontal Range - *bounded by* $horizontal^{\circ}_{start} < horizontal^{\circ}_{start} \in ]-\pi, \pi]$.

*3.* Vertical Range - *bounded by* $vertical^{\circ}_{start} < vertical^{\circ}_{start} \in ]-\pi, \pi]$.

*Separated into* layers *depending on the distance and* layer count*:*

$$\begin{aligned} layer^{i}_{start} &= (i-1) \times \frac{distance_{end} - distance_{start}}{layerCount} \\ layer^{i}_{end} &= i \times \frac{distance_{end} - distance_{start}}{layerCount} \end{aligned} \quad ; \quad i \in 1 \dots I \tag{6.29}$$

*Layer horizontal/vertical separations separations defined by horizontal/vertically cell count:*

$$\begin{aligned} horizontal^{j}_{start} &= (j-1) \times \frac{horizontal^{\circ}_{end} - horizontal^{\circ}_{start}}{horizontalCount} \\ horizontal^{j}_{end} &= j \times \frac{horizontal^{\circ}_{end} - horizontal^{\circ}_{start}}{horizontalCount} \end{aligned} \quad ; \quad j \in 1 \dots J \tag{6.30}$$

$$\begin{aligned} vertical^{k}_{start} &= (k-1) \times \frac{vertical^{\circ}_{end} - vertical^{\circ}_{start}}{verticalCount} \\ vertical^{k}_{end} &= k \times \frac{vertical^{\circ}_{end} - vertical^{\circ}_{start}}{verticalCount} \end{aligned} \quad ; \quad k \in 1 \dots K \tag{6.31}$$

*Then* $cell_{i,j,k}$ *given by (def. 23) is member cell of Avoidance Grid for boundaries:*

1. Cell Distance Range *(eq. 6.29) depending on layer index* $i$.
2. Cell Horizontal Range *(eq. 6.30) depending on horizontal index* $j$.
3. Cell Vertical Range *(eq. 6.31) depending on horizontal index* $k$.

*The example of* Avoidance Grid Cells *is given in (fig. 6.3 red boundary).*

*The* Avoidance Grid *is then given as set of cells:*

$$AvoidanceGrid = \bigcup cell_{i,j,k} \forall i \in 1 \dots I, j \in 1 \dots J, k \in 1 \dots K \tag{6.32}$$

**Trajectory Intersection:** The *trajectory* intersection with *Avoidance Grid* is solved in context of *Reach Set Approximation* (def. 26).

*Note.* The *trajectory intersection* function does not have an impact on *Reach Set Approximation*, because its done prior the flight.

**Grid Scaling:** For *Sensor Field* there is *effective sensor boundary* given as set:

$$Boundary(Sensor \in SensorField) = \{points \in \mathbb{R}^3 : \text{where reliable}\} \tag{6.33}$$

The *Boundary* for sensor fields is then given as *union of all singe sensor boundaries*:

$$Boundary(SensorField) = \bigcap_{\forall Sensors} Boundary(Sensor \in SensorField) \tag{6.34}$$

Depending on boundary properties it can be projected into maximal avoidance grid boundary values:

$$Boundary(SensorField) \rightarrow AvoidanceGrid : \begin{matrix} \max(distanceRange) \\ \max(horizontalRange) \\ \max(verticalRange) \end{matrix} \tag{6.35}$$

Our approach taken worst LiDAR performance into account [24] and following parameters for avoidance grid were calculated:

1. distance range $[0m, 10m]$,
2. horizontal range $]-180°, 180°]$,
3. vertical range $[-30°, 30°]$.

The *count of layers* is derived from *average distance traveled by one movement application*:

$$layerCount = \frac{|distanceRange|}{\text{avg.} \quad length(movement \in MovementSet)} \tag{6.36}$$

The *layer length* is based on *our movement set* (tab. 6.1, 6.2) the average movement length is 1 m, therefore the *layer count* is 10.

The *efficient boundary* is given by *Reach Set*. Estimate reach set coverage space using *ellipsoidal toolbox* [130] up to given *sensor field* maximal distance:

$$Boundary(ReachSet) = Ellipsoid(UASSystem, distance) \tag{6.37}$$

The values for *Reach Set Boundary* with distance 10 m was following:

1. distance range $[0m, 10m]$,
2. horizontal range $[-45°, 45°]$,
3. vertical range $[-45°, 45°]$,

The *Avoidance Grid* boundary is given as *intersection* of all boundaries:

$$Boundary(AvoidanceGrid) = Boundary(ReachSet) \cap Boundary(SensorField) \quad (6.38)$$

The values for *Avoidance Grid Boundary* for our UAS system (sec. 6.2.1) following:

1. distance range $[0m, 10m]$,
2. horizontal range $[-45°, 45°]$,
3. vertical range $[-45°, 45°]$,
4. layer count 10, layer distance 1m.

The *horizontal cell count* and *vertical cell count* was estimated by *rule of thumb* to have value 7 and 5.

**Cell in Avoidance Grid Properties:** For each cell $\vec{p} \in \mathbb{R}^3$ in the there are properties to be checked:

1. *Is there visibility to the cell ?* - how good is an observation of the cell by Sensor Field.
2. *Is there threat present ?* - how sure the data fusion is that there is eminent threat in the cell.
3. *Is the cell reachable ?* - if there is any trajectory which can get UAS to that cell without too much threat along the way.

The answers to these questions will be given later (tab. 6.5).

## 6.4 (R) Reach Set Approximation

**Motivation:** *Reach set* is strong tool for *Obstacle Avoidance* because it contains all possible *avoidance maneuvers* in set. The current implementation have following flaws:

1. *Realistic approximation* - *nonlinear systems* or *heavily constrained systems* can not be approximated well by *continuous-time Reach Sets*.
2. *Non Deterministic calculations* - continuous-time *Reach Set* contains infinite possibilities for *avoidance maneuvers*, the SAA system demands conflict resolution in finite time.
3. *Property binding* - binding related properties seems problematic, because *continuous- time reach sets* does not have unique identifier of maneuver, trajectory nor segment.

**Proposed Solution Features:** Our Reach set Estimation method will provide following features:

1. *System Control Interface* - implemented via *Movement Automaton*, requiring only *discrete command chain* to approximate system behaviour.
2. *Deterministic Calculation* - finite number of elements in *Reach set* will enable *scalable* calculation.
3. *Property binding* - approximation of Reach set as a set of trajectories, each trajectory can be split into finite number of segments. Each element will have unique identifier enabling both-side property binding.
4. *Behaviour encoding* - some specific behaviour, like horizontal/vertical separation, or maneuver shape can be encoded into *Reach Set*.

**Discretization of Reach set:** There is a need for a discrete finite *Reach Set approximation* to enable *Avoidance Strategy Evaluation* in finite time. Replacing *Continuous Control Set Inputs(t)* by *Movement Automaton* is feasible:

**Definition 25** (Reach set Approximation by Movement Automaton)**.** A trajectory *(def. 10) for system* $\dot{state} = f(time, state, input)$ *under control of the movement automaton* $\mathcal{MA}$ *is given as execution of movement buffer (def. 9) with initial state of system* $state_0$*. Therefore notation* $Trajectory(state_0, buffer)$ *is used.*

**The Complete Reach Set** *(6.39) for system with initial state* $state_0$ *with existing* control strategy $control(time) \in Controls(time)$*. for time* $\tau > time_0$*.*

$$ReachSet(\tau, time_0, state_0) = \bigcup \{state(s) : control(s) \in Controls(s), s \in (time_0, \tau]\} \tag{6.39}$$

**The Reach Set Approximation by Movement Automaton** *(6.40) of the system under the control of the movement automation* $\mathcal{MA}$ *consist from the set of trajectories* $Trajectory$ ($state_0$, $Buffer$)*, which are executed in constrained time* $\tau > time_0$*.*

$$ReachSet(\tau, time_0, state_0) = \left\{ Trajectory(state_0, buffer) : \begin{matrix} duration(buffer) \\ \leq \\ (time_0 - \tau) \end{matrix} \right\} \quad (6.40)$$

*Note. Reach Set Approximation* (def. 25) is subset of *Full Reach Set* (def. 2) in continuous space $\mathbb{R}^n$ it inherits all important properties, like *Invariance* [63].

*Discretization* of *Reach Set* have been achieved leaving us with *finite count* of *Trajectories*, instead of *Infinite subspace or* $\mathbb{R}^N$

**Approximated Reach Set Containment:** The *Approximated Reach Set* introduced in (def. 25) is constrained only by *future expansion time* $\tau$. UAS makes space assessment in *Avoidance Grid*. There is no point to consider Trajectories outside of *Avoidance Grid*

**Definition 26** (Contained Aproximated Reach Set)**.** *For pair (*$state_0$*,* $AvoidanceGrid_0$*) at time* $time_0$ *and* prediction horizon $\tau = \infty$ *there is* Contained Reduced Reach Set:

$$ReachSet \begin{pmatrix} time_0, \\ state_0, \\ AvoidanceGrid_0 \end{pmatrix} = \left\{ \begin{matrix} Trajectory(\dots) \\ \in \\ ReachSet(6.40) \end{matrix} : \begin{matrix} \forall segment \in AvoidanceGrid_0, \\ segment \in Trajectory(\dots) \end{matrix} \right\} \quad (6.41)$$

**Properties:** Container Aproximated Reach Set *contains only trajectories where all segments belongs to* Avoidance Grid*, there are following functions:*

1. Membership function *for any* Trajectory *in* Constrained Reduced Reach set *returns* Ordered Set *of* Passing Cells.

2. Cost function *for any* Trajectory Portion *in* Constrained Reduced Reach Set *return* Cost of Execution

**Passing cell:** Cell *of* Avoidance Grid *which has some intersection with Trajectory.*

*Note. Contained Reduced Reach Set* (eq. 6.41) which is contained in *Avoidance Grid* and have an *Membership Function* enable Property transition between Reach set and *Avoidance grid*.

*Example:* Visibility from cells along *Trajectory* can be gathered to calculate *Trajectory's* feasibility.

**Reach Set Pruning:** There is a need to implement *Set Difference* between *Reach Set* and *Constraint Set*. Constraint Set cam be *Obstacle Set* from *Known World* (sec. 4.1.10) and other different constraints.

**Reach Set Trajectory Tree:** (6.42) *Any Reach Set* where *Control Strategy Constraint* is implemented as *Movement Automaton*, with defined *Movements* set and for singe initial $state_0$. The *Reach Set* is given as discrete tree with root $Trajectory(state_0, \varnothing)$.

$$ReachSet(state_0, \dots) = \left\{ Trajectory(state_0, buffer) : \begin{matrix} buffer \in Movements^i, \\ i \in \{1, \dots, k\} \end{matrix} \right\} \tag{6.42}$$

For each *Trajectory Segment*, there exists *intersection function* which evaluates as true if there exists at least one point in *Segment* which belongs to *Constraint Set*. Formally:

$$intersection(segment, Set) : \begin{cases} \exists point \in segment, \\ \ point \in Set & : true \\ Otherwise & : false \end{cases} \tag{6.43}$$

**Definition 27** (Pruned Reach Set)**.** *For* Reach set *represented as* Trajectory Tree *(eq. 6.42) and some constraint set (Set) where exist* intersection function *(eq. 6.43). The* Pruned Reach set *is given as follows:*

$$Prune(ReachSet, Set) = \left\{ Trajectory(\dots) : \begin{matrix} \forall segment \in Trajectory, \\ \neg intersection(segment, Set) \end{matrix} \right\} \tag{6.44}$$

*Note.* Pruning(def. 27) [131] is applicable multiple times for various *Constraints Set*.

Example of *Approximated Reach set Calculation* (def. 25), *Reach Set Containment* (def. 26), and, *Pruning* is given in [7].

### 6.4.1 (R) Reach Set Performance Criteria

**Motivation:** The need to Make *Reach Set* scalable approach. This may be a problem due the *Expansion rate*. *Reach set* represented as a *Trajectory Tree* (eq. 6.42) for Avoidance Grid with *layerCount* and Movement automaton with *movementCount*, the *Node count* is given as:

$$1 + \left( \sum_{i \in \{1 \dots layerCount\}} (movementCount)^i \right) \tag{6.45}$$

*This scaling* is not feasible for *Avoidance Grid* with many layers ($< 10$) or *Movement Set* with many movements ($< 9$). There is need for *Reduced Reach set calculation*.

**Core Performance Criteria:** The scaling factor (eq. 6.45) shows that there are going to be many trajectories. The main point is that not every trajectory in *Reach Set* are giving us *maneuverability advantage*. Our expectations lies in following *Performance Requirements*:

1. *Reach set* must *Cover* maximum of the *possible unique maneuvers* in *Avoidance Grid*.
2. *Trajectories* in *Reach Set* should be smoothest possible to prevent cargo damage / UAS wear.

**Trajectory footprint:** Discrete space of *Avoidance Grid* is organized in cells. *Cell* is minimal space portion accessible by *property binding*. There is need to know if two trajectories contribution to *Maneuverability* in this environment.

Each trajectory passes trough space in *Avoidance Grid*. If there exists a method to extract unique identifier for each *trajectory passed cells*, we can compare two trajectories *Coverage* in *Avoidance Grid*.

**Definition 28** (Trajectory footprint)**.** *For* Trajectory *from* Reach set *(def. 26) defined for* Avoidance Grid *has membership function.* Membership Function *returns* ordered set of passing cells*:*

$$footprint\begin{pmatrix} Trajectory, \\ AvoidanceGrid \end{pmatrix} = \left\{ \begin{matrix} cell \in AvoidanceGrid : \\ isMember(trajectory, cell) \end{matrix} \right\} \tag{6.46}$$

*Then we can define equality function for* $Trajectory_1$ *and* $Trajectory_2$*, as comparison of their footprints in common* Avoidance Grid *as follow:*

$$isEqual\begin{pmatrix} Trajectory_1, \\ Trajectory_2, \\ AvoidanceGrid \end{pmatrix} : \begin{cases} \begin{pmatrix} footprint(Trajectory_1, \dots) \\ = \\ footprint(Trajectory_2, \dots) \end{pmatrix} & : true \\ Otherwise & : false \end{cases} \tag{6.47}$$

*Note.* Depending on *Movement Automaton's* movement set and *Avoidance Grid* parameters, there can be multiple *trajectories* which are equal.

**Coverage set:** Now it is possible to create set of unique *trajectory footprints* due to *footprint function* (eq. 6.46). Similarly there is a possibility to create *Reach set skeleton* containing unique trajectories, by using *equality function* (eq. 6.47). *Coverage set* is sufficient for now.

**Definition 29** (Coverage Set)**.** Coverage set *(6.48) is defined for* Avoidance Grid *and* Reach Set *pair as set of unique* Trajectory footprints*:*

$$CoverageSet\begin{pmatrix} AvoidanceGrid, \\ ReachSet \end{pmatrix} = \left\{ footprint\begin{pmatrix} Trajectory, \\ AvoidanceGrid \end{pmatrix} : \begin{matrix} \forall Trajectory \\ \in ReachSet \end{matrix} \right\} \tag{6.48}$$

**Coverage set properties:** Trajectory footprint (eq. 6.46)is not *bijection*, neither *injection* for $ReachSet \rightarrow CoverageSet$. This implies following properties:

1. Equal *Reach Sets* in same *Avoidance Grid* have equal *Coverage Sets*.
2. Equal *Coverage Sets* does not imply *Reach Set* equality.
3. For two Coverage Sets there is a possibility to compare their member count to create coverage ratio.

The second *Property* gives us a preposition that there is a possibility of *Reach Set Reduction* without loosing *Coverage*.

**Definition 30** (Coverage Ratio)**.** Coverage Ratio *is a ratio of* Coverage Set Member Count *between two* Reach Sets. *Reach set with* lesser count of unique Trajectories *is considered as* Reduced Reach Set. *Reach set with* greater Count of unique Trajectories *is considered as* Reference Reach Set.

$$\begin{aligned} referenceCoverage &= |CoverageSet(ReferenceReachSet, AvoidanceGrid)| \\ reducedCoverage &= |CoverageSet(ReducedReachSet, AvoidanceGrid)| \\ CoverageRatio &= \frac{reducedCoverage}{referenceCoverage} \in [0,1] \end{aligned} \tag{6.49}$$

*Note. Reference Reach Set* is usually *Full Reach Set* containing all possible trajectories in space contained by *Avoidance Grid*.

In case *Full Reach Set* can not be computed, Avoidance Grid is too large, most complex *Reach Set* is used as *Reference Reach Set*.

**Trajectory smoothness:** Trajectory other than straight line have some changes in $UAS$ heading.

The goal is to minimize *Maneuvering* of UAS, because:

1. *Every Heading Change* needs to be reported to *UTM*.
2. *Sharp Maneuvering* can damage cargo/wear UAS.
3. *Often course changes* makes *Intruder prediction* harder for other Civil General Aviation.

For this purpose *Smoothness Metric* needs to be applied for *Reach Set* or *Trajectory*. In case of *Movement Automaton Control* two distinguish *Movement Sets* can be introduced:*Smooth* nad *Chaotic* movements set with following properties:

$$\begin{aligned} MovementSet &= SmoothMovements \cup ChaoticMovements \\ SmoothMovements &\cap ChaoticMovements = \varnothing \\ |SmoothMovements| &> 0, \quad |ChaoticMovements| > 0 \end{aligned} \tag{6.50}$$

Then *Smoothnes clasificator* for $Trajectory(initialState, buffer)$ can be defined as $isSmooth$ and *Smooth Movement Counter* function as $smoothCount$ like follow:

$$\begin{aligned} isSmooth(movement) &= \begin{cases} movement \in SmoothMovements & : 1 \\ movement \in ChaoticMovements & : 0 \end{cases} \\ smoothCount(Trajectory(\ldots, buffer)) &= \sum isSmooth(movement), \\ &\qquad \forall movement \in Buffer \end{aligned} \tag{6.51}$$

**Definition 31** (Smoothness Rating for Trajectory)**.** Smoothness *for trajectory generated by* Movement Automaton *for some* Initial State *with some* Movement Buffer*, under*

*assumption of* Smooth and Chaotic Movement Set *split (eq. 6.50), with existing* classification *and* counter *functionals (eq. 6.51) is given as follows:*

$$Smoothness(Trajectory(\ldots, buffer)) = \frac{isSmooth(Trajectory)}{movementCount(Trajectory)} \in [0,1] \quad (6.52)$$

*For* Trajectory *with* $buffer = \varnothing$ Smoothness *is given as 1.*

### 6.4.2 (R) Constrained Trajectory Expansion

**Motivation:** *Purpose* of *Navigation* is to move forward to *Goal Waypoint* in *Mission*. *Structure* of *Avoidance Grid* is designed to enable *forward* and *turning* maneuvers. The *Avoidance Grid* is organized in *Layers* characteristic by same distance from *Avoidance Grid Origin*.

Survey of motion planning algorithm was given in [132]. The ideal candidate for propagation algorithm is *Wave-front* algorithm propagating *Trajectory tree* trough Layers. Due the *Avoidance Grid* onion like layers, there is possibility to implement turn maneuver trough layers iterative and effectively .

**Rapid Exploration Tree** (fig. 6.4) was selected, because it enables *Movement Automaton Utilization* and *Property Binding*. Similar approach was used in rapid exploration tree for space exploration [133].

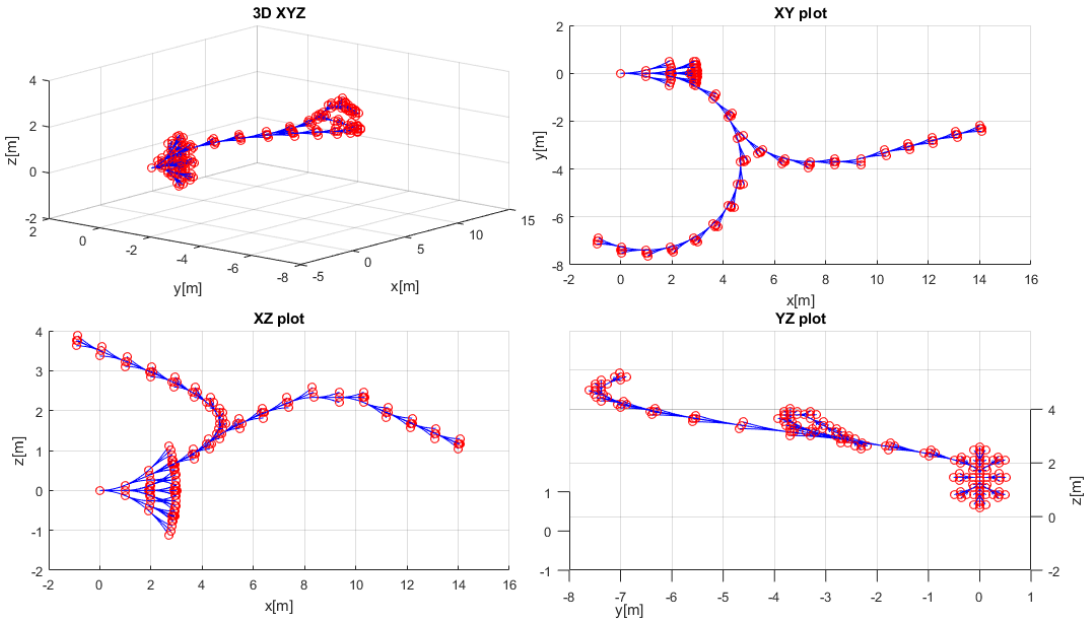



Figure 6.4: *Rapid Exploration tree as result of* Constrained trajectory expansion.

**The example** (fig. 6.4) shows a *Rapid Exploration Tree* in *Free Space* containing *Waypoint Navigation Path* and *Turn Away Path*. Both paths are starting in same *Root Node* (red circle) which was expanded with simple *Movement Automaton* (bunch of nodes originating from one node are showing way of expansion). The connection (blue line) between two nodes (red circles) represents *Trajectory portion* for *Executed Movement*.

**Rapid Exploration Tree Node** will contain following information:

1. *Initial state* - root entry point, used in state evolution calculation.
2. *Trajectory (state evolution)* - trajectory passing trough *state space* in local coordinate frame of *Avoidance Grid*.
3. *Buffer* (applied movements) - ordered list of *executed movements* applied on *initial state* to obtain *state evolution*.
4. *Cost* - calculated for *state evolution* based on *predefined cost function*.
5. *Footprint* - ordered set of *passing cells* in *Avoidance Grid*.
6. *Parent Node Reference* - tree reference for parent node, not in case of *root node*.
7. *Other Bounded Properties* - value list of other properties, depending on *Expansion Constraints* and *Reachibility* evaluation algorithm.

**Wave-front propagation of Rapid Exploration Tree** is given in (alg. 6.1).

The *Avoidance Grid* have UAS with *position* $\in$ *Initial State* at the *origin*. The *Grid Layer* is a column ordered set of cells with same *Mean distance* from origin. *Grid Layers* are indexed from origin starting with 1, there is maximum of $i \geq 1$ layers.

**Step: Initialization** contains base structure preparation like follows:

1. *Avoidance Grid* - Space containing *Reach set* (def. 26).
2. *Movement Automaton* - Used as *Predictor*, consuming *buffer* containing *Movements* to generate $Trajectory(initialState, buffer)$.
3. *Reach Set* - tree consisting from *Wave-frontNodes* representing the end point of $Trajectory(initialState, buffer)$ where each *Edge* represents *one Movement application*. The root is set as node containing *Initial State*.

Function *initializeReachSet(root,stack,grid,automaton)* will take the root and enforces *full wavefornt propagation* to *First Layer*.

**Step: Wave-front Propagation** is forced propagation of trajectories from layer $i$ to layer $i + 1$. The process goes as follows:

1. *Selection of Feasible candidates* - function *[candidates,leftovers] = ExpansionConstraints.select(stack)* for working layer, row and cell selects *feasible trajectory nodes* ordered by *Cost function*. The *Example of Cost Function* can be *Trajectory Smoothness* (def. 31).
2. *Expansion of Candidates* - for each *candidate* function *candidate.expandNode (automaton)* is invoked. This function will expand *Candidate Node structure* by appending *Full Trajectory Tree Evolution* until each *Leaf Trajectory* reaches *Next Layer*. Simply put *Par rent Node Node(initialState, buffer, cost, footprint )* buffer is appended by movements until the next layer is reached.

3. *Leftovers purge* - function *reachSet.purge(leftovers)* removes unexpanded *Nodes* leading to cell, effectively removing trajectories which does not lead to *next layer*.

4. *Append Reach Set* - function *reachSet.append(leafs)* puts newly created *Nodes (Trees)* into *Reach Set* structure. The *Wave-front Propagation* for one cell is finished.

**Step: After Layer Propagation Purge** is covered by function *reachSet.purgeSameFootprint()* which takes trajectories with same footprint and keeps some of them based on *Selection criteria*, more in (sec. 6.4.3, 6.4.4). *Pruning methods* over *Large Decision Trees* are *fast* and *viable* [134].

**Algorithm 6.1:** *Wave-front propagation* of *Rapid Exploration Tree* to form *Reach Set*.

```
Input  : Node(initialState,buffer=∅,cost=0,footprint=∅), AvoidanceGrid,
         ExpansionConstraints, MovementAutomaton(movementSet)
Output: ReachSet(AvoidanceGrid)

# Initialization Sequence;
grid=AvoidanceGrid, automaton=MovementAutomaton, root = Node;
reachSet = initializeReachSet(root,stack,grid,automaton);

# Main Expansion trough, layers (i), rows (j), cells(k);
for layer(1...i) in grid do
    for row(1...j in layer) do
        for cell(1...k) in row do

            # apply selection criteria ;
            [candidates,leftovers] = ExpansionConstraints.select(stack);

            # collect expansions ;
            leafs = [];
            for candidate in Candidates do
             |  leafs= [leafs, candidate.expandNode(automaton)];
            end
            reachSet.purge(leftovers);
            reachSet.append(leafs);
        end
    end
    reachSet.purgeSameFootprint();
end
```

*Note. Reach Set* is usually computed *Prior the Flight* for *some Initial State* in *Local Coordinate Frame* in *right had coordinate frame* with $X^+$ used as *main axis*.

### 6.4.3 (R) Chaotic Reach set

**Motivation:** Design of calculation method for *Reach Set Approximation* guarantying high *Maneuverability*.

**Background:** There is *Coverage Ratio* property of *Reach Set* (def. 30). It has been shown that creating *Reach Set* via *greedy approach* is not feasible due the *Scaling Factor*.

*Contracted Expansion* (sec. 6.4.2) is enabling to apply selection criteria while building *Reach Set* in given *Cell*.

The *Cell* $cell_{i,j,k}$ has a center and walls from UAS viewpoint: front wall , back wall (for $layer > 1$), top wall, left wall, right wall, bottom wall. It is expected that trajectory leading close to one cell walls will continue to different cell, increasing chance to obtain more *Unique Footprints*.

**Expansion Constraint Function Implementation** (alg. 6.2) is based on simple principle: *Select candidate Nodes which are closest to outer walls of Cell, with unique footprint*.

**Tuning Parameters** : *Proximity to Cell outer wall* gives good chances to break into other rows or columns in *Avoidance Grid*. *Unique footprint* guarantees future *Unique Footprint* after appending Trajectory by *Movement application*.

1. *Considered Footprint Length* - how much last cells in footprint should be considered in unique path track, minimal value 1, default value 3, maximal value $\infty$. If you want to generate non redundant trajectories use $\infty$, it will consider full footprint.

2. *Spread Limit* - upper limit of candidates which are going to be select for further expansion, minimal value 1, default value *Count of unique Moves in Movement set*, maximal value $\infty$. If more than default values is selected the algorithm will generate *redundant trajectories*. If less is selected then some trajectories are omitted and *Coverage Rate* decreases sharply.

**Step: Initialization** initialization of *candidate* array (return value), *leftovers* array (return Value). Node array *passing* is populated with *Nodes* which represents *end node of Trajectory* and the tip of *trajectory is constrained in* $\text{cell}_{i,j,k}$.

**Step: Evaluate best trajectories with unique Footprints** following steps are executed:

1. *Best Performance Map* is created with *footprint* as key set element to ensure footprint uniqueness.

2. *Wall distance* for *test node* is calculated as a closest trajectory portion distance to *top, bottom, left, right* wall of cell $cell_{i,j,k}$

3. *Footprint* for *test node* is created with maximal length given by *Footprint Length* tuning parameter.

4. *Existence and Performance Test* is executed to ensure that best performing node is selected. If there is not key entry in the *Best Performance Map*, then new entry for *Test Node* is created. If there is key entry, the performance of *Old Node* and *Test Node* is compared and better is stored.

**Step: Select candidates** is executed on *Best Performance Map* records using *Wall distance* as pivot parameter, ordering by closest proximity and limited by *Search Limit* tuning parameter. The *Leftovers* are difference set between *Passing Nodes* and *Candidate Nodes*.

**Algorithm 6.2:** Expansion Constraint function for *Chaotic Reach Set Approximation*

**Input** : Node[] stack, Cell $\text{cell}_{i,j,k}$
**Tuning Parameters:** $\text{int}^+$ footprintLength, $\text{int}^+$ spreadLimit
**Output** : Node[] candidates, Node[] leftovers

```
# Initialize structures;
Node[] candidates = [], Node[] leftovers=[];
Node[] passing = cell_{i,j,k}.getFinishingTrajectories(stack);

# Select best performing trajectories with unique footprint;
Map<Footprint,Node> bestPerformanceMap;
for Node test ∈ passing do
    wallDistance= test.minimalDistanceToWall(cell_{i,j,k})];
    footPrint = test.getFootprint(lastCells = footprintLength);
    if bestPerformanceMap.contains(footPrint) then
        old = bestPerformanceMap.getByKey(footprint);
        oldPerformance= old.minimalDistanceToWall(cell_{i,j,k});
        if oldPerformance > wallDistance then
            bestPerformanceMap.setByKey(footprint,test);
        end
    else
        bestPerformanceMap.setByKey(footprint,test);
    end
end

# Select best performing nodes up to spreadLimit count;
candidates = bestPerformanceMap.select(count =
 spreadLimit).orderBy('wallDistance','Ascending');
leftovers = passing - candidates;
return [candidates,leftovers]
```

**Example:** for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45°, +45°]$, *Horizontal Cell Count 7*, *Vertical range* $[-30°, +30°]$, and *Vertical Cell Count 5*. Is given in (fig. 6.5). The UAS is at *Back-side* of *Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid* space boundary. Each trajectory has its own color and ends at *Front-side* of *Avoidance Grid Boundary*.

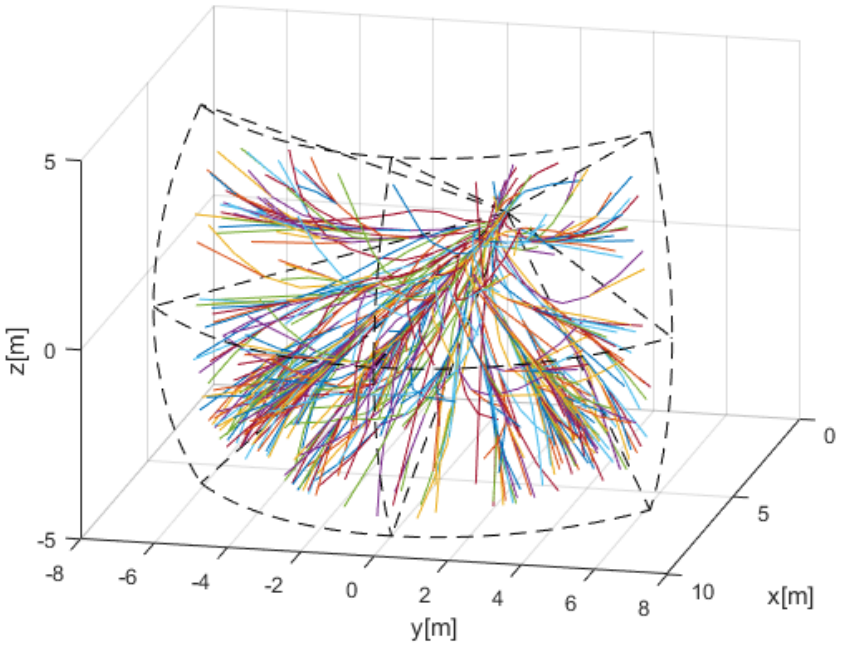



Figure 6.5: *Chaotic* reach set *approximation*.

**Pros and Cons:** It can be seen from example (fig. 6.5) that *Chaotic Reach Set Approximation Method* (alg. 6.2) generates a lot of *turning* and *shaky trajectories*.

*High Coverage Ratio* ($\sim 0.9$) is provided, while keeping *medium node count*. The calculation complexity scales linearly with grid size. The *upper limit of trajectories* is given as follow:

$$countTrajectories(ReachSet) \leq layerCellCount \times spreadLimit \times size(Movements) \quad (6.53)$$

The *upper limit of nodes* is given as follow:

$$countNodes(ReachSet) \leq layerCount \times layerCellCount \times size(Movements) \times spreadLimit \quad (6.54)$$

*Absence* of *Smooth Trajectories* disqualifies *Chaotic Reach Set Approximation* to be used for *Navigation*. This type of reach set is feasible for *Avoidance*, because it contains variety of maneuvers.

### 6.4.4 (R) Harmonic Reach set

**Motivation:** Imagine having an *Avoidance Grid* like (fig. 6.3). There is a need of *Reach Set Approximation* which will have *Smooth Trajectories* (def. 31) going nearby *cell* centers.

**Background:** The *Smoothness Rating for Trajectory* (def. 31) uses two distinct sets *Smooth Movements* and *Chaotic Movements* (eq. 6.50) which are defined for our *Movement Automaton* (sec. 6.2) like follow:

$$
\begin{aligned}
SmoothMovements &= \{Straight\} \\
ChaoticMovements &= Movements - SmoothMovements
\end{aligned}
\tag{6.55}
$$

*Smooth Movements* contains only *Straight* movement, because others are considered as extreme turning movements. *Smooth Movements* should contain only direct flight movements or slight heading correction. *Chaotic Movements* set is supplement of *Movement Automaton's Movement Set*.

The *Avoidance Grid* (fig. 6.3) cell centers for fixed indexes $j_{fix}$, $k_{fix}$ are linearly aligned with *initial state*. That means that cell centers of cells $cell_{1,j_{fix},k_{fix}}, \ldots, cell_{i,j_{fix},k_{fix}}$, where $i$ is count of *layers* lies on one line. If the trajectory can achieve *cell center* on some *layer* only minor trajectory corrections are required to stay on given line. This type of trajectory gives us following advantages:

1. *Minimal steering at beginning* - the minimal steering is advantageous in *Controlled Airspace* because is diminishing the amount of communication to *UTM Service*.

2. *Additional safe space in Linear segment* - once the *center of cell* is reached, *Trajectory* sticks to line between cell centers. Each point on this line has *maximal distance* to outer walls of cell. This gives us extra space given as minimum of distance between *UAS position* and *Outer cell walls*.

**Expansion Constraint Function Implementation** (alg. 6.3) is based on simple principle: *Select candidate Nodes which are closest to Cell center, with unique footprint.*

*Note. Cell center* can be closely reached by *smooth movement* from previous cell or *chaotic movement* from neighbouring cell from current or previous layer. These trajectories are usually equivalent in *Smoothness*.

**Tuning Parameter:** *Proximity to Cell Center* gives a good chance to keep trajectory smooth or *smooth after one correction maneuver*. It has been mentioned that *Cell Center* can be reached by various trajectories. In this method full footprint length is always considered, therefore only one tuning parameter can be offered:

1. *Spread Limit* - upper limit of candidates which are going to be selected for further expansion, minimal value 1, default value *Count of unique Moves in Movement set*, maximal value $\infty$. If maximal value $\infty$ is selected, algorithm will generate skeleton of *Reach Set* with full Coverage and with the smoothest *Trajectories*.

**Step: Initialization** sets candidate *Nodes* as empty set, leftover *Nodes* as empty set. and selects all *Nodes* from *Stack* which represents *Finishing Trajectories* in working cell $cell_{i,j,k}$.

**Step: Evaluate smoothest trajectories with unique Footprints** is implemented as *multi-criteria filtration*.

*First criterion* is *distance to Cell Center* which is penalized by trajectory *smoothness rate* implemented in method *Node.getPerformance(Cell* $cell_{i,j,k}$*)* defined as follow.

$$
getPerformance(Node, Cell) = \frac{distance(Node.Trajectory, Cell.Center)}{SmoothnessRate(Node.Trajectory)} \tag{6.56}
$$

Distance of *Trajectory* is *enumerator*, because its considered as *base value* and is defined in interval $[0, maximalWallDistance]$. The *Smoothness Rate* is in denominator, because it is a penalization coefficient defined in interval $[0, 1]$.

*Second criterion* is *trajectory uniqueness* This is provided by *Best Performance Map*, where best performing *Node* belongs to one unique *trajectory footprint*. The implementation is identical to *chaotic reach set expansion* (alg. 6.2).

**Step: Select candidates** is executed on *Best Performance Map* records using *Penalized Cell Center Distance* as pivot parameter, ordered in ascending order and limited by *Spread Limit* tuning parameter. The *Leftovers* are difference set between *Passing Nodes* and *Candidate Nodes*.

**Algorithm 6.3:** Expansion Constraint function for *Harmonic Reach Set Approximation*

**Input** : Node[] stack, Cell $\text{cell}_{i,j,k}$
**Tuning Parameters:** $\text{int}^+$ spreadLimit
**Output** : Node[] candidates, Node[] leftovers

```
# Initialize structures;
Node[] candidates = [], Node[] leftovers=[];
Node[] passing = cell_{i,j,k}.getFinishingTrajectories(stack);

# Select unique smoothest trajectories;
Map<Buffer,Node> bestPerformanceMap;
for Node test ∈ passing do
    centerDistance= test.getPerformance(cell_{i,j,k})];
    footPrint = test.getFootprint();
    if bestPerformanceMap.contains(footPrint) then
        old = bestPerformanceMap.getByKey(footprint);
        oldPerformance= old.getPerformance(cell_{i,j,k});
        if oldPerformance > centerDistance then
            bestPerformanceMap.setByKey(footprint,test);
        end
    else
        bestPerformanceMap.setByKey(footprint,test);
    end
end

# Select best performing nodes up to spreadLimit count;
candidates = bestPerformanceMap.select(count =
 spreadLimit).orderBy('cellCenterDistance','Ascending');
leftovers = passing - candidates;
return [candidates,leftovers]
```

**Example:** for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45°, +45°]$, *Horizontal Cell Count 7*, *Vertical range* $[-30°, +30°]$, and *Vertical Cell Count 5*. Is given in (fig. 6.6). The UAS is at *Back-side* of *Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid* space boundary. Each trajectory has its own color and ends at *Front-side* of *Avoidance Grid Boundary*. The *Spread Limit* in this case was set to 9 which is *Size of the Movement Set*.

*Note.* Please note *Trajectories* are organized in bundles going around *Cell Centers smoothly.* Most of the steering maneuvers are executed at the *beginning* of *Avoidance Grid.*

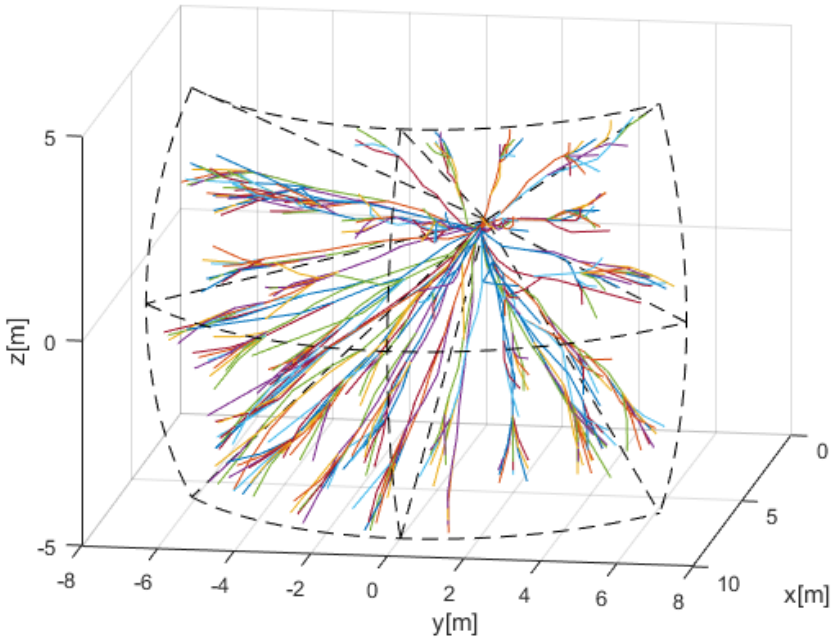



Figure 6.6: *Harmonic* reach set *approximation.*

**Pros and Cons:** It can bee seen from example (fig. 6.6) that *Harmonic Reach Set Approximation Method* (alg. 6.3) generates *smooth evenly spread trajectories.*

High smoothness ratio ($\geq 0.9$) is provided, while keeping low node count for UAS systems. The calculation complexity scales linearly with grid size. The upper limit of trajectories is given as follow:

$$countTrajectories(ReachSet) \leq layerCellCount \times spreadLimit \times size(Movements) \quad (6.57)$$

The *upper limit of nodes* is given as follow:

$$countNodes(ReachSet) \leq layerCount \times layerCellCount \times spreadLimit \quad (6.58)$$

Absence of *High Coverage Ratio* disqualifies *Harmonic Reach Set Approximation* to be used for *Emergency Avoidance.* This type of *Reach Set* is feasible for *Open Space Navigation* or *Controlled Airspace Navigation.* Its low turning rate in contained *Trajectories* are desired for such tasks.

### 6.4.5 (R) Combined Reach set

**Motivation:** Harmonic reach set (sec. 6.4.4) is *efficient* for *Navigation in* Controlled Airspace. Chaotic reach set (sec. 6.4.3) is good for *Emergency avoidance.* The need to differentiation between *Navigation* and *Emergency Avoidance* mode is necessary in

*Controlled Airspace.* but not in *Non-controlled Airspace.* The combination of *Harmonic* and *Chaotic* reach sets is obvious solution.

*Automatic mode switch* can be provided by combination of *Navigation Reach Set* and *Avoidance Reach Set* with elevated cost function. Overall having a method to merge multiple trees would be beneficial.

**Background:** If two *Reach Set Approximation* were calculated for same *Avoidance Grid* and *Initial State*, using same *Movement Automaton* and *UAS model* are possible to merge.

The *Reach Set Approximation* is *tree* with *Root Node* in *initail state* with movement buffer = $\varnothing$. The *movement buffer* in each node can be used as *route trace* during merging procedure. The example two reach set merge can be given as follow, where only *latest* applied movement is taken into account.

$$\left[\begin{array}{l}\texttt{First Reach Set} \\ \quad \varnothing \rightarrow \left\langle \begin{array}{l} left \rightarrow \left\langle \begin{array}{l} left \\ right \end{array} \right. \\ \varnothing \end{array} \right. \\ \texttt{Second Reach Set} \\ \quad \varnothing \rightarrow \left\langle \begin{array}{l} \varnothing \\ right \rightarrow \left\langle \begin{array}{l} left \\ right \end{array} \right. \end{array} \right. \end{array}\right] \rightarrow \left[\begin{array}{l}\texttt{Combined Reach Set} \\ \quad \varnothing \rightarrow \left\langle \begin{array}{ll} left & \rightarrow \left\langle \begin{array}{l} left \\ right \end{array} \right. \\ right & \rightarrow \left\langle \begin{array}{l} left \\ right \end{array} \right. \end{array} \right. \end{array}\right] \tag{6.59}$$

*First Reach Set* contains two trajectories given by buffers {*left,left*} and {*left,right*}. *Second Reach Set* contains two trajectories given by buffers {*right,left*} and {*right,right*}. The *Combined Reach Set* contains all four trajectories.

*Note.* The combined tree [135] does not need to have combined amount of original *Reach Sets* trajectories. There can be *Duplicity* which means that any bounded property like *Cost* must be *calculated* again.

**Combined Reach Set Calculation Function** (alg. 6.4) is implemented as function $NodecombinedReachSet(\dots)$ which takes root Node with *initial State*, *Avoidance Grid* and respective parameters for each calculation method. *Harmonic spread* for *Harmonic Reach set calculation* and *Chaotic Spread*, *Footprint Length* for *Chaotic Reach set calculation.*

*Separate Reach Sets* are calculated using *Wave-front propagation* (alg. 6.1) using respective *Constrained Expansion* functions for *Harmonic* (alg. 6.3) and *Chaotic* (alg. 6.2) reach sets.

*Combined Reach Set* is created using *Node mergeTree(...)* function. Because different cost function or *Bounded Parameters Calculation* may be applied on *Original Reach Sets.*

*Cost* for *each node* needs to be recalculated due to original reach sets disparity. Function *combined.applyCostFunction()* will recalculate the new cost for each node.

The Goal is to have penalization for *Chaotic behaviour*, implementation of *Automatic Mode Switch* can be done like follows:

1. *Calculate Normal Cost* for Node $Cost(Node)$ for associated trajectory: $Cost(Node.Trajectory)$.

2. *Calculate Penalization for* Chaotic Behaviour, calculate *Smoothness Rating for Trajectory* (def. 31) in interval $[0, 1]$, introduce penalization with base 100%.

The final $Cost(Node)$ function is applied on each *Combined Reach Set Node* and look like follows:

$$Cost(Node) = Cost(Node.Trajectory) \times \ldots \\ \cdots \times (1 + (1 - SmoothnessRate(Node.Trajectory))) \quad (6.60)$$

**Merge Tree Function** $mergeTree(\ldots)$ implements *Outer Join* operation on two trees. Example was given in (eq. 6.59). Function is applied on *root Node* iterating over *Movements in Movement Set*, because *Movement is pivot*.

**Algorithm 6.4:** Reach Set Merge Function and Combined Reach Set calculation

```
# Tree merge function;
Node mergeTree(Node firstNode, Node secondNode)
    # Try to copy reference node or return null;
    Node referenceNode = (firstNode?:(secondNode?: return null));
    Node merged = new Node(referenceNode);
    merged.leafs= [];
    # Try to fetch movement nodes if exist in any sub tree;
    for movement ∈ Movements do
        firstLeaf = firstNode.getLeafFor(movement);
        secondLeaf = secondNode.getLeafFor(movement);
        newLeaf = mergeTree(firstLeaf,secondLeaf);
        if newLeaf ∼= null then
            merged.leafs.append(newLeaf);
        end
    end
    return merged
# Combined Reach Set calculation function;
Node combinedReachSet(Node root, AvoidanceGrid grid,int+ chaoticSpread,
 int+ harmonicSpread, int+ footprintLength)
    Node chaotic = chaoticReachSet(root,grid, footprintLength,chaoticSpread);
    Node harmonic = harmonicReachSet(root,grid, harmonicSpread);
    Node combined = mergeTree(chaotic,harmonic);
    combined.applyCostFunction();
    return combined
```

**Example:** for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45°, +45°]$, *Horizontal Cell Count 7*, *Vertical range* $[-30°, +30°]$, and *Vertical Cell Count 5*. Is given in (fig. 6.7). The UAS is at *Back-side* of *Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid* space boundary. Each trajectory has its own color and ends at *Front-side* of *Avoidance Grid Boundary*. The *Chaotic Spread* was set to 8, *Footprint Length* to 3 and *Harmonic Spread* to 1.

*Note.* Notice there are typical trajectories from both *Harmonic* (fig. 6.6) and *Chaotic* (fig. 6.5) *Reach Set Approximations*.

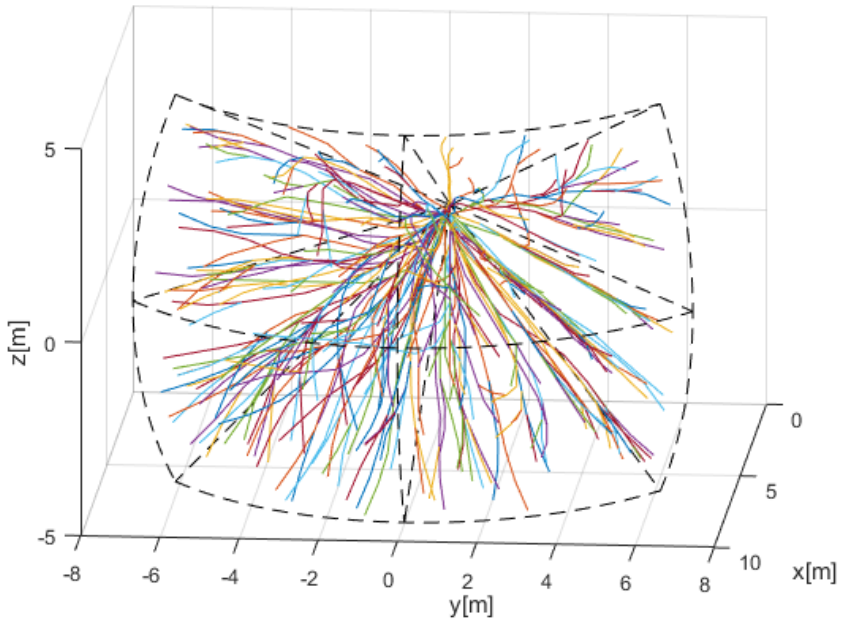



Figure 6.7: *Combined* reach set *approximation.*

**Pros and Cons:** It can bee seen from example (fig. 6.7) that *Combined Reach Set Approximation* (alg. 6.4) contains both types of maneuvers. *Cheaper Smooth* for navigation and *More Expensive Chaotic* for *Emergency Avoidance.* The upper limit of trajectories is given as follow:

$$countTrajectories(ReachSet) \leq countTrajectories(Chaotic) + countTrajectories(Harmonic) \quad (6.61)$$

The *upper limit of nodes* is given as follow:

$$countNodes(ReachSet) \leq countNodes(Chaotic) + countNodes(Harmonic) \quad (6.62)$$

*Harmonic Reach Set* is ideal for *Non-controlled Airspace* missions, because it contains *Automatic Mode Switch* between *Navigation* and *Emergency Avoidance.*

### 6.4.6 (R) ACAS-X like Reach set

**Motivation:** The implementation of *ACAS-Xu* behavior in DAA system will be mandatory for *National Airspace System Integration* in United spaces [136].

Implementation of ACAS-Xu like behaviour increase usability of approach, if it can be achieved without major concept changes.

**Background:** The *ACAS-Xu* system on operational level has been described in [22]. The *Policy for Collision Avoidance* proposal has been given in [137].

Some behavioural patterns can be encoded into *Reach Set.* ACAS-Xu navigation part is basically *Look-up table of Maneuvers for Allowed Separations.*

The *Evasive Maneuver* selection process in ACAS-Xu is similar to our approach: *Select most energy efficient maneuver in compliance with space-time constraints.* ACAS-Xu intruder model is similar to our *Body Volume Intersection Model* (sec. 6.6.3). The *ACAS-Xu* defines following base separations:

1. *Horizontal* - movements on *Horizontal Plane* in *Global Coordinate System.*
2. *Vertical* - movements on *Vertical Plane* in *Global Coordinate System.*

There are allowed custom separations which can be used, for further experimentation:

1. *Slash* - movement on $+45°$ *Tilted Plane to Horizontal Plane* in *Global Coordinate System.*
2. *Backslash* - movement on $-45°$ *Tilted Plane to Horizontal Plane* in *Global Coordinate System.*

For given *Movement Automaton* implementation (sec. 6.2) the separations are given as follow:

$$\begin{aligned} Horizontal &= \{Straight, Left, Right\} \\ Vertical &= \{Straight, Up, Down\} \\ Slash &= \{Straight, UpLeft, DownRight\} \\ Backslash &= \{Straight, UpRight, DownLeft\} \end{aligned} \tag{6.63}$$

For each $Node(\dots, buffer)$ and each *separation* there is a evaluation function *isSeparation* which decides, if *Trajectory* defined by node buffer is made up only from *Separation* movements. The function $isSeparation(\dots)$ is defined like:

$$isSeparation(buffer, separation) = \begin{cases} \forall movement \in buffer, & \\ \quad movement \in separation & : true \\ otherwise & : false \end{cases} \tag{6.64}$$

Following *Separation Modes* can be defined with given *separations*:

1. *Horizontal* (ACAS-X defined mode) containing *horizontal* separation.
2. *Vertical* (ACAS-X defined mode) containing *vertical* separation.
3. *Horizontal-Vertical* (ACAS-X defined mode) containing *horizontal, vertical* separations.
4. *Full* (custom defined mode) containing all *Separation Modes.*

*Note.* Every separation modes generates 2D trajectories set on *Respective plane.* There is no need for *Tuning parameters* for further *Expansion Constraint.*

**Expansion Constraint Function Implementation** (alg. 6.5) is based on simple principle: *Select only candidate Nodes which Trajectories have at least one desired Separation Mode.*

**Step: Initialization** sets candidate *Nodes* as empty set, leftover *Nodes* as empty set, and, select all nodes form *stack* which represents *Finishing Trajectories* in working $cell_{i,j,k}$,

**Step: Candidate Selection Process** is evaluated for each *test Node* from *passing Node Set*.

For each *applicable separation*, given as input parameter *separations*, The test function *isSeparation* (eq. 6.64) is applied:

1. If *test Node* trajectory belongs to at least one allowed separation it is added to candidates set.

2. Else is added to *Leftovers*.

*Note. Separation sets* (eq. 6.63) are not *exclusive sets* in *Movement Automaton* domain. One *Trajectory* contained by Node can belong to multiple *Separations*.

**Algorithm 6.5:** Expansion Constraint function for *ACAS-like Reach Set Approximation*

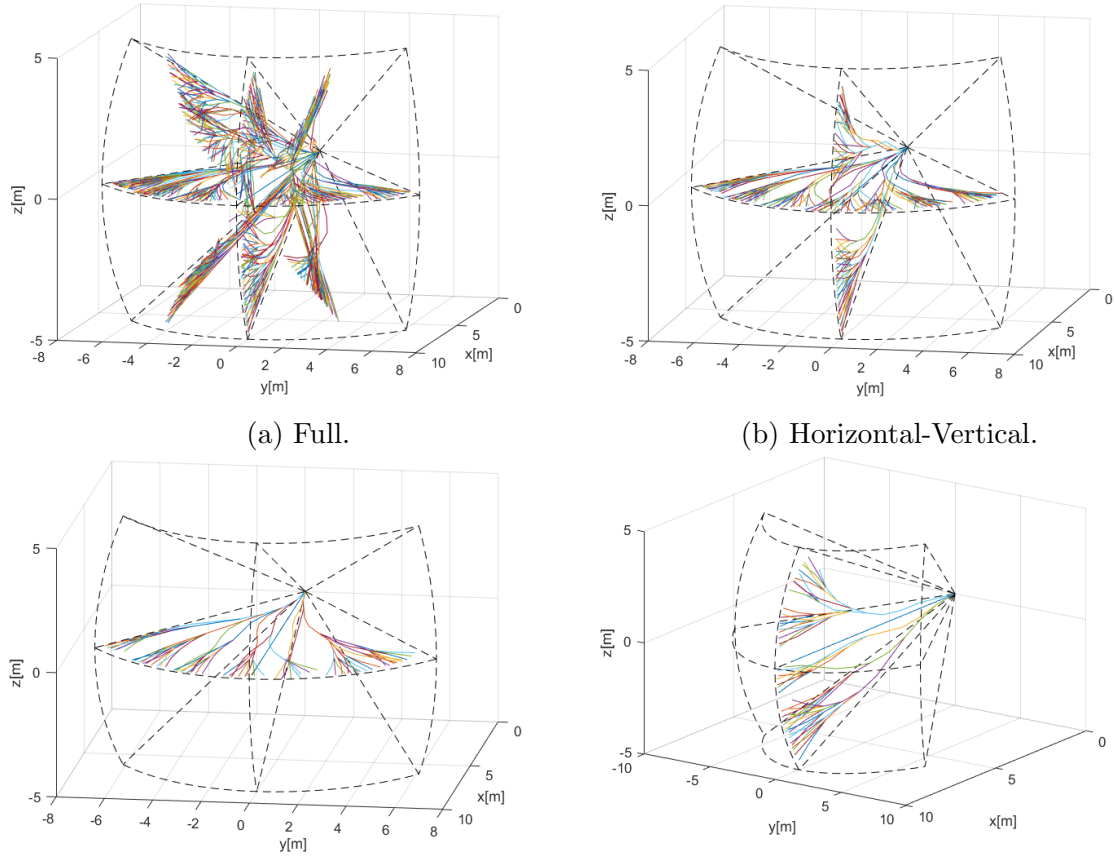
**Input** : Node[] stack, Cell $cell_{i,j,k}$, Separation[] separations
**Tuning Parameters:** $None : \varnothing$
**Output** : Node[] candidates, Node[] leftovers

```
# Initialize structures;
Node[] candidates = [], Node[] leftovers=[];
Node[] passing = cell_{i,j,k}.getFinishingTrajectories(stack);

# Select nodes containing trajectories with usable separations;
for Node test ∈ passing do
    for separation ∈ separations do
        # Get separations for Node;
        Separations[] nodeSeparations = test.getSeparations();
        # If trajectory given by buffer is on Separation plane;
        if isIn(isSeparation(test.buffer,separation)(6.64) then
            candidates.append(test);
        end
    end
    # If there was no applicable separation, throw Node away;
    if test ∉ candidates then
        leftovers.append(test);
    end
end
# Return results;
return [candidates,leftovers]
```

**Example:** for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45°, +45°]$, *Horizontal Cell Count 7*, *Vertical range* $[-30°, +30°]$, and *Vertical Cell Count 5*. Is given in (fig. 6.8). The UAS is at *Back-side* of *Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid* space boundary. Each trajectory has its own color and ends at *Front-side* of *Avoidance Grid Boundary*.

*Full* separation mode given in (fig. 6.8a). *Horizontal-Vertical* separation mode, used in original *ACAS-Xu* testing [22], given in (fig. 6.8b). *Horizontal* separation mode given in (fig. 6.8c) is usually used by planes. *Vertical* separation mode given in (fig. 6.8d) is usually used by copters.




(a) Full.

(b) Horizontal-Vertical.

(c) Horizontal.

(d) Vertical.

Figure 6.8: ACAS-X imitation *reach set* approximation for various *separation modes*.

**Pros and Cons:** It can be seen from examples (fig. 6.8) that *ACAS-like Reach Set Approximation Method* (alg. 6.5) generates full reach set for 2D plane located in 3D space.

The *Reach Set* contains trajectories with *high coverage ratio* and *high smoothness rating* for selected 2D separation plane. Overall performance compared to full 3D reach sets (sec. 6.4.3, 6.4.4 6.4.5) is poor.

The *node* and *trajectory* count boundary was not implemented. It is common knowledge that *2D* avoidance sets does not require scaling [22]. Otherwise trajectory footprint mechanism like in *Harmonic Reach Set Approximation* (alg. 6.3) can be introduced.

This reach set implements *Planar-Separation* as native feature, it can be used for both *navigation* and *avoidance* tasks in *Controlled Airspace*. For *Non-controlled Airspace* there are far more superior *Combined Reach Set* (sec. 6.4.5).

## 6.5 (R) Static Obstacles and Constraints

**Introduction:** The *static obstacles* were used in original concept [138], the *Avoidance Grid* and *Movement Automaton* were repurposed to enable *finite time deterministic* avoidance. An *Constraint based path search* and *obstacle modeling* is summarized in [139].

This section is handling basic problems of *static obstacle* detection and its focused on following real-world fixed position threats:

1. *Static Obstacles* - detected by LiDAR sensor or fused from *Obstacle Map* information source.

2. *Geo-fencing Areas* - defined by offline/online information source as permanent flight restriction zones. There is usually no physical obstacle. The space is considered as *hard/soft constraint*.

3. *Long-term bad weather Areas* - the *weather* is changing often (hour period), there are *weather events* which lasts for *hours* or *days*.

**Changing Scanning Density of LiDAR:** A LiDAR sensor is scanning in conic section given by $distanceRange, horizontalRange, verticalRange$, where distance range is in interval $[0, maxDistance]$, horizontal offset range is in $[-pi, pi]$, and vertical offset range is in $[\varphi_s, \varphi_e]$.

Let say that $\partial horizontal°, \partial vertical°$ is unitary angle offset in which one LiDAR send and return is executed. That means the *LiDAR* ray is sent every $\partial horizontal°, \partial vertical°$ offset movement. The *LiDAR* ray density is decreasing with *distance offset*. The same amount of *LiDAR* rays passes trough $cell_{i,j,k}$ in Avoidance Grid.

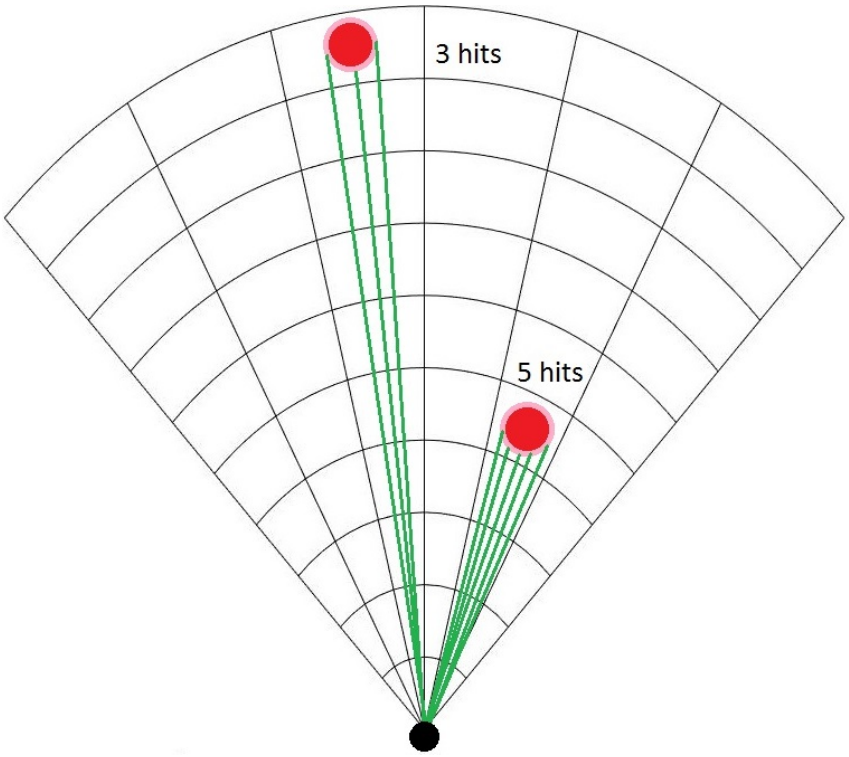



Figure 6.9: Different count of LiDAR hits with different distance from UAS.

The surface of area given by some distance d, and unitary offsets$\partial horizontal°, \partial vertical°$ is changing with *distance*. The minimal triggering area of object surface is not changing. This fact has an impact on count of the hits on object surface.

The example is given in (fig. 6.9) where we have two identical objects (red circle) in distances 5 and 10 meters. The closer object consumes 5 LiDAR beam hits and the farther object consumes only 3 LiDAR beam hits. The probability of obstacle encounter is remaining the same for closer and farther object. The *detected obstacle rate* assessment should return the same detected obstacle collision rate for objects with same scanned surface (with different LiDAR ray hit count).

**Map and Detected Obstacles Fusion:** The concept of *offline/online obstacle map* is mandatory in modern obstacle avoidance systems and increases the safety of navigation/avoidance path. The *older* concept was considering only LiDAR reading or *real-time sensor readings* in general [138].

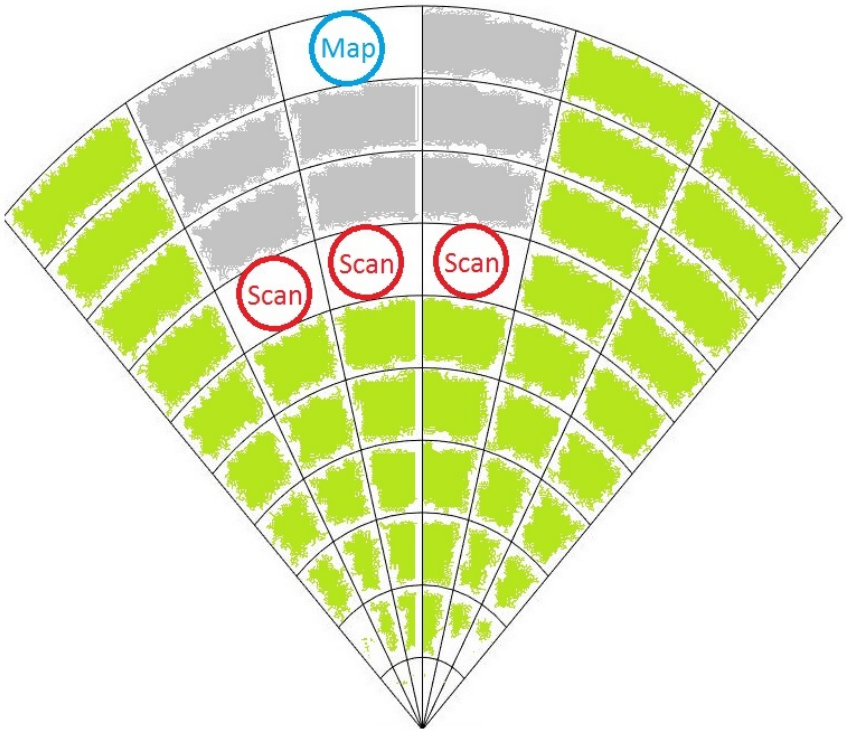



Figure 6.10: Overshadowed map obstacle by detected obstacles.

The fusion of real time sensor readings and obstacle map (prior knowledge) is required. Data fusion of these two sources is strongly depending on visibility property, because there are three basic scenarios:

1. *Dual detection* - the obstacle is marked on the map and detected by sensory system at some point of the time (older concept works).

2. *Hindered vision* - the detected obstacles are hindering vision to map obstacle therefore map obstacle uncertainty arises (older concept fails).

3. *False-positive map* - map obstacle occupied space is visible by sensory system, but negative detection is returned. Therefore the map is giving *false-positive* information.

The second case is given in fig. 6.10, where map obstacle (blue circle) is overshadowed by three scanned obstacles (red circle). The visible space is denoted by green fill, the invisible space is denoted by gray fill.

### 6.5.1 (R) Detected obstacles

**Idea:** The *visibility* inside avoidance grid and *obstacle* probability are interconnected for most ranging sensors (ex. LiDAR). The goal of this section is to introduce *visibility hindrance* concept which includes space uncertainty assessment and detected obstacle processing.

**Detected Obstacle Rating:** The *detected obstacle rating* defines UAS chances to encounter detected obstacle in avoidance grid $cell_{i,j,k}$. Final *detected obstacle rating* is merged information (eq. 6.133). The *sensor field* can contain *multiple static obstacle sensors*.

**Detected Obstacle Rate for LiDAR:** ,Lets have only one sensor set as homogeneous two axis rotary LiDAR. For one $cell_{i,j,k}$ there exists set of passing LiDAR beams:

$$lidarRays(cell_{i,j,k}) = \left\{ \begin{bmatrix} horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \tag{6.65}$$

The horizontal and vertical offset of LiDAR ray is homogeneous. Meaning the horizontal/vertical distances between each two neighbouring LiDAR beams are equal.

The set $lidarRays(cell_{i,j,k}))$ (eq. 6.65) is finite countable and nonempty for any $c_{i,j,k}$, otherwise it will contradict the definition of avoidance grid (def. 24).

The hit function $lidarScan()$ returns a distance of single beam return for beam with dislocation $[horizontal^\circ, vertical^\circ] \in lidarRays(cell_{i,j,k})$ angle offsets. The set of LiDAR hits (eq. 6.66) in cell $cell_{i,j,k}$ is defined like follow:

$$lidarHits(cell_{i,j,k}) = \left\{ \begin{bmatrix} distance = lidarScan(), \\ horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} distance \in cell_{i,j,k}.distanceRange, \\ horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \tag{6.66}$$

The *naive* obstacle rate in case of LiDAR sensor defined as ratio between landed hits and possible hits:

$$obstacle^{LiDAR}_{cell_{i,j,k}} = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \tag{6.67}$$

*Note.* The *naive obstacle rate* (eq. 6.67) ignores that *LiDAR rays* are getting more far apart from each other. The *cell surface* is increasing with cell distance from *UAS*.

The hindrance (eq. 6.68) rate is naturally defined as supplement to naive obstacle rate. This definition is sufficient, because its reflecting the *remaining sensing capability* of LiDAR.

$$hindrance^{LiDAR}_{cell_{i,j,k}} = 1 - \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \tag{6.68}$$

**Cell Density Function:** Let's start with differential form of cell surface (eq. 3.62). The target object have several hits in *Avoidance Grid*. Target $cell_{i,j,k}$ has following properties which are used in surface calculation:

1. *Horizontal span* - defines range of horizontal scanner partition.
2. *Vertical span* - defines range of vertical scanner partition.

By rewriting (eq. 3.62) and using horizontal range parameter and inverted vertical range parameter following surface integral is obtained (eq. 6.69).

$$Area(cell_{i,j,k}) = \int_{horizontal^\circ_{start}}^{horizontal^\circ_{end}} \int_{vertical^\circ_{end}}^{vertical^\circ_{start}} radius^2 \cos(vertical^\circ) \quad \mathrm{d}vertical^\circ \mathrm{d}horizontal^\circ \tag{6.69}$$

*Note.* The *radius* parameter is *average* distance of hits landed in $cell_{i,j,k}$. This helps to reflect real *scanned surface.*

Numerically stable integration exist for boundaries $horizontal^\circ\ in[-\pi, \pi]$, $vertical^\circ \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ given as follow:

$$Area(radius, horizontalRange, vertical^\circ_{start}, vertical^\circ_{end}) = \ldots$$
$$= \begin{cases} vertical^\circ_{start} < 0, vertical^\circ_{end} \leq 0 : \\ \qquad radius^2(\sin|vertical^\circ_{start}| - \sin|vertical^\circ_{end}|) \times horizontalRange) \\ vertical^\circ_{start} < 0, vertical^\circ_{end} > 0 : \\ \qquad r^2(\sin|vertical^\circ_{start}| + \sin|vertical^\circ_{end}|) \times horizontalRange) \\ vertical^\circ_{start} \geq 0 vertical^\circ_{end} < 0 : \\ \qquad r^2(\sin vertical^\circ_{end} - \sin vertical^\circ_{start}) \times horizontalRange) \end{cases} \tag{6.70}$$

An intersection surface for cell is defined in (eq. 6.70). Area covered by LiDAR hits (eq. 6.71) is defined as LiDAR hit rate (hits to passing rays ratio) multiplied by *Average* cell intersection surface (eq. 6.70).

$$lidarHitArea(cell_{i,j,k}) = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \times Area \begin{pmatrix} radius, horizontalRange, \\ vertical^\circ_{start}, vertical^\circ_{end} \end{pmatrix} \tag{6.71}$$

There is user defined parameter for *LiDAR treshold area*, which represents minimal considerable surface area for obstacle to be threat. The *detected obstacle rate* considering surface is defined in (eq. 6.72) and it removes bias of naive approach (eq.6.67).

$$obstacle(LiDAR, cell_{i,j,k}) = \min\left\{\frac{lidarHitArea(cell_{i,j,k})}{UAS.lidarThresholdArea}, 1\right\} \tag{6.72}$$

**Visibility Rate for LiDAR:** For each $cell_{i,j,k}$ and each sensor in sensor field there exist hindrance rate, which defines how much vision is clouded in single cell. Example of hindrance calculation for LiDAR has been given by (eq. 6.68). Let us consider cell row $cellRow(j_{fix}, k_{fix})$ with fixed horizontal index $j_fix$ and vertical index $k_{fix}$ is given as series of cells (eq. 6.73).

$$cellRow(j_{fix}, k_{fix}) = \left\{ cell_{i,j,k} \in AvoidanceGrid : \begin{array}{l} i \in \{1, .., layersCount\}, \\ j = j_{fix}, k = k_{fix} \end{array} \right\} \tag{6.73}$$

For each $cell_{i,j,k}$ there exists a function which calculates final visibility hindrance rate. Then for ordered cell row:

$$cellRow(j_{fix}, k_{fix}) = \{cell_{1,j_{fix},k_{fix}}, cell_{2,j_{fix},k_{fix}}, \ldots, cell_{layersCount,j_{fix},k_{fix}}\}$$

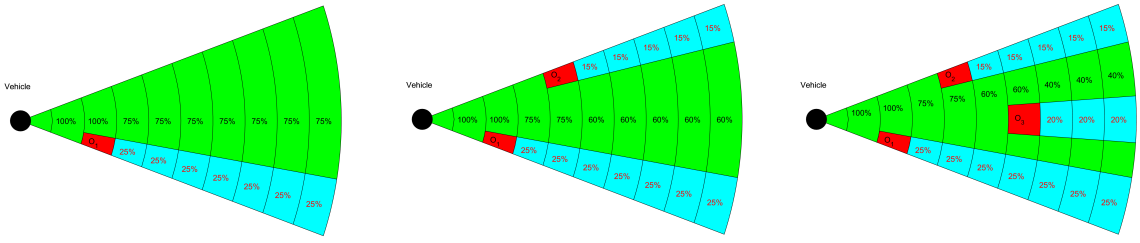
and for one selected $cell_{i,j,k}$ the visibility rate is naturally defined as a supplement to hindrance from previous cells. The visibility is defined in (eq. 6.74).

$$\begin{aligned} visibility(cell_{i_c,j_c,k_c}) &= \ldots \\ \cdots &= 1 - \sum_{index \in \mathbb{N}^+}^{index<i_c} hindrance(cell_{a,j_c,k_c} : cell_{a,j_c,k_c} \in cellRow(j_c, k_c) \end{aligned} \tag{6.74}$$

**Example:** Let be $cell_{4,j_{fix},k_{fix}}$ is selected for visibility rate assessment, then $cell_{1,j_{fix},k_{fix}}$, $cell_{2,j_{fix},k_{fix}}$, and $cell_{3,j_{fix},k_{fix}}$, are used as a base of cumulative hindrance rate.
*The cumulative hindrance rate* for any $cellRow(j_{fix}, k_{fix})$ is bounded:

$$0 \leq \sum_{cell \in cellRow(j_{fix},k_{fix})} visibility(cell) \leq 1 \tag{6.75}$$

*Note.* A *cumulative hindrance rate* does not always reach 1 in case of LiDAR sensor, because some rays may pass or hit after leaving *avoidance grid* range.




(a) 1st hindrance. (b) 2nd hindrance. (c) 3rd hindrance.

Figure 6.11: Obstacle hindrance impact on visibility in *Avoidance Grid Slice*.

For one cell row $cellRow(j_{fix}, k_{fix})$, where count of layers is equal to 10, and layers have equal spacing. There is LiDAR sensor

During consequent LiDAR scans $s(t_0)$, $s(t_1)$, $s(t_2)$, and $s(t_3)$ the obstacle sets $\mathcal{O}_1(t_1) = \{o_1\}$, $\mathcal{O}_2(t_2) = \{o_1, o_2\}$, and $\mathcal{O}_3(t_3) = \{o_1, o_2, o_3\}$ are discovered. Assigned hindrance rates are like follow:

1. *Time* $t_0$ - there is no obstacle nor hindrance, all cells are fully visible.

2. *Time* $t_1$ (fig. 6.11a) - $\mathcal{O}_1(t_1) = \{o_1\}$ was detected, the hindrance rate for $cell_{3,j_{fix},k_{fix}}$) is equal to 0.25. The visibility rate in cells $cells_{4-10,j_{fix},k_{fix}}$ is 0.75.

3. *Time* $t_2$ (fig. 6.11b) - $\mathcal{O}_2(t_2) = \{o_1, o_2\}$ was detected, the additional hindrance rate for $cell_{5,j_{fix},k_{fix}}$ is 0.15. The visibility rate in $cells_{6-10,j_{fix},k_{fix}}$ is lowered by additional 0.15 and its set to 0.60 now.

4. *Time* $t_3$ (fig. 6.11c) - $\mathcal{O}_3(t_3) = \{o_1, o_2, o_3\}$ was detected the additional hindrance rate for $cell_{7,j_{fix},k_{fix}}$ is 0.20. The visibility rate in $cells_{8-10,j_{fix},k_{fix}}$ is lowered by additional 0.20 and its set to 0.40 now.

### 6.5.2 (R) Map obstacles

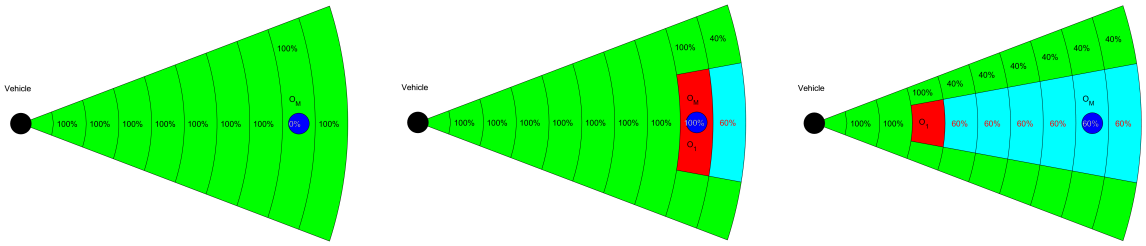
**Idea:** Use *stored LiDAR readings* from previous mission to build an compact obstacle map [11]. Then use *this map* as a additional information source.

**Concept:** A *map obstacle* state has very simple logic, there are three possible cases:

1. *Undetected* - Map obstacle $O_M$ is charted on map (fig. 6.12a), but is undetected by any sensor in sensor field, therefore the probability of map obstacle occurrence is equal to 0.

2. *Detected* Map obstacle $O_M$ is charted on map and detected by any sensor in sensor field (fig. 6.12b). The map obstacle rate is equal to detected obstacle rate, usually its equal to 1.

3. *Hindered* Map obstacle $O_M$ is hindered behind other detected obstacle $O_1$ (fig. 6.12c). The detected obstacle $O_1$ is in $cell_{i,j,k}$ and is reducing visibility in follow up $cellRow_{if>i,j,k}$ by 60 percent.




(a) Undetected. (b) Detected. (c) Hindered.

Figure 6.12: Map obstacle states after *Data fusion*.

**Implementation:** The formulation of final map obstacle rate $map(cell_{i,j,k})$ was outlined in previous examples. These examples are showing the *desired behaviour* and its solved by *data fusion* (sec. 6.7.1).

First we start with obstacle map definition. The obstacle map (eq. 6.76) defines an map obstacle set of information vectors with position in global coordinate frame ,

orientation bounded to global coordinate reference frame, safety margin and additional parameters.

$$obstacleMap = \left\{ \begin{bmatrix} position, \\ orientation, \\ safetyMargin, \\ parameters \end{bmatrix} : \begin{matrix} position \in \mathbb{R}^3(GCF), \\ orientation \in \mathbb{R}^3(GCF), \\ safetyMargin \in \mathbb{R}^+(m), \\ parameters \in \{\dots\} \end{matrix} \right\} \tag{6.76}$$

The *Map Obstacle* concept is taken from my *master student work* [11], implementing *compact representation* of point-cloud obstacle map. Te example of *cuboid obstacles* with *safe zone* is given in (fig. 6.13).

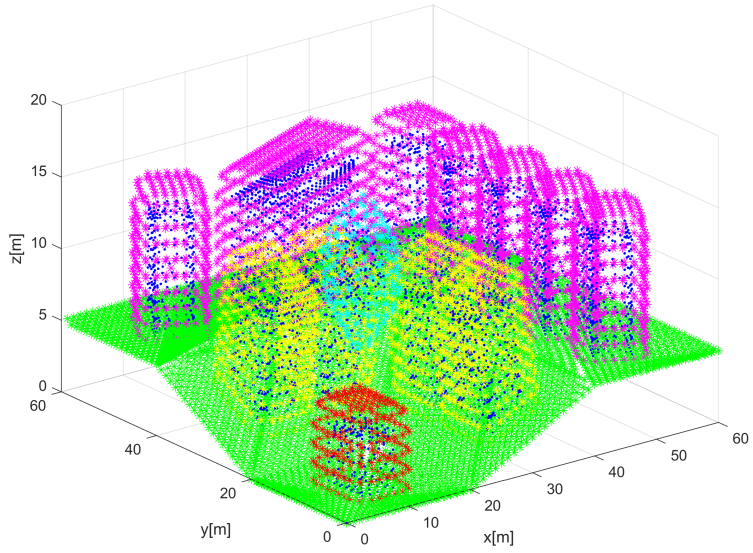



Figure 6.13: Example of Extracted Map Obstacle [11].

The space covered by any obstacle is non-empty by definition. There are following types of map charted obstacles which are implemented in framework:

1. *Ball obstacle parameters* $= \varnothing$ - simple ball with center at *position*, with offset safety margin.

2. *Line obstacle parameters* $= [length]$ - simple line bounded by length $\in ]0, \infty[$ with center at *position* and given orientation with respect to main axis in global coordinate frame, with safety margin $< 0$.

3. *Plane obstacle parameters* $= [length, width]$ - bounded rectangle plane partition defined by length $\in ]0, \infty[$, and width $w \in ]0, \infty[$ with center at $\vec{p}$ and given orientation $\vec{o}$ with respect to main axis in global coordinate frame, with safety margin.

4. *Cuboid obstacle parameters* $= [length, width, depth]$ - bounded cuboid space partition defined by length $\in ]0, \infty[$, width $\in ]0, \infty[$, and depth $d \in ]0, \infty[$ with center at *position* and rotated in orientation with respect to main axis in global coordinate frame, with safety margin.

The *map obstacles* are stored in clustered database. The *selection criterion* is given in (eq. 6.77).

$$avoidanceGrid.radius \geq distance(UAS.position, mapObstacle) - totalMargin \quad (6.77)$$

The *total margin* is combination of *safety margin* and *body margin* (in case of line, plane, cuboid obstacle). The *selection* was implemented as standard cluster select, selecting 26 surrounding clusters around UAS + own UAS cluster.

The *compact obstacle representation* is transformed into *h*omogeneous point-cloud representations:

1. *Body Point-cloud* - representing obstacle body approximation by geometrical shape (eq. 6.78). This point cloud is considered as hard constraints.

$$bodyPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapObstacleBody\} \quad (6.78)$$

2. *Safety Margin Point Cloud* - representing safety coating around mapped obstacle body approximation (eq. 6.79). This point cloud is considered as soft constraint.

$$marginPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapSafetyMargin\} \quad (6.79)$$

*Note.* The *safety margin point cloud* is hollow in relationship to an *body point cloud*, therefore:

$$bodyPointCloud \cap marginPointCloud = \varnothing$$

The *map obstacle* discretization to point cloud leads to problem how to calculate *impact rate*. The *theoretical impact rate* for *obstacle* is given as:

$$impactRate = \frac{volume(mapObstacle \cap cell_{i,j,k})}{volume(cell_{i,j,k})} \in [0, 1]$$

The *map obstacle related point clouds* (eq. 6.78, 6.79) are homogeneous [11]. That means *each point* in point clouds covers similar portion of object volume. There is *threshold volume* (eq. 6.80) which represents minimal object volume to be considered as an *obstacle*.

$$0 < thresholdVolume \leq \frac{volume(pointCloud)}{|pointCloud|} \quad (6.80)$$

The *impact rate* of one point when intersecting a $cell_{i,j,k}$ is given as count of *threshold obstacle bodies* in *point cloud covered mass* multiplied by inverted point count (eq. 6.81).

$$point.rate = \frac{pointCloudVolume}{thresholdVolume} \times \frac{1}{|pointCloud|} \quad (6.81)$$

The *intersection set* between *point cloud* and $cell_{i,j,k}$ is defined in (eq. 6.81). The *cell* intersection with points is defined in (eq. 6.26).

$$intersection(map, cell_{i,j,k}) = \ldots$$
$$\ldots \{points \in \mathbb{R}^3 : (point \rightarrow AvoidanceGridFrame) \in cell_{i,j,k}\} \quad (6.82)$$

The *map obstacle rating* for $cell_{i,j,k}$ and obstacle for our *information source* is defined in (eq. 6.83).

$$map(cell_{i,j,k}, obstacle) = \max \left\{ \sum_{\forall point \in intersection(map, cell_{i,j,k})} point.rate, 1 \right\} \tag{6.83}$$

The *map obstacle rating* for $cell_{i,j,k}$ and *our information source* is given as maximum of all possible cumulative ratings form each obstacle in *active map obstacles* set (eq. 6.84).

$$map(cell_{i,j,k} = \max \{map(cell_{i,j,k}, obstacle) : \forall obstacle \in ActiveMapObstacles\} \tag{6.84}$$

*Note.* The *body point clouds* (eq. 6.78) never intersects, because they are created for inclusive obstacles. The *safety margin point clouds* (eq. 6.79) can intersects, because they represents protection zones around physical obstacles. Therefore the *maximum obstacle rating* (eq. 6.84) needs to be selected.

### 6.5.3 (R) Static constraints

**Idea:** The *constraints* (ex. weather, airspace) usually covers large portion of the *operation airspace*.

Converting constraints into valued *point-cloud* is not feasible, due the *huge amount of created points* and low *intersection rate*. The *polygon intersection* or *circular boundary of 2D polygon* is simple and effective solution [33, 140].

The key idea is to create *constraint barrels* around dangerous areas. Each *constraint barrel* is defined by circle on *horizontal plane* and *vertical limit range*.

**Representation:** The *minimal representation* is based on (sec. 2.3, 2.5.2) and geofencing principle. The *horizontal-vertical separation* is ensured by *projecting boundary* as 2D polygon oh horizontal plane and *vertical boundary* (barrel height) as *altitude limit*.

The *static constraint* (eq. 6.85) is defined as structure vector including:

1. *Position* - the center position in global coordinates*2D horizontal plane*.
2. *Boundary* - the ordered set of boundary points forming edges in global coordinates*2D horizontal plane*.
3. *Altitude Range* - the *barometric altitude* range [$altitude_{start}$, $altitude_{end}$].
4. *Safety Margin* - the *protection zone* (soft constraint) around constraint body (hard constraints) in meters.

$$constraint = \{position, boundary, altitude_{start}, altitude_{end}, safetyMargin\} \tag{6.85}$$

**Active constrain selection:** The *active constraints* are constraints which are impacting *UAS active avoidance range*.

The *active constraints set* (eq. 6.86) is defined as set of *constraints* from all *reliable Information Sources* where the *the distance* between UAS and constraint body (including safety margin) is lesser than the avoidance grid range. The *horizontal altitude range* of avoidance grid musts also intersect with *constraint altitude range*.

$$\begin{aligned} &ActiveConstraints = \dots \\ &\cdots = \left\{ \begin{array}{r} constraint \in InformationSource : \\ distance(constraint, UAS) \leq AvoidanceGrid.distance, \\ constraint.altitudeRange \cap UAS.altitudeRange \neq \varnothing \end{array} \right\} \end{aligned} \tag{6.86}$$

**Cell Intersection:** The *importance of constraints* is on their impact on *avoidance grid cells*. The *most of the constraints* (weather, ATC) are represented as 2D convex polygons. Even the *irregularly shaped constraints* are usually split into smaller convex 2D polygons.

The idea is to represent convex polygon boundary as sufficiently large circle to cover polygon. The Welzl algorithm to find *minimal polygon cover circle* [33] is used.

First the *set of contraint edges* (eq. 6.87) is a enclosed set of 2D edges between neighboring points defined as follow:

$$edges(constraint) = \left\{ \begin{array}{c} point \in boundary, \\ \left[point_i, point_j\right] : i \in \{1, \dots, |boundary|\}, \\ j \in \{2, \dots, |boundary|, 1\} \end{array} \right\} \tag{6.87}$$

The *constraint circle boundary* with calculated center on 2D horizontal plane and radius (representing body margin) is defined in (eq. 6.88).

$$circle(constraint) = \left[ \begin{array}{ll} center = \dfrac{\sum boundary.point}{|boundary.point|} + correction & [33] \\ radius = smallestCircle(edges(constraints)) & [33] \end{array} \right] \tag{6.88}$$

The ($cell_{i,j,k}$ and *constraint* intersection (eq. 6.89) is classification function. The *classification* is necessary, because one *constraint* induce:

1. *Body Constraint* (hard constraint) - the distance between $cell_{i,j,k}$ closest border and *circular boundary* center is in interval $[0, radius]$.

2. *Protection Zone Constraint* (soft constraint) - the distance between $cell_{i,j,k}$ closest border and *circular boundary* center is in interval $]radius, radius + safetyMargin]$.

$$intersection, constraint) = \ldots$$

$$\ldots = \begin{cases} hard & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) \leq \ldots \\ \quad \ldots \leq circle(constraint).radius, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\ soft & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) > \ldots \\ \quad \ldots > circle(constraint).radius, \\ distance(cell_{i,j,k}, circle(constraint)) \leq \ldots \\ \quad \ldots \leq circle(constraint).radius + safetyMargin, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\ none & : otherwise \end{cases} \tag{6.89}$$

The *intersection impact* of constraint is handled separately for *soft* and *hard* constraints. The *avoidance* of hard constraints is *mandatory*, the *avoidance* of soft constraints is *voluntary*.

The constraints which have an *soft intersection with cell* are added to cells impacting constraints set:

$$cell_{i,j,k}.softConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = soft \end{matrix} \right\} \tag{6.90}$$

The constraints which have an *hard intersection with cell* are added to cells impacting constraints set:

$$cell_{i,j,k}.hardConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = hard \end{matrix} \right\} \tag{6.91}$$

*Note.* The final *constraint rate value* (eq. 6.136) is determined based on *mission control run* feed to *avoidance grid* (fig. 6.25) defined in $7^{\text{th}}$ to $10^{\text{th}}$ step.

## 6.6 (R) Intruders and Moving Constraints

**Intruder behaviour:** *Adversarial behaviour* of moving obstacle is trying to destroy avoiding our UAS. The *Intruder* UAS [141] is not trying to hurt our *UAS* actively. The *Adversarial behaviour* is neglected in this work. The non-cooperative avoidance is assumed, it can be relaxed to *cooperative avoidance* in *UTM controlled airspace*.

**Intruder information:** The *observable intruder information set* for any kind of intruder, obtained trough sensor/C2 line, is following:

1. *Position* - position of intruder in *local* or *global* coordinate frame, which can be transformed into *avoidance grid coordinate frame*.
2. *Heading and Velocity* - intruder heading and linear velocity in avoidance grid coordinate frame.
3. *Horizontal/Vertical Maneuver Uncertainty Spreads* - how much can an *intruder* deviate from *original linear path* in *horizontal/vertical* plane in *Global coordinate Frame*.

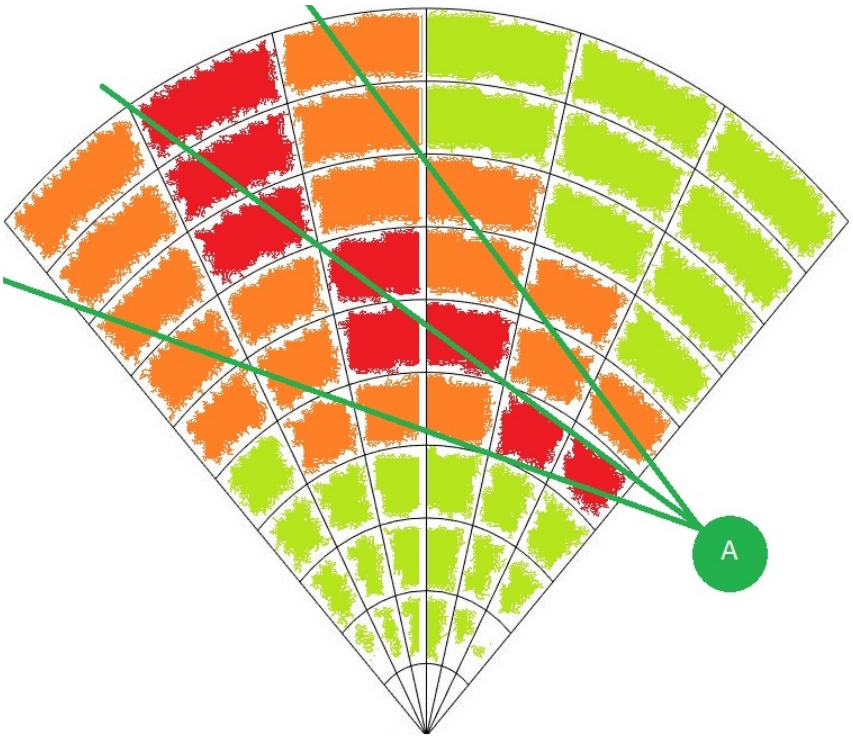



Figure 6.14: Intruder UAS intersection rate along expected trajectory.

**Example of Intruder Intersection:** Lets neglect the *time-impact* aspect on *intersection*. The *intruder* (black "I" circle) is intersecting one *avoidance grid horizontal slice* (fig. 6.14). The intruder is moving along linear path approximation based on velocity (middle green line). The *Horizontal Maneuver Uncertainty spread* is in *green line boundary area intruder intersection rating* is denoted as green-orange-red cell fill reflecting intersection severity: red is high rate of intersection, orange is medium rate of intersection and green is low rate of intersection.

**Moving Threats:** The *UAS* can encounter following threats during the *mission execution*:

1. *Non-cooperative Intruders* - the intruders whom does not implement any approach to ensure mutual avoidance efficiency.

2. *Cooperative Intruders* - the intruders whom actively communicate or follow common agreed behaviour pattern (ex. Rules of the Air).

3. *Moving Constraints* - the constrained portion of *free* space which is shifting its boundary over time (ex. Short term bad weather).

*Note.* Our approach considers only *UAS* intruders, because *Data Fusion* considers data received trough *ADS-B* messages. The *Intruders* extracted from *LiDAR* scan were not considered (ex. birds). The proposed *intruder intersection models* are reusable for other *intruder sources*.

**Approach Overview:** The *Avoidance Grid* (def. 24) is adapted to *LiDAR* sensor. The *euclidean grid intersections* are fairly simple. The *polar coordinates grid* are not. The need to keep *polar coordinates grid* is prevalent, because of fast *LiDAR* reading assessment. There are following commonly known methods to address this issue:

1. *Point-cloud Intersections* - the *threat impact area* is discredited into sufficiently thick point cloud. This point-cloud have *point impact rate* and *intersection time* assigned to each point. The *point-cloud* is projected to *Avoidance Grid*. If *impact point* hits $cell_{i,j,k}$ the cell's impact rate is increased by amount of *point impact rate*. The final *threat impact rate* in $cell_{i,j,k}$ is given when *all* points from point cloud are consumed. Close point problem [142] was solved by application of method [143].

2. *Polygon Intersections* - the *threat impact area* is modeled as polygon, each $cell_{i,j,k}$ in *Avoidance Grid* is considered as *polygon*. There is a possibility to calculate cell space geometrical inclusive intersection. The *impact rate* is then given as rate between *intersection volume* and $cell_{i,j,k}$ volume. The algorithm used for intersection selected based on:[144] the selected algorithm *Shamos-Hoey* [145].

*Note.* The *Intruder Intersection* models are based on *analytically geometry* for *cones and ellipsoids* taken from [146].

### 6.6.1 (R) Intruder Behaviour Prediction

**Idea:** *Intruder Intersection Models* is about space-time intersection of *intruder body* with *avoidance Grid* and *Reach Set*:

1. The *UAS* reach set defines *time boundaries* to *enter/leave* cell in avoidance grid.

2. The *Intruder* behavioral pattern defines *rate* of *space intersection* with cell bounded space in avoidance grid.

The multiplication of *space intersection rate* and *time intersection rate* will give us *intruder intersection* rate for our *UAS* and intruder.

**Intruder Dynamic Model:** The definition of avoidance grid enforces the most of these methods to be numeric. Let us introduce intruder dynamic model:

$$\partial position/\partial time = velocity \quad | \quad \begin{array}{l} position_x(t) = position_x(0) + velocity_x \times t \\ position_y(t) = position_y(0) + velocity_y \times t \\ position_z(t) = position_z(0) + velocity_z \times t \end{array} \tag{6.92}$$

Position vector in euclidean coordinates $[x, y, z]$ is transformed into *Avoidance Grid* coordinate frame. Velocity vector for $[x, y, z]$ is *estimated and not changing*. The time is in interval $[entry, leave]$, where *entry* is intruder entry time into avoidance grid and *leave* is intruder leave time from avoidance grid.

*Note.* If *intruder* is considered, time of entry is marked as $intruder_{entry,k}$ where k is intruder identification, time of leave is marked as $intruder_{leave,k}$ where k is intruder identification.

**Cell Entry and Leave Times** $UAS_{entry}(cell_{i,j,k})$ and $UAS_{leave}(cell_{i,j,k})$ are depending on intersecting *Trajectories* and *bounded cell space* (eq. 6.26). There is *Trajectory Intersection* function from (def. 26) which evaluates *Trajectory segment* entry and leave time.

The UAS *Cell Entry* time is given as minimum of all *passing trajectory segments* entry times (eq. 6.93), if there is no *passing trajectories* the UAS *entry time* is set to 0.

$$UAS_{entry}(cell_{i,j,k}) = \min \left\{ \begin{array}{l} 0, entry(Trajectory, cell_{i,j,k}) : \\ \quad Trajectory \in PassingTrajectories \end{array} \right\} \tag{6.93}$$

The UAS *Cell Leave* time is given as maximum of all *passing trajectory segments* entry times (eq. 6.94), if there is no *passing trajectories* the UAS *leave time* is set to 0.

$$UAS_{leave}(cell_{i,j,k}) = \max \left\{ \begin{array}{l} 0, leave(Trajectory, cell_{i,j,k}) : \\ \quad Trajectory \in PassingTrajectories \end{array} \right\} \tag{6.94}$$

**Time Intersection Rate:** The key idea is to calculate how long the *UAS* and *Intruder* spends together in same space portion ($cell_{i,j,k}$). The *Intruder* can spent some time in $cell_{i,j,k}$ bounded by interval of *intruder* entry/leave time.

The *UAS* can spent some time, depending on *selected trajectory* from *Reach Set*. The time spent by UAS is bounded by entry (eq. 6.93) and leave (eq. 6.94).

The intersection duration of these two intervals creates *time intersection rate* numerator, the *maximal duration* of *UAS* stay gives us *denominator*. The *time intersection rate* is formally defined in (eq. 6.95).

$$time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} = \circ \end{pmatrix} = \frac{\left| \begin{array}{c} [intruder_{entry}(\circ), intruder_{leave}(\circ)] \\ \cap \\ [UAS_{entry}(\circ), UAS_{leave}(\circ)] \end{array} \right|}{|[UAS_{entry}(\circ), UAS_{leave}(cell_{\circ})]|} \tag{6.95}$$

**Intruder Intersection Rate:** The *Intruder Intersection Rate* (eq. 6.96) is calculated as *multiplication* of *space intersection rate* (defined later) and *time intersection rate* (eq. 6.95).

$$intruder \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} = time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \times space \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \tag{6.96}$$

*Note.* If there is no information to derive *Intruder* entry/leave time for cells the *time intersection rate* is considered 1.

The *Intruder cell reach* time (eq. 6.97) is bounded to discrete point in intersection model [142, 143]. The intruder *entry/leave time* is calculated similar to *UAS cell entry (eq. 6.93)/leave (eq. 6.94) time.*

$$pointReachTime(Intruder, point) = \frac{distance(Intruder.initialPosition, point)}{|Intruder.velocity} \tag{6.97}$$

**Space Intersection Rate:** The *Space Intersection Rate* reflects probability of *Intruder* intersection with portion of space bounded by $cell_{i,j,k}$, to be precise with intruder trajectory or vehicle body shifted along the trajectory. The principles for *space intersection rate* calculation are following:

1. *Line trajectory* - *intruder* trajectory is given by linear approximation (eq. 6.92), depending on *intruder size* the intersection with avoidance grid can be:

   a. *Simple line* - intersection is going along the trajectory line line defined by intruder model (eq.6.92).

   b. *Volume line* - intersection is going along the trajectory line defined by intruder model (eq. 6.92) and intruder‘s *body radius* is considered in intersection.

2. *Elliptic cone* - initial position is considered as the top of a cone, the main cone axis is defined by intruder linear trajectory (eq. 6.92) $time \in [0, \infty]$. The cone width is set by horizontal and vertical spread.

### 6.6.2 (R) Linear Intersection

**Idea:** There are *small intruders* which have body *smaller* than average $cell_{i,j,k}$ cell size. Its trajectory will stick to *linear trajectory* prediction with high probability.

**Space Intersection Rate:** The *Space Intersection Rate* for $cell_{i,j,k}$ is implemented as simple point cloud intersection. Where *sufficiently thick* point cloud is defined along *line* (eq. 6.98):

$$position(time) = position(time_0) + velocity \times time, \quad time \in [0, \infty[ \tag{6.98}$$

Then there exist projection function from local euclidean coordinates to local polar coordinates (eq. 6.99. The function projects intruder trajectory (eq. 6.98) to planar coordinates [*distance*, *horizontal*°, *vertical*°] as a set of sufficiently thick point cloud.

$$polarSet : position(t) \rightarrow \{[distance, horizontal^\circ], vertical^\circ\} \tag{6.99}$$

The *space intersection rating* $SpaceIntersection(\circ)$ for line type is given as (eq. 6.100). If there exist non empty intersection of $polarSet \cap cell_{i,j,k}$ there is space intersection rate equal to 1, if intersection $polarSet \cap cell_{i,j,k} = \varnothing$ then the rate is zero.

$$space \begin{pmatrix} Intruder, \\ cell_{i,j,k} \end{pmatrix} = \begin{cases} 1: & \exists point \in polarSet(eq.6.99) : point \in c_{i,j,k} \\ 0: & \text{otherwise} \end{cases} \tag{6.100}$$

*Note.* The *intruder intersection rate* is multiplication of *space intersection rate* and time intersection rate. The *intersection rate* is calculated for *every intruder* and *selected intersection model* separately.

### 6.6.3 (R) Body-volume Intersection

**Idea:** The *Intruder* has body volume greater than *average* $cell_{i,j,k}$ volume. The *intruder body* is considered as the ball moving along *intruder position*. The *intersection* of the intruder body is realized as sufficiently thick *point-cloud intersection*.

**Space Intersection Rate - Body Volume:** The *body volume mass* with center at $position(t)$ is moving along intruder trajectory prediction (eq. 6.101) in time interval $[0, \infty[$:

$$position(time) = position(time_0) + velocity \times time \tag{6.101}$$

The body *Volume ball* $Body(position(t), radius)$ (eq. 6.102) is defined as set of points in $\mathbb{R}^3$ euclidean space. The center is moving along the *position(t)*. The body *volume ball* is a set of points sufficiently thick including also inner points. The *thickness* is guaranteed by existence of neighbour point which is close enough.

$$Body(position(t), radius) = \left\{ \begin{array}{c} \|position(t) - point\| \leq radius \\ point \in \mathbb{R}^3 : \forall point_i \exists point_{j \neq i}, \\ distance(point_i, point_j) \leq thickness \end{array} \right\} \tag{6.102}$$

The *polar volume ball* $polarBody$ (eq. 6.103) is projection of body volume ball set $Body(position(t), radius)$ to a set of planar coordinates in avoidance grid coordinate frame:

$$polarBall(t) : Body(position(t), radius) \rightarrow \left\{ \begin{bmatrix} distance, horizontal^\circ, \\ vertical^\circ, intersectionTime \end{bmatrix} \right\} \tag{6.103}$$

The *space intersection rate for vehicle body* $space(Intruder, cell_{i,j,k})$ (eq. 6.104) is calculated as intersection of polar body volume ball and $cell_{i,j,k}$. If intersection is non empty then base probability is one, zero otherwise:

$$space\begin{pmatrix} Intruder, \\ cell_{i,j,k} \end{pmatrix} = \begin{cases} 1: & \exists point \in polarBall(eq.6.103) : point \in c_{i,j,k} \\ 0: & \text{otherwise} \end{cases} \tag{6.104}$$

**Intersection Time:** The *intersection time* id depending on point cloud (eq. 6.103) where each point *have intersection time* given as *body-center position* time (eq. 6.101).

*Note.* The *body-volume* intersection model, can insert the *multiple intersection times* into one $cell_{i,j,k}$. the *interval length* considers all of these for intersection rates (eq. 6.95).

### 6.6.4 (R) Maneuverability Uncertainty Intersection

**Idea:** The *intruders* are not bullets they are not sticking to predicted linear paths. The *intruder* maneuverability is given as horizontal and vertical spread. Therefore *intruder reach set* will form a *elliptic cone*. This cone can be transformed into *finite discrete* point-cloud, each *point* should have assigned *severity* impact value. The point cloud intersection with *Avoidance Grid* will give us space impact of *uncertain* intruder.

*Note.* Following section will use condensed notation, due the equation complexity. The *terminology* is consistent with rest of section.

**Sprace Intersection Rate - Body Volume Intersection:** $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ computation is less straight-forward than other space intersection rates. First let us define the linear intruder $i_k$ positions $x$ at time $t$ (eq. 6.105) model, where $x(t)$ defines intruder position in *avoidance grid euclidean coordinate frame* at time $t_i$, $v$ defines intruder velocity, and $t$ is time offset.

$$x(t) = x_s + v_I.t \tag{6.105}$$

Intruder *horizontal spread* $\theta$ and *vertical spread* $\varphi$ are introduced. These spreads represents intruder deviation limits along from linear trajectory prediction $x(t) \in \mathbb{R}^3$. The example is given by (fig. 6.15) where the intruder starts at point $x_s$ with fixed velocity $v$, the linear trajectory prediction is outlined by blue line. The *predicted intruder position* at time $t = 10s$ is given by $x(10)$ (blue point). The ellipsoidal space $E(x)$ is projected on the plane $D(x(t))$. The plane $D$ (eq. 6.106) for point $x(t)$ and velocity $v$ is defined as an orthogonal plane to velocity vector $v \in \mathbb{R}^3$ with origin at intruder position $x(t)$.

$$D(x(t), v) = \left\{ a \in \mathbb{R}^3 : (a - x(t)) \perp v, \right\} \tag{6.106}$$

To construct ellipsoidal space boundary on orthogonal plane $D(x(t), v)$ some parameters are defined in (eq. 6.107). The *scalar distance* $d_d x(t)$ is simple euclidean norm, *maximal horizontal offset* $d_\theta(x_t)$ is given as product of sinus of horizontal offset angle $\theta$ and scalar distance $d_d$, and *maximal vertical offset* $d_\varphi(x(t))$ is given a product of sinus of vertical offset angle $\varphi$ and scalar distance $d_d$.

$$\begin{aligned} d_d &= d_d(x(t), x_s) = \|x(t) - x_s\|_2 \\ d_{\theta_{\max}} &= d_\theta(x(t)) = \sin\theta(i_k).d_d(x(t)) \\ d_{\varphi_{\max}} &= d_\varphi(x(t)) = \sin\varphi(i_k).d_d(x(t)) \end{aligned} \tag{6.107}$$

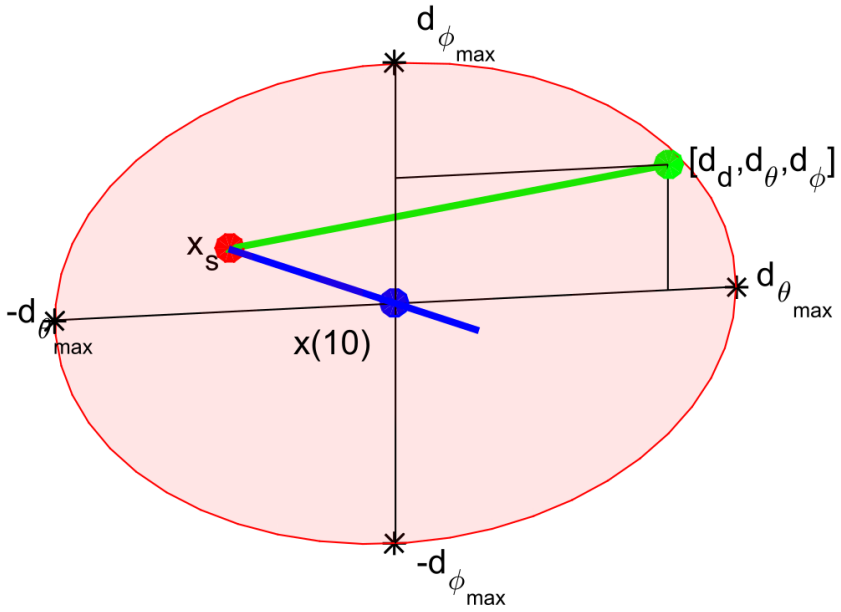



Figure 6.15: One rate position $[d_d, d_\theta, d_\varphi]$ (green). deviated from linear trajectory (blue line) at point $x(10)$.(blue) with initial position $x_s$ (red)

The *Ellipsoid* $E(x(t), v)$ (eq. 6.108) for fixed intruder position $x(t)$ and fixed intruder velocity $v$ is given as constrained portion of orthogonal plane $D(x(t), v)$. The constraint is defined by an internal coordinate frame $p \in \mathbb{R}^2$ which is space reduction of plane $D(x(t), v)$.

The internal coordinate frame $p \in \mathbb{R}^2$ has origin in $x(t) \to \mathbb{R}^2$. The points of plane $p$ are bounded by projection $p = (b - x(t)) \to \mathbb{R}^2$, where $b \in D(x(t), v)$. The point of ellipsoidal $p$ is then given as standard ellipse boundary with vertical span $d_\theta(x(t))$ and horizontal span $d_\varphi(x(t))$.

The 2D *Ellipsoid* $E(x(t), v)$ for specific time $t = 10s$ example is portrayed as red ellipsoid (in fig. 6.15).

$$E(x(t), v) = \left\{ \begin{matrix} b \in \mathbb{R}^3 : b \in D(x(t), v), p = (b - x(t)) \to \mathbb{R}^2, \\ \left( \frac{p(1)^2}{d_\theta(x(t))^2} + \frac{p(2)^2}{d_\varphi(x(t))^2} \right) \leq 1 \end{matrix} \right\} \tag{6.108}$$

The expected behaviour of an intruder $i_k$ is to stick to predicted linear trajectory $x(t)$ (6.105). The probability of deviation should be decreasing with distance from ellipse center (fig. 6.16.).

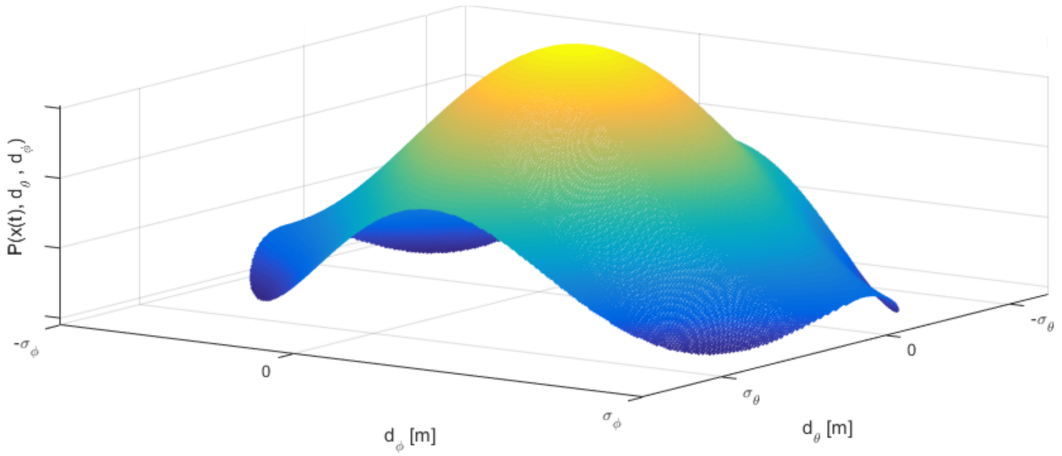



Figure 6.16: Probability of intruder $i_k$ position in ellipsoid $E(x(t), v)$

*Probability density function* for ellipsoid $E(x(t), v)$defined in (eq. 6.108) is depending on maximal horizontal spread $d_\theta(x(t))$, maximal vertical spread $d_\varphi(x(t))$, defined by (eq. 6.107).

Two standard probabilistic distributions are established $\mathcal{N}(\mu_\theta, \sigma_\theta)$ (eq. 6.109) for horizontal spread $\theta(x(t))$ and $\mathcal{N}(\mu_\varphi, \sigma_\varphi)$ (eq. 6.110) for vertical spread $\varphi(x(t))$. The means $\mu_\theta$ and $\mu_\varphi$ are set to zero, and internal coordinate frame $p \in \mathbb{R}^2$ where $x(t) \to \mathbb{R}^2$ is frame center. The variances $\sigma_\theta$ and $\sigma_\varphi$ are set as maximal distances on horizontal/vertical spread axes $d_\theta(x(t))$ and $d_\varphi(x(t))$.

$$P(x(t), d_\theta) = \mathcal{N}(\mu_\theta, \sigma_\theta) = \mathcal{N}(0, d_\theta(x(t))) \tag{6.109}$$

$$P(x(t), d_\varphi) = \mathcal{N}(\mu_\varphi, \sigma_\varphi) = \mathcal{N}(0, d_\varphi(x(t))) \tag{6.110}$$

The combined *probability density function* for maximal spreads $d_\theta$ and $d_\varphi$ is given by (eq. 6.111). Because probability density function is defined for internal space $p \in \mathbb{R}^2$ and one may need to calculate impact rate for cell space $c_{i,j,k} \in \mathbb{R}^3$.

The reduction from two parameter probability distribution function to scalar rate distribution function is needed. An scalar rate distribution function $P(x(t), d_\theta, d_\varphi)$ over ellipsoid $E(x(t), v)$ is defined as (eq.6.111), where final rate is given as average of two partial probabilities.

Final space intersection rate $P(x(t), d_\theta, d_\varphi)$ needs to be normalized to hold *normal distribution condition* (eq. 6.112). Normal distribution condition value (eq. 6.112) is given as surface integral over ellipsoid $E(x(0), v)$ with rate distribution function $P(x(t), d_\theta, d_\varphi)$.

$$P(x(t), d_\theta, d_\varphi) = \frac{\mathcal{N}(\mu_\theta, \sigma_\theta) + \mathcal{N}(\mu_\varphi, \sigma_\varphi)}{2} \tag{6.111}$$

$$\iint_{E(x(\tau))} P(x(t), d_\theta, d_\varphi)\, \mathrm{d}d_\theta\, \mathrm{d}d_\varphi = 1 \tag{6.112}$$

Final space intersection rate $P(x(t), c_{i,j,k}, \theta, \varphi)$ (space portion, time portion is calculated in (eq.6.96) is given by (eq. 6.114). Its mean value of all intersection rates $P(x(\tau), c_{i,j,k}, \theta, \varphi)$ where $\tau \in [i_e(c_{i,j,k}), i_l(c_{i,j,k})]$ is fixed point in intersection time interval.

An $P(x(\tau), c_{i,j,k}, \theta, \varphi)$ (6.113) is integration of rate density function $P(x(\tau), d_\theta, d_\varphi)$ (eq. 6.111) in surface $E(x(\tau), v)$ to cell $c_{i,j,k}$ volume intersection.

To get a volume integration partial rate in surface intersection must be integrated and normalized in time interval $\tau \in [i_e(c_{i,j,k}), i_l(c_{i,j,k})]$, the *base intersection probability* $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ is given by (eq. 6.114). Example of intersection of intruder $i_r$ uncertain ellipsoid cone with avoidance grid $\mathcal{A}(t_i)$ is given in (fig. 6.17).

$$P(x(\tau), c_{i,j,k}, \theta, \varphi) = \iint_{E(x(\tau),v) \cap c_{i,j,k}} P(x(\tau), d_\theta, d_\varphi) \tag{6.113}$$

$$P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \frac{\int_{i_e(c_{i,j,k})}^{i_l(c_{i,j,k})} P(x(\tau), c_{i,j,k}, \theta, \varphi)\, \mathrm{d}\tau}{i_l(c_{i,j,k}) - i_e(c_{i,j,k})} \tag{6.114}$$

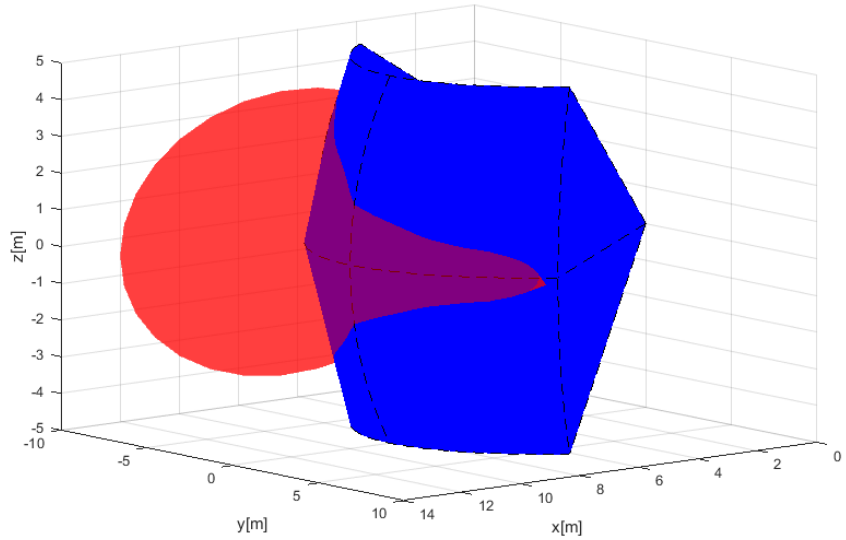



Figure 6.17: Avoidance grid $\mathcal{A}(t_i)$ (blue) intersection with elliptic cone intruder $i_k(x, v, \theta, \varphi)$ (red) example.

An *numeric approximation* of space intersection rate $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ is more implementation feasible than symbolic calculation due the multiple intersection constraints and bad intersection algorithm complexity.

Let us define homogeneous discrete subset of real numbers $\mathcal{R}$ which is non empty subset of real numbers $\mathbb{R}$. The set $\mathcal{R}$ (eq. 6.115) is homogeneous, that means for any equal interval $(i, i+1], i \in \mathbb{Z}$ subset the count of members is equal to some positive natural number $k$. The parameter $k$ can be understand as *unit approximation density*.

Similarly the power sets $\mathcal{R}^2 \subset \mathbb{R}^2$, $\mathcal{R}^3 \subset \mathbb{R}^3$, ... $\mathcal{R}^i \subset \mathbb{R}^i, i \in \mathbb{N}^+$ keeps homogeneous distribution.

$$\mathcal{R} = \left\{ \begin{array}{c} a \in \mathbb{R} : \forall i \in \mathbb{Z}, |i < a \leq i+1| = k, k \in \mathbb{N}^+, \\ \forall j \in \mathbb{N}^+ a_{j+1} - a_j = m, m \in \mathbb{R}^+ \end{array} \right\}, \mathcal{R} \subset \mathbb{R} \tag{6.115}$$

The orthogonal plane for $x(t), v, t \in \mathbb{R}$ is defined by (eq. 6.106). The orthogonality property is also kept for any subspace $\mathcal{R}^n \in \mathbb{R}^n, n \in \mathbb{N}^+$. Numeric approximation of $D(x(t), v)$ is given as $D_D(x(t), v)$ (eq. 6.116).

The only difference is that discrete approximation is countable $|D_D| = m, m \in \mathbb{N}^+$, but continuous representation $|D| \approx \infty$ is uncountable. Because ellipsoid is subset of orthogonal plane it keep its countability property, therefore $E_D$ is also countable and must contains at-least one member.

$$D_D(x(t), v) = \left\{ a \in \Re^3 : (a - x(t)) \perp v, \right\}, t \in \Re \tag{6.116}$$

The *base ellipsoid* $E(x(t), v)$ for continuous-space is given by (eq. 6.108). Every element, expect the base of internal projection $\Re^2$ and orthogonal plane $D_D$ is same in discrete case $E_D(x(t), v)$ (eq. 6.117).

$$\bar{E}_D(x(t), v) = \left\{ \begin{matrix} b \in \Re^3 : b \in D_D(x(t), v), p = (b - x(t)) \rightarrow \Re^2, \\ \left( \frac{p(1)^2}{d_\theta(x(t))^2} + \frac{p(2)^2}{d_\varphi(x(t))^2} \right) \leq 1 \end{matrix} \right\}, t \in \Re \tag{6.117}$$

The *numeric calculation disproportion* can occur in case that ellipsoid $\bar{E}_D(x(t), v)$ (6.117) in case of $d_\theta(x(t)) \approx 0$ and $d_\varphi(x(t)) \approx 0$. The count of ellipsoid members can be $|\bar{E}_D(x(t), v)| = 0$, which is in contradiction with assumption $|\bar{E}_D(x(t), v)| \neq 0$.

Let assume for discrete times $\tau = \{t_1, t_2, \ldots, t_i\}$, $i \in \mathbb{N}^+$ there exists ellipsoids $\bar{E}_D(x(t_1), v)$,$\bar{E}_D(x(t_1), v)$, $\ldots$, $\bar{E}_D(x(t_i), v)$ which are non empty and in space $\Re^2$ in internal coordinate frame and space $\Re^3$ in avoidance grid $\mathcal{A}(t_i)$ coordinate frame. The intersection of these partial ellipsoids in both spaces is equal to:

$$\bar{E}_D(x(t_1), v) \cap \bar{E}_D(x(t_2), v) \cdots \cap \ldots \bar{E}_D(x(t_i), v) = \varnothing \tag{6.118}$$

An *empty intersection* enables us to keep homogeneity property of ellipsoids by adding points so it is safe to add specific point $x(t)$ into empty ellipsoid. But only one, because it does not impact probability density functions $\mathcal{N}(\mu_\theta, \sigma_\theta)$ and $\mathcal{N}(\mu_\varphi, \sigma_\varphi)$, neither space intersection rate density function $P(x, d_\theta, d_\varphi)$.

The final ellipsoid used forward $E_D(x(t), v)$(eq. 6.119) is keeping all properties of ellipsoid $E(x(t), v)$ (eq. 6.119).

$$E_D(x(t), v) = \begin{cases} |\bar{E}_D(x(t), v)| = 0 & : \{x(t)\} \\ |\bar{E}_D(x(t), v)| \geq 0 & : \bar{E}_D(x(t), v) \end{cases} \tag{6.119}$$

The normal distribution condition for rate distribution function $P_D(x(t), d_\theta, d_\varphi, p)$, which is instance of to rate density function $P(x(y), d_\theta, d_\varphi)$ (eq. 6.111) is used. This rate distribution must be normalized according to (eq. 6.120).

$$\sum_{p \in E_D(x(t))} P_D(x(t), d_\theta, d_\varphi, p) = 1, \forall t \in \Re^+ \tag{6.120}$$

The equations for *space intersection rate* are similar to (eq. 6.113, 6.114). For cell $c_{i,j,k}$ there exist intruder entry time $i_e(c_{i,j,k})$ its the earliest intersection with ellipsoid $E_D(x(i_e(c_{i,j,k}))), v$. Same situation occurs with intruder leave time $i_l(c_i, j, k)$. Because $E_D$ is countable set, it means additional attributes can be attached to each point $p \in E_D$. Based on system dynamic (eq. 6.92) the *Time Of Arrival* (TOA) can be calculated. The example of TOA is given in fig. 6.18.

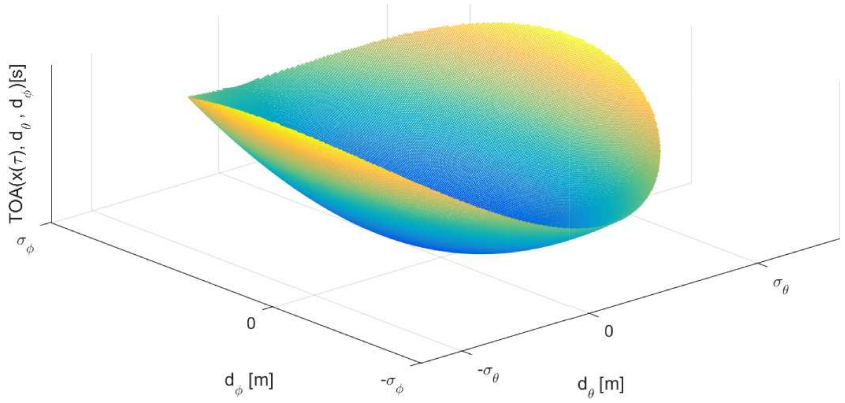



Figure 6.18: Time Of Arrival (TOA) for one ellipsoid $E_D(x(\tau), v)$.

The intersection rate $P_D(x(\tau), c_{i,j,k}, \theta, \varphi)$ for one time sample $\tau$ is given by (eq. 6.121), which has similar notation to (eq. 6.113), sums are used instead of integrals and discrete rate density function $P_D(x(\tau), d_\theta, d_\varphi, p)$ for points form ellipse and cell intersection are used as iterator base set $p \in \{E_D(x(\tau), v) \cap c_{i,j,k}\}$.

$$P_D(x(\tau), c_{i,j,k}, \theta, \varphi) = \sum_{p \in \left\{E_D(x(\tau), v) \cap c_{i,j,k}\right\}} P_D(x(\tau), d_\theta, d_\varphi, p) \tag{6.121}$$

The *space intersection rate* $P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.122) is given as mean intersection rate of partial intersections $P_D(x(\tau), c_{i,j,k}, \theta, \varphi)$ where step set $T = \{ i_e(c_{i,j,k}), \ldots, i_l(c_{i,j,k})\}$ contains all viable intersection times with ellipsoids $E(x(\tau \in T), v)$. The denominator is basically count of samples in sample time set $T$.

$$P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \frac{\sum_{\tau=i_e(c_{i,j,k})}^{i_l(c_{i,j,k})} \sum_{p \in E_D(x(\tau), v)} P_D(x(\tau), c_{i,j,k}, \theta, \varphi, p)}{\sum_{\tau i_l(c_{i,j,k})}^{i_e(c_{i,j,k})} 1} \tag{6.122}$$

An *intersection of intruder cone and cell* $c_{i,j,k}$ cell is defined by (eq. 6.123) The set of point $p \in \mathcal{R}^3$ where condition of intersection between ellipsoids $E_D(x(\tau), v)$ for times $\tau \in \mathcal{R}^+$ and cell space $c_{i,j,k}$ is met.

$$\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \bigcup_{\forall \tau \in \mathcal{R}^+} \left\{p \in \mathcal{R}^3 : p \in c_{i,j,k} \cap E_D(x(\tau), v)\right\} \tag{6.123}$$

An *intruder time of entry* $i_e(i_k, c_{i,j,k})$ (eq. 6.124), for intruder $i, k$ and cell $c_{i,j,k}$ is approximated for discrete point set $\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.123) as minimal time of arrival $t_{TOA}(p)$ of member points $p$.

$$i_e(i_k, c_{i,j,k}) \approx \min \left\{t_{TOA}(p) : p \in \mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})\right\} \tag{6.124}$$

An *intruder time of leave* $i_l(i_k, c_{i,j,k})$ (eq. 6.125), for intruder $i, k$ and cell $c_{i,j,k}$ is approximated for discrete point set $\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.123) as maximal time of arrival $t_{TOA}(p)$ of member points $p$.

$$i_l(i_k, c_{i,j,k}) \approx \max \left\{t_{TOA}(p) : p \in \mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})\right\} \tag{6.125}$$

**Combined intersection model:** The *combined intersection model* $P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau)$ is defined for intruder $i_k$ with parameters:

1. *Starting position* $x_s$ - expected position of intruder $i_r$ in 3D space at time of avoidance $t_i$ in avoidance grid frame $\mathcal{A}(t_i)$.

2. *Velocity vector* $v$ - oriented velocity of intruder $i_r$ at time of avoidance $t_i$ in avoidance grid frame $\mathcal{A}(t_i)$.

3. *Horizontal uncertainty spread* $\theta$ - defines how much can intruder $i_r$ deviate on horizontal axis of intruder local coordinate frame (if X+ is main axis, then Y is horizontal axis in right-hand euclidean coordinate frame), due the properties of intersection definition, the horizontal uncertainty spread can have following values $\theta \in [0, \pi/2]$.

4. *Vertical uncertainty spread* $\varphi$ -defines how much can intruder $i_r$ deviate on vertical axis of intruder local coordinate frame (if X+ is main axis in local right-hand euclidean intruder coordinate frame, then Z is horizontal vertical axis), due the intersection definition, the vertical uncertainty spread can have following values $\varphi \in [0, \pi/2]$.

5. *Body volume radius* $r$ - defines the body volume of intruder in meters and it is having $\mathbb{R}^+$ value.

The *flag vector* $l, b, s, \tau \in \{0, 1\}$ is parametrization of rate calculation: $l$ stands for *lined intersection*, $b$ stands for *body intersection*, $s$ stands for *spread intersection*, $\tau$ stands for *time account*.

The *space intersection for line* $P_L(i_k, c_{i,j,k})$ is defined as $P_T(i_k(x, v), c_{i,j,k})$, where $i_k$ is intruder with properties of initial position $x$, velocity vector $v$ and $c_{i,j,k}$ is target cell. (eq. 6.100).

The *space intersection rate for body volume* $P_B(i_k, c_{i,j,k})$ is defined as $P_T(i_k(x, v, r), c_{i,j,k})$ (eq. 6.104), where intruder $i_r$ has additional property of the intruder body volume radius $r$.

The *space intersection probability for maneuverability uncertainty* $P_S(i_k, c_{i,j,k})$ is defined as $P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.122), where intruder properties $\theta$, $\varphi$ stands for intruder horizontal and vertical uncertainty spread.

The *time intersection rate* $P_{\tau,x}(i_k, c_{i,j,k}) \in [0, 1]$ is defined in (eq. 6.96). This probability has two calculation modes, first is for 1D intersection (line), second is for volume intersection (body volume, spread elliptic cone).

UAS cell entry time $t_e$ and cell leave time $t_l$ time for vehicle in avoidance grid $\mathcal{A}(t_i)$ are given by (eq. 6.93) and (eq. 6.94).

Intruder leave and entry time for 1D intersections is trivial and is omitted in this section. Intruder entry $i_e$ and intruder leave $i_l$ for 3D intersection are given by (eq. 6.124, 6.125).

All partial rates with respective definition references are summarized in (eq. 6.126)

$$\begin{aligned}
P_L(i_k, c_{i,j,k}) &= P_T(i_k(x, v), c_{i,j,k}) && (6.100)\\
P_B(i_k, c_{i,j,k}) &= P_T(i_k(x, v, r), c_{i,j,k}) && (6.104)\\
P_S(i_k, c_{i,j,k}) &= P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) && (6.122)\\
P_{\tau,x}(i_k, c_{i,j,k}) &= \frac{\|[i_e(c_{i,j,k}), i_l(c_{i,j,k})] \cap [t_e, t_l]\|}{\|[t_e, t_l]\|} && (6.96)
\end{aligned} \tag{6.126}$$

With definition of all space and time intersection rates (eq. 6.126) and given flag vector $l, b, s, \tau \in \{0, 1\}$ one can formulate combined intersection rate $P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau)$ (eq. 6.127) for intruder $i_k$ and cell $c_{i,j,k}$. The principle is following: *maximum of selected rates product based on flag vector is final intersection rate of intruder $i_k$ in cell.*

The time-use flag $\tau$ is adding time intersection rate $P_{\tau,x}(i_k, c_{i,j,k})$, where time intersection rate is defined by $x = \{L, B, S\}$ for line, body volume, spread ellipse time intersections ($P_{\tau,L}(i_k, c_{i,j,k}) \neq P_{\tau,B}(i_k, c_{i,j,k}) \neq P_{\tau,B}(i_k, c_{i,j,k})$ for one intruder $i_k$).

$$P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau) = \begin{cases} \tau = 0 & : \max \begin{Bmatrix} P_L(i_k, c_{i,j,k}).l \\ P_B(i_k, c_{i,j,k}).b \\ P_S(i_k, c_{i,j,k}).s \end{Bmatrix} \\ \tau = 1 & : \max \begin{Bmatrix} P_{\tau,L}(i_k, c_{i,j,k}).P_L(i_k, c_{i,j,k}).l \\ P_{\tau,B}(i_k, c_{i,j,k}).P_B(i_k, c_{i,j,k}).b \\ P_{\tau,S}(i_k, c_{i,j,k}).P_S(i_k, c_{i,j,k}).s \end{Bmatrix} \end{cases} \tag{6.127}$$

### 6.6.5 (W) Moving Constraints

**Idea:** The basic ideas is the same as in case *static constraints* (sec. 6.5.3). There is horizontal constraint and altitude constraint outlining the constrained space. The only additional concept is moving of *constraint* on horizontal plane in global coordinate system.

The constraint intersection with *avoidance grid* is done in *fixed decision Time*, for cell in *fixed cell leave time* (eq. 6.94), which means concept from static obstacles can be fully reused.

**Definition:** The *moving constraint definition* (eq. 6.128) covers minimal data scope for moving constraint, assuming linear constraint movement.

The original definition (eq. 6.85) is enhanced with additional parameters to support constraint moving:

1. *Velocity* - velocity vector on 2D horizontal plane.
2. *Detection time* - the time when *constraint* was created/detected, this is the time when *c*enter and boundary points position were valid.

$$\begin{aligned} constraint = \{&position, boundary, \dots \\ &\dots, velocity, detectionTime, \dots \\ &\dots altitude_{start}, altitude_{end}, safetyMargin\} \end{aligned} \tag{6.128}$$

**Cell Intersection:** The *intersection algorithm* follows (eq. 6.89), only shift of the *center and boundary points* is required.

First let us introduce $\Delta time$ (eq. 6.129), which represents difference between the constraint detection time and expected cell leave time (eq. 6.94).

$$\Delta time = UAS_{leave}(cell_{i,j,k}) - detectionTime \tag{6.129}$$

The constraint boundary is shifted to:

$$shiftedBoundary(constraint) = \{newPoint = point + velocity \times \Delta time : \ldots \\ \ldots \forall point \in constraint.boundary\} \quad (6.130)$$

The constraint center is shifted to:

$$shiftedCenter(constraint) = constraint.center + velocity \quad (6.131)$$

*Note.* The $\Delta time$ is calculated separately for each $cell_{i,j,k}$, because $UAS$ is also moving and reaching cells in different times. The *cell leave time* can be calculated in advance after reach set approximation.

**Alternative Intersection Implementation:** The alternative used for intersection selected based on polygon intersection algorithms review [144], the selected algorithm is *Shamos-Hoey* [145].

The implementation was tested on *Storm scenario* (sec. 7.3.4) and it yelds same results.

## 6.7 (R) Avoidance Concept

This section introduces *Platform Independent Avoidance Concept* core functionality (fig. 6.2) modules responsible for *path finding* and *navigation* including *data fusion* interface. The sections are organized like follow:

1. *Data Fusion* (sec. 6.7.1) - implementation details of *input interface* responsible for *processing partial known world data* into final visibility, obstacle, intruder, and, constraints ratings.

2. *Avoidance Grid Run* (sec.6.7.2) (inner avoidance run) - the *best path finding* in one *Avoidance Grid* with *situation assessment* done.

3. *Mission Control Run* (sec . 6.7.3) (outer navigation run) - main navigation and decision making algorithm for *non-cooperative obstacle avoidance*.

4. *Computation Complexity* (sec. 6.7.4) - the *computational feasibility study* and *weak point identification* of our approach.

5. *Safety Margin Calculation* (sec. 6.7.5) - the boundaries of *Safety Margin* and identified *impact factors*.

### 6.7.1 (R) Data fusion

The data fusion interfaces *Sensor Field* and *Information Sources* from *cell/trajectory properties*. The *Data Fusion Function* is outlined in (4.19).

First there will be an outline of *Partial Ratings* commutation. Then these ratings will be discredited into Boolean values as properties of *Avoidance Grid/Trajectory*. Then these Boolean values will be used for further classification of space into *Free(t), Occupied(t), Restricted(t)* and *Uncertain(t)*.

All mentioned ratings are result of *Filtered Sensor Readings* from *Sensor Field* and *Information Sources* with prior processing. This section will focus on *final fuzzy value calculation* and *discretization*.

*Note.* All rating values are in *range:* $[0, 1]$ and they were introduced in previous sections.

**Visibility:** The *sensor reading* of *sensor* if *Sensor field* returns a value of *visibility* for cell space in time of decision $t_i$.

The *visibility* for cell is given in (eq. 6.132) as minimal visibility calculated from all capable sensors in *Sensor Field*.

$$visibility(cell_{i,j,k}) = \min \left\{ \begin{matrix} visibility(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \tag{6.132}$$

The example of *visibility* calculation for *LiDAR* sensor is given in (sec. 6.5.2).

*Note.* Sensor reliability for *visibility* is already accounted prior *data fusion*. If not *weighted average* should be used instead.

**Detected Obstacle:** The *physical obstacles* are detected by *sensors* is *Sensor Field*. Each *sensor* returns *detected obstacle rating* in range $[0, 1]$ reflecting the probability of obstacle occurrence in given cell.

The *maximal value* of *detected obstacle* rating is selected from readings multiplied by *visibility rating* to enforce *visibility bias*.

$$obstacle(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \times \dots \\ \dots \times visibility(cell_{i,j,k}) \quad (6.133)$$

The example of *detected obstacle rating* calculation for *LiDAR* sensor is given in (sec. 6.5.1).

**Map Obstacle:** The *Information Sources* are feeding *Avoidance Grid* with partial information of *Map obstacle rating*. *Map Obstacle Rating* shows the certainty that *charted obstacle* is in given cell. This property is bound to *Information Source* and it has *range* in $[0, 1]$.

The *Map Obstacle Rating* for cell (eq. 6.134) is calculated as product of maximal *Map Obstacle Rating* and *inverse visibility*. This gives *visibility biased* certainty of *Map Obstacle*.

$$map(cell_{i,j,k}) = \max \left\{ \begin{matrix} map(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \times \dots \\ \dots \times (1 - visibility(cell_{i,j,k})) \quad (6.134)$$

The example of *Map Obstacle Rating* calculation is given in (sec. 6.5.2).

**Intruder:** There is a set of *Active Intruders*, each intruder is using its own *parametric intersection model*. This parametric *intersection* model calculates *partial intersection ratings* representing *intersection certainty* ranging in $[0, 1]$. The more *partial intersection rating* is closer to 1 the higher is the probability of aerial collision with that intruder in that cell.

The *geometrical bias* is used for cumulative of multiple intruders, the *intruders are not cooperative*, therefore their occurrence can not be addressed by simple *maximum*. The proposed formula (eq. 6.135) is simply bypassing the intruder rating if there is one intruder. If there is more intruders the geometrical bias is applied.

$$intruder(cell_{i,j,k}) = 1 - \prod_{\forall intruder_i \in Intruders} \left( 1 - intersection \begin{pmatrix} cell_{i,j,k}, \\ intruder_i \end{pmatrix} \right) \quad (6.135)$$

The *intruder intersection models* are outlined in (sec. 6.6.1).

**Constraint:** The *constraints* are coming from various *Information Sources*, the *hierarchical constraint application* is resolved by higher level logic. All *constraints* in this context are considered as *hard*.

The *Constraints rating* (eq. 6.136) is in *range* $[0, 1]$ reflecting certainty of constraint application in cell (usually 1).

$$constraint(cell_{i,j,k}) = \max \left\{ \begin{matrix} constraint(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \quad (6.136)$$

The *Constraint Rating* calculation example for *static* constraints is given in (sec. 6.5.3), the example for *moving* constraints is given in (sec. 6.6.5).

*Note.* Weather is already considered in constraints, the weather is handled as soft/hard static/moving constraints.

**Threat:** The concept of threat is *rating of expected harm* to receive in given segment of space. The threat can be time-bound to *decision time* $t_i$ (time sensitive *intruder intersection models*).

The *harm prioritization* is addressed by higher navigation logic (fig. 6.25). All *sources of harm* are considered as equal. Threat is formalized in *following definition*:

**Definition 32.** *Threat is considered as any source of harm. The threat is* maximal aggregation *of various harm ratings. Our* threat *for specific cell is defined by (eq. 6.137).*

$$threat(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}), map(cell_{i,j,k}), \\ intruder(cell_{i,j,k}), constraint(cell_{i,j,k}) \end{matrix} \right\} \quad (6.137)$$

**Reachibility:** The *Reachibility* for trajectory reflects how safe is *path along*. The *Threat* (def. 32) for each cell has been already assessed. The set of *Passing Cells* is defined in *Trajectory Footprint* (eq. 6.46).

The *Trajectory Reachibility* is given as product of *Threats* along the trajectory (eq. 6.138). The *Trajectory Reachibility* can be calculated for each *trajectory segment* given as $\{movement_1, \ldots, movement_i\} \subset Buffer$ originating from $state_0$.

$$reachibility(Trajectory) = \prod_{PassingCells}^{\forall cell_{i,j,k} \in} (1 - threat(c_{i,j,k})) \quad (6.138)$$

*Note.* The *Reachibility* of *trajectory* segment gives the property of *safety* of route from beginning, until last point of segment. There can be a very unsafe trajectory which is very safe from beginning.

The *Reachibility* of *cell* is given by best trajectory segment passing trough the *given cell*. This is given by property, that every trajectory is originating from root $state_0$, which means that one safe route is sufficient to reach space in cell.

The *Trajectory segment* reachability is sufficient, because the overall performance is not interesting, the *local reachability* is sufficient. The cell reachibility is formally defined in (eq. 6.139).

$$\begin{aligned} reachibility(cell_{i,j,k}) = \max\{&Trajectory.Segment(cell_{i,j,k}).Reachibility : \\ &\forall Trajectory \in PassingTrajectories(celli, j, k)\} \end{aligned} \quad (6.139)$$

*Note.* Function Trajectory.Segment($cell_{i,j,k}$).Reachibility gives same results for any segment in $cell_{i,j,k}$, because (eq. 6.138) accounts each cell *threat* only once.

**Discretization:** The *fault tolerant* implementation needs to implement sharp Boolean values of properties mentioned before. The *fuzzy values* are usually threshold to Boolean equivalent. The *operational standards* for *Manned Aviation* [147] demands the fail rate below $10^{-7}$, because there is no definition for *UAS* the *minimal fail rate* is expected to be at similar level.

The *fuzzy values* $[0, 1]$ are projected to *Boolean* properties of *cell* and *Trajectory* in following manner (tab. 6.5).

| Threshold $= 10^{-7}$ | | | |
|---|---|---|---|
| Visibile | $visibility(cell_{i,j,k})$ | $\geq$ | $(1 - threshold)$ |
| Detected Obstacle | $obstacle(cell_{i,j,k})$ | $\geq$ | $threshold$ |
| Map Obstacle | $map(cell_{i,j,k})$ | $\geq$ | $threshold$ |
| Intruder | $intruder(cell_{i,j,k})$ | $\geq$ | $threshold$ |
| Constraint | $constraint(cell_{i,j,k})$ | $\geq$ | $threshold$ |
| Reachable Trajectory | $reachibility(trajectory)$ | $\geq$ | $(1 - threshold)$ |
| Reachable Cell | $reachibility(cell_{i,j,k})$ | $\geq$ | $(1 - threshold)$ |

Table 6.5: Changing ratings from fuzzy to Boolean parameters.

The high values of *Visibility* (eq. 6.132) and *Reachability* (eq. 6.139, 6.138) are expected. The low *threshold* for *threaths* values is expected. The error margin is solved by *Sensor Fusion* therefore initial *false positive* cases have low rate. The *Detected Obstacle Rate* (eq. 6.133), *Map Obstacle Rate* (eq. 6.134), *Intruder Rate* (eq. 6.135), and *Constraint Rate* (eq. 6.136) thresholds are considered low.

**Space Classification:** The *Data Fusion Function* is outlined in (4.19). This classification is resulting into four distinct space sets.

The *Uncertain* space for decision time $t_i$ is a portion of *Avoidance Grid* which *UAS* can not *read* with *Sensor Field*. The *cells* with $\neg Visible$ property. The *Uncertain* space is given by (eq. 6.140).

$$Uncertain(t_i) = \{cell_{i,j,k} : cell_{i,j,k} \in AvoidanceGrid(t_i), cell_{i,j,k}.\neg Visible\} \tag{6.140}$$

The *Occupied* space for decision time $t_i$ is a set of cell which are classified as *Detected Obstacles*. The *Visibility* is not an issue, due the initial damping in (eq. 6.133). The formal definition is the space portion where it is possible to detect *obstacle bodies* or their portions (eq. 6.141).

$$Occupied(t_i) = \left\{ cell_{i,j,k} : \begin{matrix} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.DetectedObstacle \end{matrix} \right\} \tag{6.141}$$

The *Constrained* space for decision time $t_i$ is *Visible* portion of *Avoidance Grid* where the *Intruder* or *Constraint* is present. The mathematical formulation is given in (eq. 6.142).

$$Constrained(t_i) = \left\{ \begin{matrix} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k} : cell_{i,j,k}.Visible, \\ cell_{i,j,k}.Constraint \vee cell_{i,j,k}.Intruder \end{matrix} \right\} \tag{6.142}$$

The *Free* space is the space which is *Visible* and $\neg Obstacle$, $\neg Intruder$, and, $\neg Constrained$. The mathematical definition is simple set subtractions from *Avoidance Grid* (eq. 6.143).

$$\begin{aligned} Free(t_i) = AvoidanceGrid(t_i) - \ldots \\ \cdots - (Uncertain(t_i) \cup Occupied(t_i) \cup Constrained(t_i)) \end{aligned} \tag{6.143}$$

The *Reachable* space for time $t_i$, used in *Avoidance* because its free and there is a safe trajectory, is given as a set of cells from *Avoidance Grid* which are *Reachable*. The mathematical definition is given in (eq. 6.144).

$$Reachable(t_i) = \left\{ cell_{i,j,k} : \begin{matrix} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Reachable \end{matrix} \right\} \tag{6.144}$$

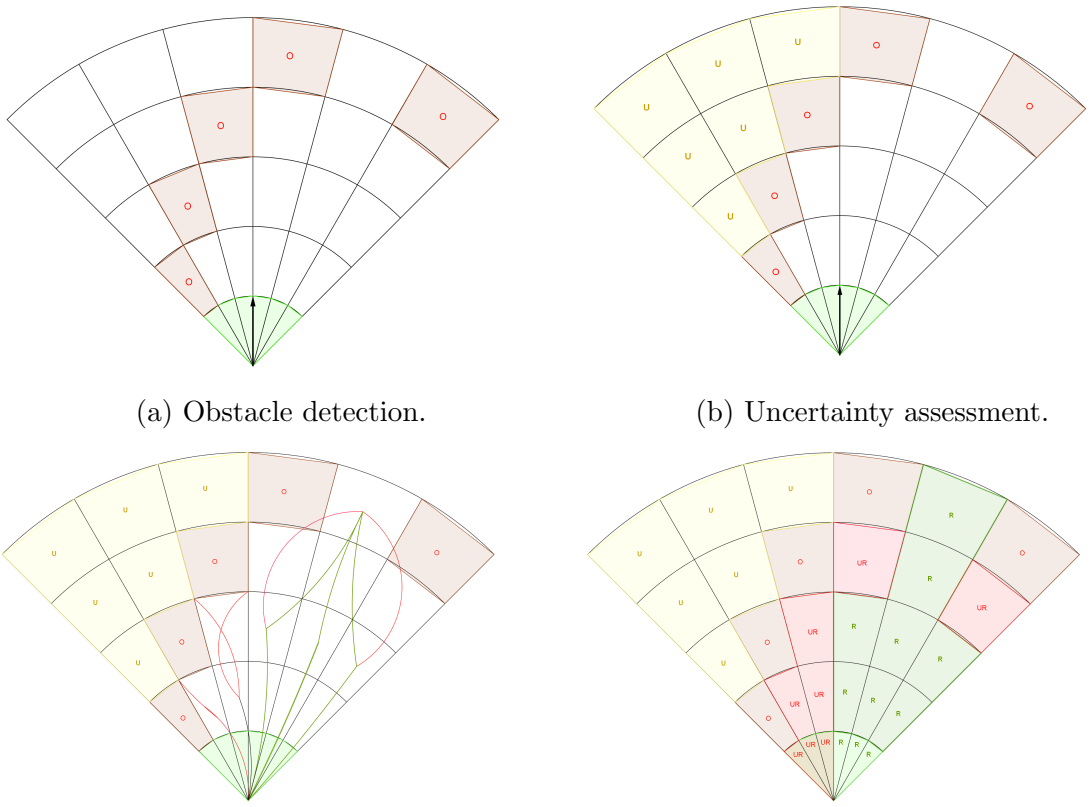
*Note.* The Reachable Space at decision time $t_i$: The *Reachable space* is non-empty set and its a subset of *Free($t_i$)* space:

$$|Reachable(t_i)| > 0, \quad Reachable(t_i) \subset Free(t) \tag{6.145}$$

### 6.7.2 (R) Avoidance Grid Run

**Main Goal:** The main goal of this section is to introduce the trajectory selection process, based on a *situation assessment*, originating from *Data Fusion Procedure* (sec. 6.7.1).

*Note.* The *rating calculation* is outlined in (sec. 6.7.1). Low cost sensor fusion example usable to feed our data fusion procedure is given in [148]. Semi-optimal concatenation trajectory search like ours can be found in [149].



(a) Obstacle detection.

(b) Uncertainty assessment.

(c) Trajectories reachibility evaluation.

(d) Cell reachibility evaluation.

Figure 6.19: Significant steps of *Avoidance grid run* (inner loop).

*Note.* The *Sensor Fusion Procedure* is solving all following steps (sec. 6.7.1). The *main purpose* of *Avoidance Run* is finding best path under certain conditions.

**Space Assessment Principle:** The *Avoidance Grid* is fed trough *Data Fusion* (sec. 6.7.1). The process of *ratings assessment* (tab. 6.5) is given in (fig. 6.19):

1. *Obstacle detection* (fig. 6.19a) - assessment of *detected obstacles* (eq. 6.133). The red (O) *cells* have Detected obstacle set as *true*. The other threats: *map obstacles* (eq. 6.134), *intruders* (eq. 6.135), *constraints* (eq. 6.136) are false. The red (0) *cells* are representing $Occupied(t_i)$ (eq. 6.141) space in *Avoidance Grid* at decision time $t_i$.

2. *Uncertainty assessment* (fig. 6.19b) - the uncertain cells are cells which status can not be *assessed*. The *Visibility* (eq. 6.132) is low. The *Uncertain* cells (yellow (U)

mark) are equal to $Uncertain(t_i)$ (eq. 6.140) in *Avoidance Grid* in *decision time* $t_i$. The $Constrained(t_i)$ (eq. 6.141) space is equal to $\varnothing$ in this example.

3. *Trajectory reachibility evaluation* (fig. 6.19c) - the *Reach Set* given as *Trajectory Set* (eq. 6.42). is then projected trough *Avoidance Grid* and pruned according to (def. 27). *Reachable Trajectories* (eq. 6.138) are only those contained in $Free(t_i)$ space (eq. 6.143). The *Reachable Trajectories* are denoted as *green lines*. The *Unreachable* trajectory segments are denoted as *red lines*.

4. *Cell reachibility evaluation* (fig. 6.19d) - the evaluation of *cells* reachibility is going according to (eq. 6.139). The *Reachable cells* are those which *contains* at least one *Reachable Trajectory Segment*.

**Finding Best Path:** [2] Each $cell_{i,j,k}$ in *Avoidance Grid* at *decision time* $t_i$ has assessed ratings according to *data fusion procedure* (tab. 6.5). The following properties are know prior the *trajectory* selection:

1. *Reachibility* for each $cell_{i,j,k}$ (eq. 6.139).

2. *Reachibility* for each $Trajectory(\circ)$ (eq. 6.138).

3. *Free Space* as non empty set of *cells* in *Avoidance Grid* (eq. 6.143), with *Reachable Space* (eq. 6.144).

4. *Goal Waypoint* $\mathcal{WP}_G$ from *Mission Control Run* (sec. 6.7.3).

The *Algorithm* (alg. 6.6) is based on *shortest path* search. Navigation is trying to reach *goal waypoint*, therefore it tries to shorter distance between *trajectory final cell* and *goal waypoint*. If there is *reachable space* two situations can occur:

1. *Goal waypoint is inside the Avoidance Grid* - the *avoidance cell* is $\text{cell}_{i,j,k}$ containing *goal waypoint* if reachable.

2. *Goal waypoint is outside the Avoidance Grid* - the *avoidance cell* is closest cell considered as *outer cell* to *goal waypoint*.

*Note. Outer cell* is a $\text{cell}_{i,j,k}$ which has at least one *wall* directly neighbouring with *outer space* ($Universe - KnownWorld(t_i)$). The *outer cell* is selected to prevent navigation to the *trap*.

The *Avoidance Path* selection is simple lowest cost selection of $Trajectory \in \text{cell}_{i,j,k}$.

[2] Avoidance Run Function Implementation: `RuleEngine/MissionControl/MissionControl.m::findBestPath(avoidanceGrid)`

**Algorithm 6.6:** Find best *Path* in *Avoidance Grid*

**Input :** Cell[] reachable (eq. 6.144), Waypoint goal, AvoidanceGrid($t_i$) grid
**Output:** Trajectory avoidancePath, Error message

```
# Initialization & Reachibility test;
avoidancePath = ∅;
if reachable == ∅ then
    message = "No path available, empty Reach Set";
    return [avoidancePath,message]
end
avoidanceCell = GetRandomCell(reachable);

# Look for for goal cell;
if goal ∈ grid then
    # Goal is inside Avoidance Grid, Check if reachable;
    avoidanceCell = grid.selectCellXYZ(goal);
    if avoidanceCell.Reachable != true then
        message = "Waypoint not Reachable";
        return [avoidancePath,message]
    end
else
    # Goal is outside Avoidance Grid, look for closest reachable cell_{i,j,k};
    minimalDistance = distance(avoidanceCell,goal);
    for cell_{i,j,k} ∈ reachable do
        if distance(cell_{i,j,k},goal) < minimalDistance then
            if isOuterCell(cell_{i,j,k}) then
                minimalDistance = distance(cell_{i,j,k},goal);
                avoidanceCell = cell_{i,j,k};
            end
        end
    end
end

# Reachable cell was found, Look for cheapest reachable trajectory;
avoidancePath = GetRandomTrajectory(avoidanceCell);
for trajectory ∈ avoidance Cell && trajectory.Reachable == true do
    if trajectory.Cost < avoidancePath.cost then
        avoidancePath = trajectory;
    end
end
message = ∅;
return [avoidancePath,message]
```

**Space Assessment Example:** For better understanding there is following example of *space assessment* and *Best Path Selection*.

The *UAS* (blue plane) is following *mission plan* in open space. Then there is a detection of an *collision situation* (fig. 6.20). The *Obstacle* is detected in *top-right* Avoidance Grid corner.

The *LiDAR hits* are denoted as red filled circles. The *Avoidance Grid* space is constrained by black dashed line. The *Avoidance Grid* is separated into 5 layers going from top to *bottom*. The *Reach Set* is projected as a set of *Trajectories* with colorization.

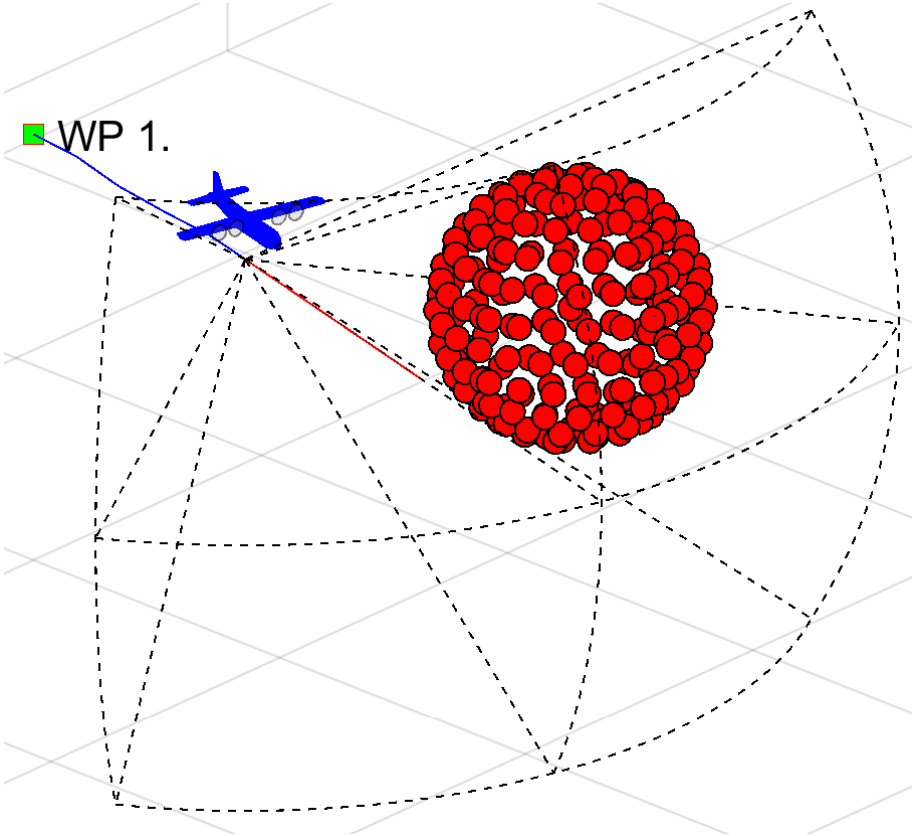



Figure 6.20: Example: The situation to be evaluated by *Avoidance Run*.

*Visibility Assessment:* The visibility assessment (fig. 6.21) divides the *Avoidance Grid* into two

1. *Visible space* (blue filled cells) is space *trough* which *LiDAR* rays roamed freely until they hit an *Obstacle*.

2. *Uncertain space* (black filled cells) is space where no *LiDAR ray* passed nor hit. Therefore its status is uncertain.

*Note.* The *detected obstacle cells* are part of *visible space*, because there is certainty about its containment.

The *Reach Set* trajectories are colored based on their visibility, blue for *uncertain* trajectories and *green* for visible trajectories.

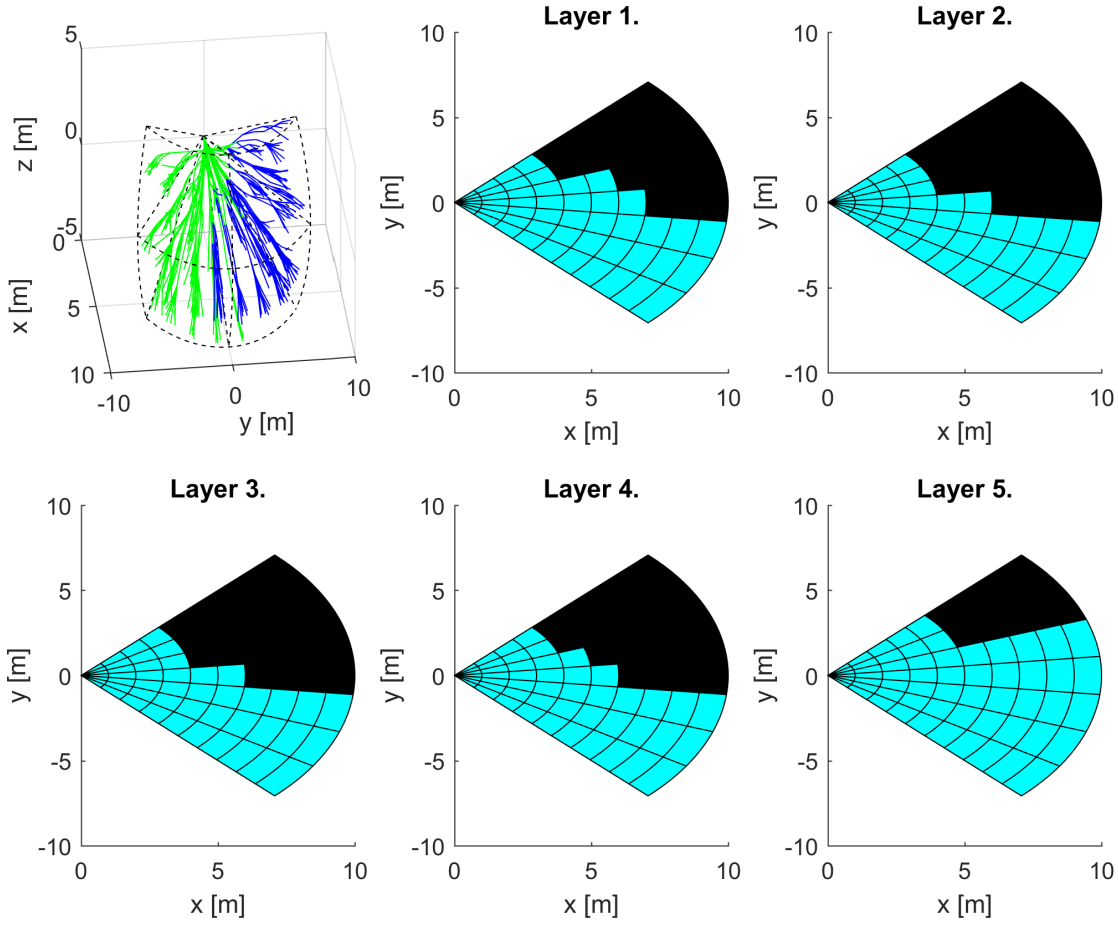



Figure 6.21: Example: The *Visibility* evaluation by *Avoidance Run*.

*Reachibility Assessment:* For Each trajectory the *Reachibility* is assessed (fig. 6.22). The *Obstacle Space* and *Uncertain Space* are rendering *reachibility*, effectively separating *trajectories* into two categories:

1. *Unreachable Trajectories* (red lines) - there is at least one trajectory segment leading trough *Obstacle* or *Uncertain* space.

2. *Reachable Trajectories* (green lines) - all trajectory segments are lying in *Free* space.

Cells in Avoidance grid are divided in similar matter, depending on count of *reachable trajectories* passing trough them:

1. *Unreachable Cells* (red fill) - there is no trajectory trough *free space* or the *cell* is not in *free space*.

2. *Reachable cells* (green fill) - there is at least one *feasible trajectory* reaching *free cell*.

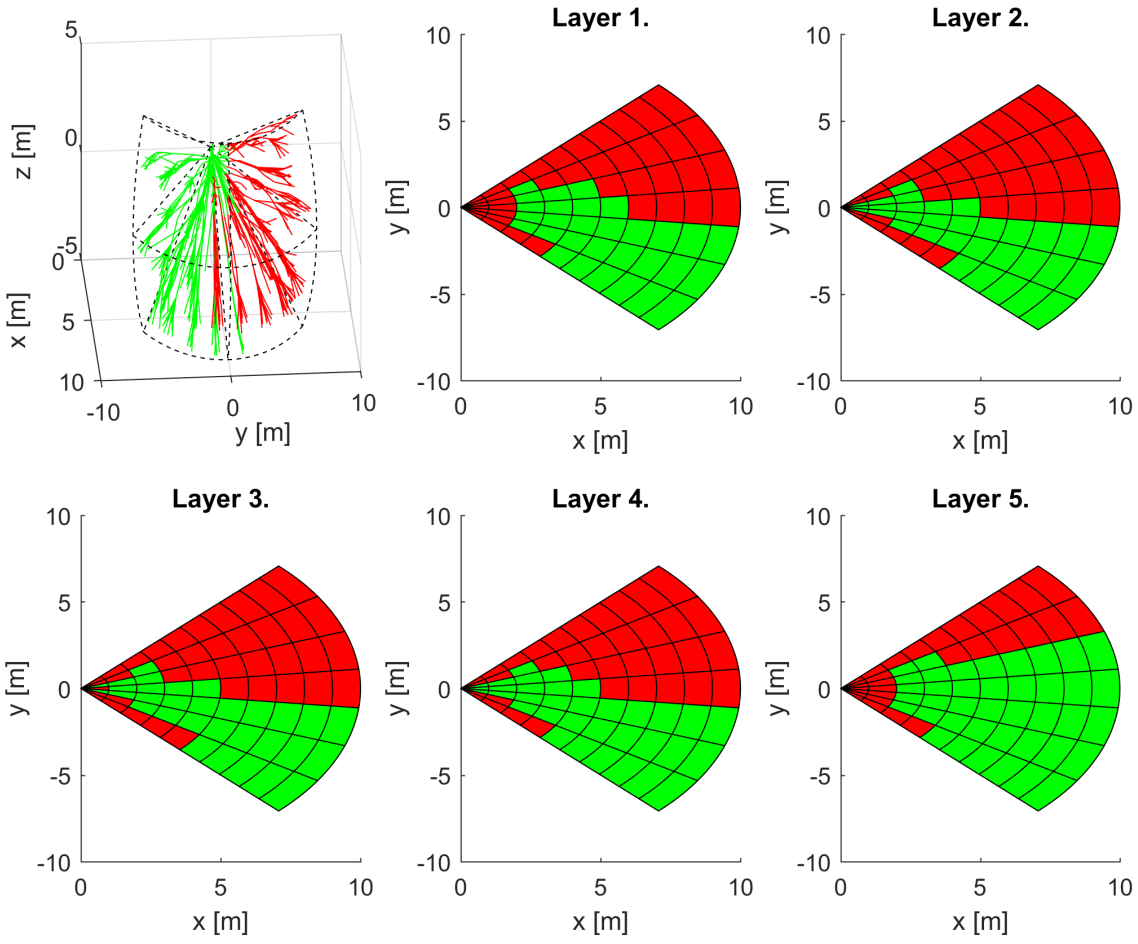



Figure 6.22: Example: The *Reachibility* evaluation by *Avoidance Run*.

*Note.* The *best avoidance path* is selected form *reachable outer cells* (green fill in fig. 6.22), depending on *goal waypoint* according to (alg. 6.6).

### 6.7.3 (R) Mission Control Run

**Introduction and Motivation:** This section will introduce *Navigation Concept* using *Reach Set Approximation*. The *Avoidance Framework Concept* (fig. 6.2) defines *Navigation Module* as *sub-system* for long term *trajectory tracking*. The *Avoidance Grid Run* (sec. 6.7.2) is solving the *Path Search* problem inside operation space constrained by *Avoidance Grid* for time $t_i$.

There is a need to build a trajectory between *Waypoints* which are further away than *distance* of one *Avoidance Grid*. The *UAS* is controlled via *Movement Automaton*. The *Movements* which are in *Movement Buffer* can be replaced with another movements. This feature of *Movement Automaton* is called *Movement Chaining* (eq. 3.22).

To join the multiple *Avoidance Grids* paths following terminology needs to be established (fig. 6.23a):

1. *Goal* (Selecting Goal of Navigation) - the point where UAS want to get in global coordinate frame. The selection needs to be defined.

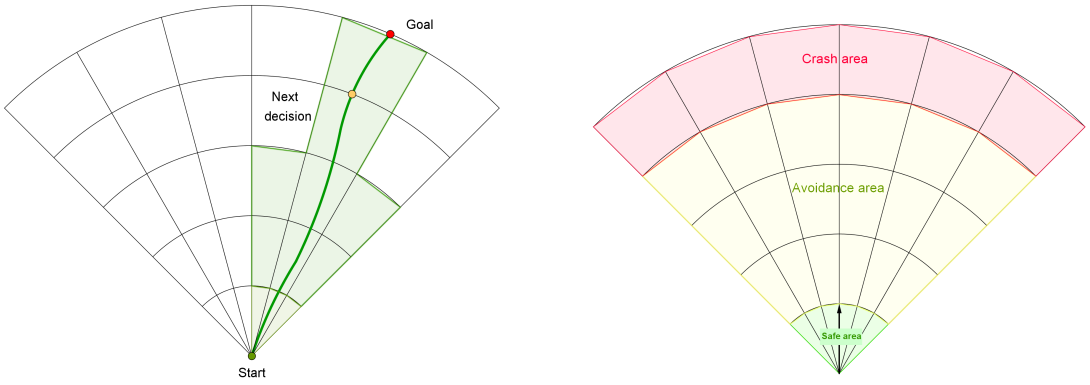
2. *Next Decision* - the point when the next *Avoidance Grid Run* is applied. The outline of events and triggers is required. The *decision* will be made in *next decision time* $t_{i+1}$.

The *Avoidance Grid* from *UAS* viewpoint can be separated into following zones (fig. 6.23b):

1. *Crash Area* (last layers) - there is no place for safe return and the *border* of *Avoidance Grid* is near. The *Decision Point* needs to lie before this zone.

2. *Avoidance Area* (middle layers) - the area of *Active Avoidance Maneuvering*. The *Reach Set Approximation* performance (sec. 6.4.1) is important in this area.

3. *Safe Zone* (first layers) - there is space for safe return or damage mitigation.




(a) Mission control run example. (b) Grid Zones.

Figure 6.23: Definitions for *Mission Control Run* (outer loop).

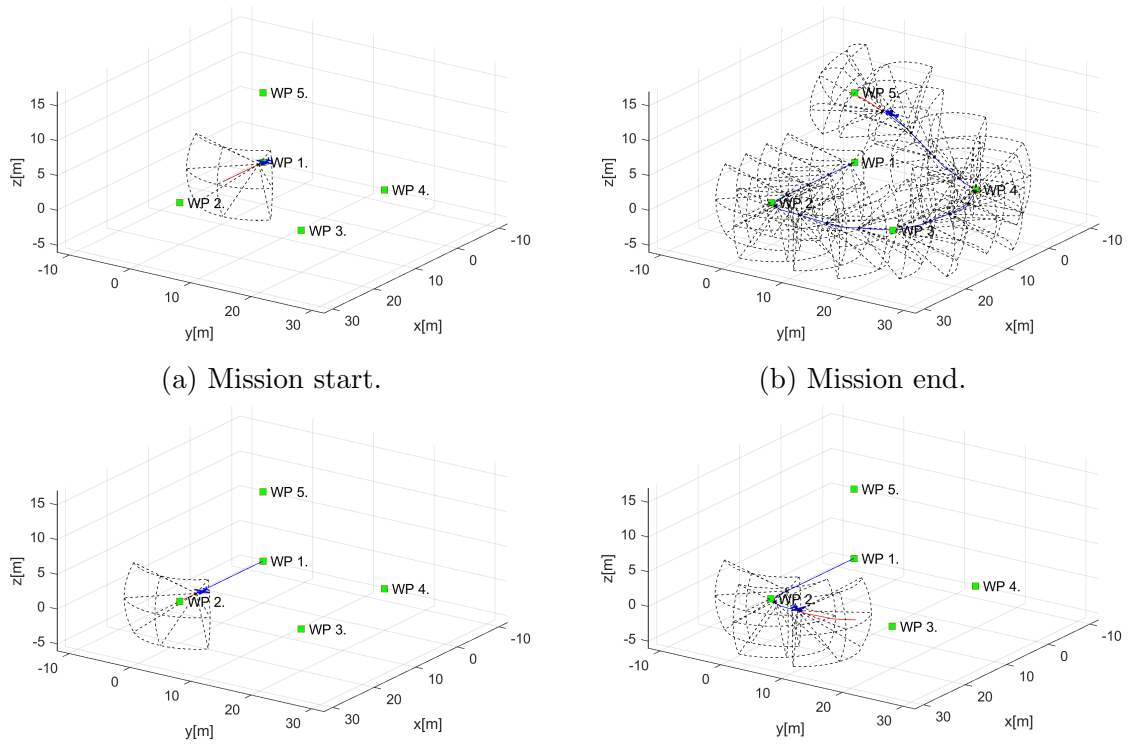
Joining *Avoidance Grid Runs* (fig. 6.24) example portrays *Avoidance Grid Runs* invoked on various *Decision Points* to achieve *Navigation* functionality. The UAS (blue plane) is flying Mission (green numbered waypoints). The *Avoidance Grid* boundary (black dashed line) for each *Decision Point* (UAS position at time $t_i$). Following example of *Navigation* (fig. 6.25) run is shown:

1. *Mission Start* (fig. 6.24a) - UAS at the start of the mission have one *Avoidance Grid* at its position to determine the *Navigation Path* to *Waypoint 2* (goal waypoint). The planned path (red line) is leading directly to *Avoidance Grid* boundary (black dashed line).

2. *Mission End* (fig. 6.24b) - UAS have reached *last waypoint*. All *Avoidance Grid* boundaries (black dashed line) for all *runs* are drawn along flown trajectory.

3. *Waypoint Reach* (fig. 6.24c) - the *waypoint* is inside *Avoidance Grid*, the navigation path (red line) leads directly to *goal waypoint*. (Excessive *Avoidance Grid* boundaries are removed.)

4. *Next Waypoint* (fig. 6.24d) - the new *Goal Waypoint* is selected, the UAS moves to new goal (invoking *Avoidance Grid Runs* when necessary).




(a) Mission start.

(b) Mission end.

(c) Waypoint reach.

(d) Next waypoint.

Figure 6.24: Joining multiple *Avoidance Grid Runs* for achieve Navigation.

**General Concept:** [3] The *General Concept* is taken from [150, 151], consisting from following main modules:

1. *Navigation Loop* - module responsible for *Navigation* providing *Goal Waypoint*.
2. *Data Fusion* (background in sec. 6.7.1) - module responsible for *Surveillance Data Feed*.
3. *Situation Assessment* - module responsible for *UAS Safety Evaluation*.
4. *Avoidance Run* (background in sec. 6.7.2) responsible for *Avoidance Path* selection.

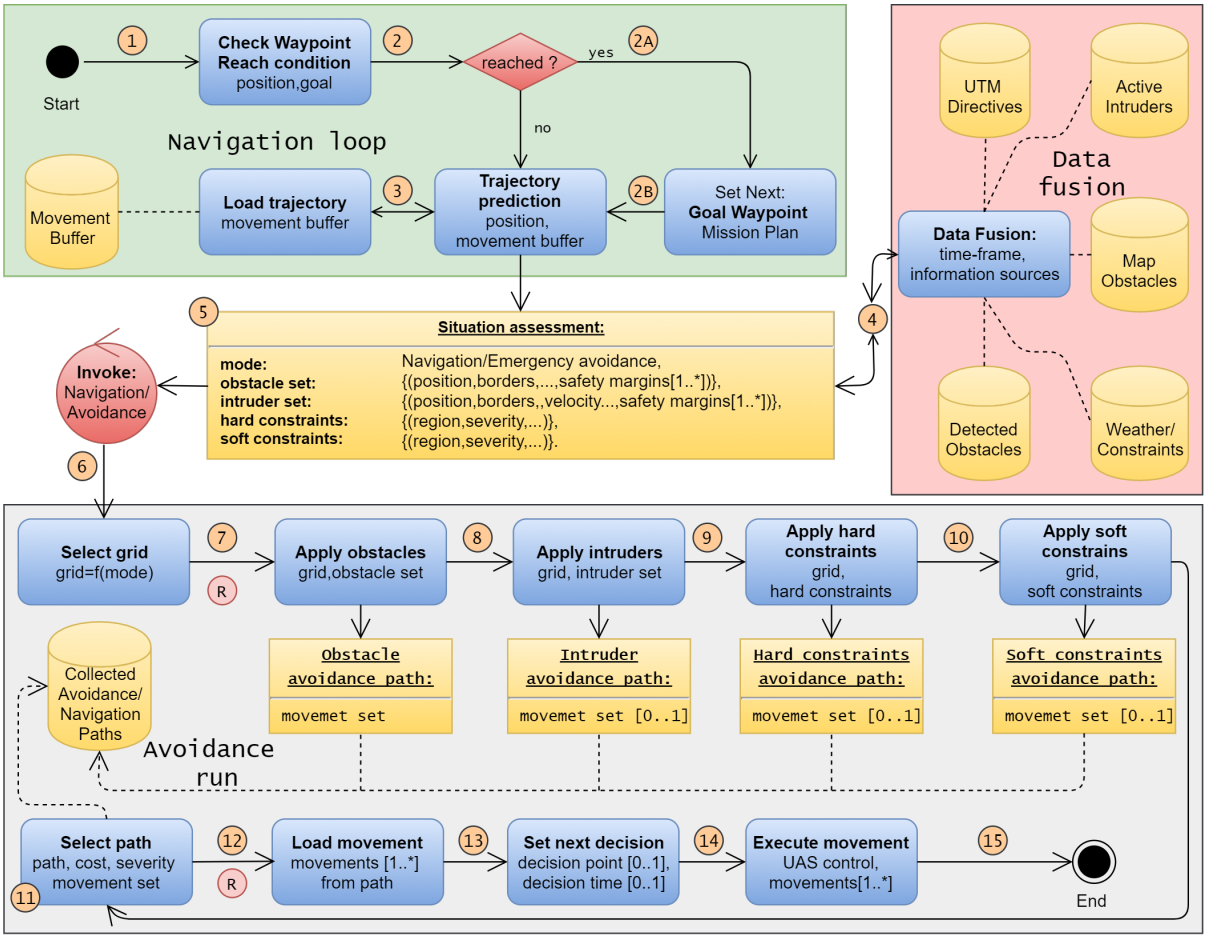



Figure 6.25: Mission control run activity diagram.

The main changes to *Navigation architecture* are given in *Mission Control Run* activity diagram (fig. 6.25):

1. *Situation Assessment* - added event-based mode switching control.
2. *Avoidance Run* - added hierarchical evaluation for *Avoidance Path* selection. Prioritizing threat avoidance according to a type.

[3] Mission Control Run Function Implementation: `RuleEngine/MissionControl/MissionControl.m::runOnce(.)`

The *Operation Mode* is introduced, based on *Situation assessment* and *Triggering Events* one of following modes are selected in *Avoidance Run*:

1. *Navigation Mode* - the *UAS* is navigating trough *Airspace* following *cost effective patterns* and obeying *Airspace Authority* (UTM). The *Navigation Grid* is a instance of *Avoidance Grid* (sec. 6.3) with initialized *Navigation Reach Set* (ex. *Harmonic Reach Set Approximation* (sec. 6.4.4)).

2. *Emergency Avoidance Mode* - the *UAS* is *threatened* by obstacle, intruder, hard constraint or *soft constraint*, the *UAS* is navigating trough *Airspace* following *safe avoidance patterns* and *minimizing the impact* of possible damages. The *Avoidance Grid* is term used for *Emergency Avoidance Mode*. The *Avoidance Reach Set Approximation* is initialized in *Avoidance Grid* (ex. *Chaotic Reach Set Approximation* (sec. 6.4.3))

*Note.* Depending on *Operation Mode* the pair of *Avoidance Grid* and *Reach Set* is selected in *Avoidance Run* part.

The *Navigation Grid* and *Avoidance Grid* shares the space segmentation pattern, therefore the *Data Fusion* (sec. 6.7.1) needs to be evaluated only once for both grids.

**Decision Time Frame ($[t_i, t_{i+1}[$):** The *Mission Control Run* is executed for *Decision Time Frame* bounded to the *period* of the *UAS executed movement* (fig. 6.2).

The *UAS System* (sec. 6.2.1) controlled by *Movement Automaton Implementation* (sec. 6.2.2) *Planned Movements* can be changed at any time. The real impact on control is shown after the *actual movement* is executed.

*Note.* For our *Movement Automaton Implementation* movements the average *movement duration* is *1/velocity second* (tab. 6.1, 6.2).

The *Decisions* are made based on *system* state in *current* time-frame started at $t_i$ for *next* time frame starting at $t_{i+1}$.

*Note.* Because the *Decision Delay* is crucial in *Avoidance System* it is beneficial to have *short time movements*. On the other hands, the *length and duration of movements* is impacting *Reach Set Complexity*. The proper construction of movement automaton is greatly impacting overall *approach performance*.

**Initialization:** The *UAS* is going to solve a problem for *Rules of the Air* (eq. 4.27). Using control scheme (fig. 6.2) with given *Sensors*:

$$Sensors = \{LiDAR, ADS - B\} \tag{6.146}$$

The sensors obstacle assessment into avoidance grid is outlined for static obstacles in (sec. 6.5) and for moving obstacles in (sec. 6.6.)

The *Data Fusion Procedure* is given as follow:

$$DataFusion = \{RatingBasedDataFusion \quad (sec.6.7.1)\} \tag{6.147}$$

Then the *UAS system* (sec. 6.2.1) with *Movement Automaton Implementation* (sec. 6.2.2) with empty movement buffer:

$$MovementBuffer = \{\} \tag{6.148}$$

The *Avoidance Grids* for both *Operation Modes* are created with *identical space segmentation*. The *Reach Set Approximations* are loaded based on initial *UAS State* at decision time 0. The *Reach Set Approximation* is always selected based on *UAS System State*. The initial *Operation Mode* is set up as *Navigation*. The initialization is summarized like follow:

$$\begin{aligned} AvoidanceGrid(0) &= \{UAS.position(0), AvoidanceReachSet(UAS.ReachSet)\} \\ NavigationGrid(0) &= \{UAS.position(0), NavigationReachSet(UAS.ReachSet)\} \\ OperationMode &= Navigation \end{aligned} \tag{6.149}$$

The *Mission* is set up as a set of *ordered waypoints*. The *initial goal waypoint* is *first waypoint*. The initialization is summarized like follow:

$$\begin{aligned} Mission &= \{Waypoint_1 \dots Waypoint_n\} \\ GoalWaypoint &= Mission.waypoint_1 \\ LastWaypoint &= Mission.waypoint_n \end{aligned} \tag{6.150}$$

The *actual threats* are set as empty sets for *decision time* $t_i = 0$:

$$obstacles = \{\}, intruders = \{\}, hardConstraints = \{\}, softConstraints = \{\} \tag{6.151}$$

**Navigation Loop (1st-3rd step):** The purpose of *Navigation Loop* is to select proper *Goal Waypoint* from *Mission* (sec. 4.1.2). If *last waypoint* have been reached the *Landing Procedure* will be initiated and *Mission Control Run* Ends.

First start with definition of *waypoint reach condition* (def. 33) and *Unreachable waypoint* (def. 34).

**Definition 33.** *Waypoint Reach Condition for* current *decision time* $t_i$ *for* UAS *position and current* Goal Waypoint *is satisfied only if:*

$$\begin{aligned} &distance(UAS.position(t_i), GoalWaypoint(t_i)) \\ &\qquad \leq \\ &2 \times \max\{length(movement) : \forall movement \in MovementSet\} \end{aligned} \tag{6.152}$$

*Note.* The movements in our solution have *uniform length* of *1 m* (tab. 6.1, 6.2), therefore the waypoint reach condition is satisfied when *distance to goal waypoint* is lesser than 2 m. The maximal movement length has impact on *navigation/avoidance* precision.

**Definition 34.** *Unreachable Waypoint. The* Goal Waypoint *is evaluated as unreachable in decision time* $t_i$ *when* Avoidance Grid Run *(alg. 6.6) can not find the* navigation/avoidance path *leading to it.*
*Formally: The* Avoidance/Navigation Grid *has range defined as* final layer distance. *When the* Goal Waypoint *is in* range *of* Grid*:*

$$Grid(t_i).range \geq distance(UAS.position(t_i), GoalWaypoint(t_i)) \tag{6.153}$$

*and following condition is satisfied:*

$$\forall cell_{i,j,k} \in Grid(t_i) \; \nexists cell_{i,j,k}.Reachable == true \wedge \ldots \\ \cdots \wedge distance(cell_{i,j,k}, GoalWaypoint(t_i)) \leq \ldots \\ \cdots \leq 2 \times \max\{length(movement) : \forall movement \in MovementSet\} \quad (6.154)$$

*The* Goal Waypoint *is unreachable.*

Then the *Navigation Loop* is invoked every *decision time* $t_i$, *Mission Control Run* (fig. 6.25), it is described as sequence of following steps:

**1st Check Waypoint Reach Condition** - the *UAS position* for given *time frame* $t_i$ is checked under condition (eq. 6.152). If condition is met continue with 2nd step otherwise continue with 3rd step.

**2nd Set Next Waypoint** - until following condition is met:

$$GoalWaypoint == LastWaypoint$$

Set next goal waypoint like follow:

$$GoalWaypoint = Mission.getNextWaypoint()$$

Otherwise enforce *Landing sequence* (Out of Scope).

**3rd Trajectory Prediction** - the *Movement Buffer* is loaded with planned movements from *Movement Automaton*. The *future trajectory* is predicted according to (eq. 6.13):

$$PredictedTrajectory = \\ Trajectory(state = UAS.state(t_i), buffer = futureMovements)$$

The *Predicted Trajectory* is used in 5th step *Situation Assessment*.

**Data Fusion (4th step)** The *Data Fusion* (sec. 6.7.1) in this context is *Threat Sets* preparation for *Avoidance Run*. It is depending on values of *Boolean values* defined in (tab. 6.5) for *threat* classification.

*Note.* Avoidance Grid's Data fusion (sec. 6.7.1) is run in 7th- 10th step (fig. 6.25).

The *static obstacles* source is from *LiDAR* scan received at least at beginning of current *decision frame* $t_i$:

$$obstacles = LiDAR.scan(UAS.position(t_i))$$

The *intruders* source are valid *active intruders notifications* received from ADS-B In positioned to *future expected positions* at *decision time* $t_{i+1}$:

$$intruders = ADS - B.getActiveIntruders(t_{i+1})$$

*Note.* The *Intruders* needs to be predicted for the next decision time-frame starting at time $t_{i+1}$ Due their mobility.

The *hard/soft constraints* are obtained from *Information Sources* and the area of next decision time $t_{i+1}$ *Avoidance Frame* is used as space parameter in search. The sets of hard and soft constraints are obtained in following manner:

$$hardConstraints = InformationSources.fuse(AvoidanceGrid(t_{i+1}))$$

$$softConstraints = InformationSources.fuse(AvoidanceGrid(t_{i+1}))$$

The results of *Data Fusion* threats set preparation are used in next step.

**Invoke Navigation/Avoidance based on Situation Assessment ($5^{th}$-$6^{th}$ step):** The *deciding events* depending on *Trajectory Prediction* ($3^{rd}$ step) and *Data Fusion* ($4^{th}$ step) (fig. 6.25) are following:

1. *General Events* are *triggered* regardless *Operation Mode*. They are considered after *specific mode events* are handled and *Navigation/Avoidance Grid* is selected:
   a. *Empty Movement Buffer* ($MovementBuffer = \varnothing$) - if there is no movement in *Movement buffer* to be executed (from 3rd step: Load Trajectory), the *Avoidance Run* is enforced to run with *Navigation/Avoidance Reach Set Approximation* to generate new path.
   b. *Waypoint Reached* (2nd step) - the *Navigation Loop* run is forced to set goal *Goal Waypoint*. If *last waypoint* from *Mission* (sec. 4.1.2) the *Landing Procedure* is enforced.
   c. *Waypoint Unreachable* - this type of event is very situations based. The *Waypoint Reachibility* (assumption. 4) has not been relaxed, therefore this event is not properly handled in approach. The *implementation* considers *selecting next waypoint in mission* as a goal waypoint of *first waypoint* if *unreached/unreachable waypoints* are exhausted.
2. *Navigation Mode Events* are triggered if *Operation Mode* is set as *Navigation*:
   a. *Empty Navigation Grid* ($|threats| = 0$) - if *movement buffer* contains at least one *movement*, the *Avoidance Run* is omitted. The *Operation Mode* stays in *Navigation Mode*.
   b. *Collision Case Resolution* ($|ActiveCollisionCases| > 0$) - there is new/active *Collision Case* (sec. 6.8.8), the *Navigation Reach Set Approximation* trajectories will be constrained according to active *Collision Case(s)* requirements. If there exists at least one *Reachable* avoidance path, the *Operation Mode* will remain *Navigation*. If there is no *Reachable* avoidance path, the *Operation Mode* switches to *Emergency Avoidance*.
   c. *Static Obstacle Detection* ($LiDAR.Hits > threshold$) - if *static obstacle set* contains at least one *detected obstacle* (sec. 6.5.1) intersecting with *Navigation grid* the *Operation Mode* will be *switched* to *Emergency Avoidance Mode*.
   d. *Intruder Detection* ($intruders > 0$) - if *active intruders set* contains at least one *intruder* which expected impact area (intersection models (sec. 6.6.1)) *Navigation grid* the *Operation Mode* will be *switched* to *Emergency Avoidance Mode*.

e. *Hard or Soft Constraint Occurrence* ($|hardConstraints| > 0 \vee |softConstraints| > 0$) - if *hard/soft constraint set* contains at least one *constraints* which intersects (static constraints (sec. 6.5.3), moving constraints (sec. 6.6.5)) *Navigation grid* the *Operation Mode* will be *switched* to *Emergency Avoidance Mode*.

3. *Emergency Avoidance Events* are triggered if *Operation Mode* is set as *Emergency Avoidance*:

   a. *Empty Avoidance Grid* ($|threats| = 0$) - if there is no *detectable* threat, the remainder of *avoidance path* is removed from *Movement Buffer*. The *Operation Mode* is switched to *Navigation* and new *navigation path* is selected.

**5th Situation Assessment** - if there is any flag raised by *Event Triggers*, there is an *avoidance situation*.

The *Event Triggers* describe complex *Operation Mode* switching. The simplified principle is following: *If UAS is in Emergency Avoidance Mode Always Invoke Avoidance Run. If UAS is in Navigation Mode Invoke Only if Necessary.*

If there was event trigger continue with 7th step, otherwise wait for *next decision time* $t_{i+1}$, execute movement and continue with 1st step.

**6th Invoke Navigation/Avoidance** depending on the *Operation Mode* the *Reach Set/Grid* pair is selected. The future $state(t_{i+1})$ in next decision frame $t_{i+1}$ is necessary for Grid/Reach Set initialization. The *next decision frame initial state* is obtained by *prediction*:

$$state(t_{i+1}) = Trajectory(state(t_i), currentMovement)$$

The *Reach Set Approximation* is loaded based on *mode* and $state(t_{i+1})$. The *Grid* is initialized as $Free(t_{i+1})$ (eq. 6.143) for all cells.

**Avoidance Run (7th-15th step):** The *Avoidance Run* goal is to obtain *Path* represented as *Trajectory*(state($t_{+1}$,MovementBuffer)) (eq. 6.13) from *Navigation/Avoidance Grid* and associated *Navigation/Avoidance Reach Set Approximation*.

If the *Operation Mode* is set as *Navigation Mode* the algorithm continues with 11th step. Otherwise the *Avoidance Grid Space Assessment* is run multiple times to obtain $Reachable(t_{i+1})$ (eq. 6.144). The *Threat Data* obtained from 4th step are used.

**7th Apply Obstacles** - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated with following threat modification:

$$intruders = \varnothing, softConstraints = \varnothing, hardConstraints = \varnothing$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Obstacle Avoidance Path*.

**8th Apply Intruders** - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated with following threat modification:

$$softConstraints = \varnothing, hardConstraints = \varnothing$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Intruders Avoidance Path*.

**9th** **Apply Hard Constraints** - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated with following threat modification:

$$hardConstraints = \varnothing$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Hard Constraint Avoidance Path*.

**10th** **Apply Soft Constraints** - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated without any modification.

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Soft Constraints Avoidance Path*.

*Note.* The 7th to 10th steps are code-optimized for efficient calculation.

**11th** **Select Path** - based on *Operation Mode* the *Navigation/Avoidance Path* is selected.

The *Navigation Path* for *Navigation Mode* is selected by standard *Find Best Path* (alg. 6.6) procedure. The *Navigation Reach Set Approximation* can be constrained by *Rule Engine* (fig. 6.33).

The *Avoidance Path* for *Emergency Avoidance Mode* is selected from *Collected Avoidance Paths* with following priority:

1. *Soft Constraints Avoidance Path* - if exists continue with 12th step, if does not exist try to select:
2. *Hard Constraints Avoidance Path* - if exists continue with 12th step, if does not exist try to select:
3. *Intruders Avoidance Path* - if exists continue with 12th step, if does not exist try to select:
4. *Obstacle Avoidance Path* - continue with 12th step.

*Note.* The *Waypoint Reachibility* (assumption 4) is weakened to the point that it is necessary for waypoint to be *Reachable* only in static obstacle environment. The *Constrained* and *Occupied* spaces are shrunk in following matter to increase UAS survival chances. There are following relaxations with their conditions:

1. *Soft Constraint Relaxation* - they are breakable by default. This kind of situation is allowed to happen under any circumstances.
2. *Hard Constraints Relaxation* - they can be broken in case of emergency (airspace constraints) or UAS robust build (Weather Constraints). This kind of situation is allowed under very specific conditions depending on *broken constraint* severity.
3. *Intruder Occupied Space Relaxation* - this can be broken if and only if there is guarantee the Intruder dynamic and navigation algorithm allows to avoid *Collision* with UAS. This relaxation should be used as *the last resort*.

**12th Load Movements** - the *Movement Buffer* is flushed for *future decision times* $t_{i+1}, \ldots, t_{i+k}$. The *Navigation/Avoidance Path* movements are pushed into *Movement Buffer* instead. The *executed movement* for *decision time* $t_i$ remains (because its executed at this time point).

**13th Set Next Decision** - the *next decision point* is set depending on circumstances:

1. Navigation Mode (no active collision cases) - *Decision Point* is set as point before *UAS* enters into *Crash Zone* (fig. 6.23b) in *Navigation Grid*.
2. *Navigation Mode (at least one active collision case)* - *Decision Point* is set after *next movement execution*. Current decision point $UAS.Position(t_i)$, next decision point $UAS.Position(t_{i+1})$.
3. *Emergency Avoidance Mode (any circumstances)* - *Decision Point* is set after *next movement execution*. Current decision point $UAS.Position(t_i)$, next decision point $UAS.Position(t_{i+1})$.

**14th Execute Movement** - the *First Movement* from *Movement Buffer* is loaded to be executed in decision time frame $[t_{i+1}, t_{i+2}[$.

**15th Finish Avoidance Run** - if the *UAS* is flying, continue with 1st step.

### 6.7.4 (R) Computation Complexity

**Introduction:** The *Computation Complexity* one mission control run assessment is necessary to identify the strong and weak points of approach. Lets get trough modules to assess notable calculations/algorithms complexity on high abstraction level.

**Navigation Loop:** I the navigation loop, the *waypoint reach condition* (eq. 6.152) is checked, this is unitary operation with worst complexity $\mathcal{O}(1)$. The selection process of the next *Goal Waypoint* can get trough all waypoints in the mission if they are all unreachable the complexity is $\mathcal{O}(|waypoints|)$.

The *notable steps* complexity is following:

$$\begin{aligned} \texttt{Reach Condition:} &\quad \mathcal{O}(1) \\ \texttt{Select Next Waypoint:} &\quad \mathcal{O}(|waypoints|) \end{aligned}$$

**Data Fusion:** The *data fusion* is all about *threat selection*.

If *UAS* is in *controlled airspace* it needs to iterate over received *collision Cases* to select *active ones*. The complexity of this step is linear, therefore boundary is given as $\mathcal{O}(|collisionCases|)$.

Thresholding *Detected Obstacles* is done by simple comparison of *LiDAR ray hits* in given $cell_{i,j,k}$ of *Avoidance Grid*.

Any loading of *threats* from *information sources* is depending on clustering. The *Airspace Clustering* is considered as static for our setup. Therefore the *count of active airspace clusters* has main impact on complexity. The *count of information sources* is static and not changing over mission time. Information sources usually implement *Hash search function* with complexity $\mathcal{O} \ln |searchedItemSet|$.

The *computation complexity* boundaries for *Data fusion* in our setup are following:

$$\begin{aligned} \texttt{Select Active Collision Cases:} &\quad \mathcal{O}(|collisionCases|) \\ \texttt{Threshold Detected Obstacles:} &\quad \mathcal{O}(|cells|) \\ \texttt{Load Map Obstacles:} &\quad \mathcal{O}(\ln |activeClusters| \times |informationSources|) \\ \texttt{Load Hard Constraints:} &\quad \mathcal{O}(\ln |activeClusters| \times |informationSources|) \\ \texttt{Load Soft Constraints:} &\quad \mathcal{O}(\ln |activeClusters| \times |informationSources|) \end{aligned}$$

*Note.* The *real-time clustering* is *hard non-polynomial problem* [152]. Usually all information sources and sensor have *polynomial complexity* of processing. The *controlled airspace clusters* are usually set for very long period of time. Therefore *Obstacle Map*, *Airspace Constraints*, and, *Weather Constraints* can be considered as preprocessed

**Situation Assessment:** The *Situation Assessment* is evaluating triggering events. The *evaluation* is usually simple existence question without further calculations. The *complexity* of *event evaluation* for our case is $\mathcal{O}(1)$. There are 8 triggers. The count of *triggers* needs to be accounted in complexity boundary:

$$\mathcal{O}(|triggers| \times eventEvaluationComplexity)$$

*Note.* The *trigger calculation complexity* needs to stay low, because the *triggers* are verified every *Mission Control Run*. The *Avoidance Run* trigger frequency should be very low under normal conditions.

**Avoidance Run:** The *Avoidance run* is most critical part of *Mission Control Run*, because *Avoidance Path* calculation. The *Navigation Path* calculation is less complex (Rule engine is not accounted), therefore *Emergency Avoidance Mode* is assumed.

The *threat insertion* is realized in $7^{\text{th}}$ to $10^{\text{th}}$ step. The first is *Avoidance Grid* filled with *Static Obstacles*. The *Avoidance Grid* is designed to separate rotary *LiDAR* ray space into hit count even cells. Insertion of *LiDAR* scan into *Avoidance Grid* complexity depends on *total cell count*. The *upper boundary* for *insert obstacles* is given like follow:

$$\texttt{Insert Obstacles:} \quad \mathcal{O}(|cells|)$$

The *intruders intersection model* type impact the insertion complexity. The *linear intersection* (sec. 6.6.2) is going trough maximum of *layers count* cells.

The *body volume intersection model* (sec. 6.6.3) can check the *simple intersection condition* over all *Avoidance Grid* in worst case, therefore complexity for this check is bounded by *count of cells*.

The *Maneuverability Uncertainty Intersection* (sec. 6.6.4) can hit all cells in *Avoidance Grid*. The calculation complexity boundary is exponential depending on *horizontal/vertical* spread in $[rad]$. The *intersection* implementation was done *ad-hoc*. The impact of *intersection application* is visible only when there is more than *4* concurrence intruders (fig. 7.28).

The *complexity boundary for* intruder insertion is given like follow:

$$\texttt{Insert Intruders:} \quad \mathcal{O}\left(\sum \begin{bmatrix} |linearIntersections| \times |layers| \\ |bodyvolumeIntersections| \times |cells| \\ |cells|^{horizontalSpread \times verticalSpread} \end{bmatrix}\right)$$

*Note.* The *intruder intersection* is critical in *non-controlled airspace*. The main complexity gain in *controlled airspace* is from *rule application*. Our *rule complexity* is in worst case depending on *Reach Set* node count and *Active Collision Cases* count.

$$\texttt{Apply Our Rules:} \quad \mathcal{O}(|activeCollisionCases| \times |nodes|)$$

For *Hard/Soft Constraints* The algorithm used for intersection polygons was selected based on study [144], the selected algorithm *Shamos-Hoey* [145]. The *calculation complexity* boundary is given like follow:

$$\texttt{Hard Constraints Intersection:}$$
$$\mathcal{O}(|cells| \times |hardConstraints| \times \max|constraintPoints|^2)$$

$$\texttt{Soft Constraints Intersection:}$$
$$\mathcal{O}(|cells| \times |softConstraints| \times \max|constraintPoints|^2)$$

Each *threat* category application in *Mission Control Run* is done after *each intersection* in $7^{\text{th}}$ to $10^{\text{th}}$ step. All ratings (tab. 6.5) expect $Reachibility(cell_{ij,k})$ and $Reachibility(Trajectory)$ are calculated. The *calculation complexity* boundary for one *reachibility rating* is $\mathcal{O}(1)$. (eq. 6.138, 6.139). The *Recalculate Reachibility* operation applied $4\times$ have maximal *complexity* boundary given as follow:

$$\texttt{Recalculate Reachibility:} \quad \mathcal{O}(4 \times (|nodes| + |cells|))$$

Each time at the end of in $7^{\text{th}}$ to $10^{\text{th}}$ step the *Avoidance Path is Selected*. The *Worst Case* (expected) scenario is to *select* four paths for each *treath* application. The algorithm for *best path selection* (alg. 6.6) iterates over all *cells* in avoidance grid and over all *trajectories* passing trough that cell. The complexity boundary for *path selection* is given as follow:

$$\texttt{Select Path:} \quad \mathcal{O}\left(4 \times \left(|cells| + \frac{|nodes|}{|cells|}\right)\right)$$

**Conclusion:** Overall approach complexity is *low*. If proper *Information Sources* with efficient clustering and *intersection models for intruders* are used, the approach will stay within *non-polynomial complexity*. The average load time for *testing scenarios* is summarized in (tab. 7.53).

*Note.* The calculation of *Reach Set* is eliminated by pre-calculation for *state range* [7].

### 6.7.5 (R) Safety Margin Calculation

**Safety Margin Determination:** To determine *safety Margin* the *Rule of Thumb* is used:

$$maximalBodyRadius \leq safetyMargin \leq 2 \times turningRadius \tag{6.155}$$

The *lower boundary* is given by *UAS* construction. because the *UAS* body is considered as *unit ball* with radius given as *maximal body radius*.

The *upper boundary* is optional, The *double of* turning radius is used by the *conservative approach* [153].

**Safety Margin Bloating:** The *discretization* of *Reach Set*, *Operation Space* and *Decisions* imposes standard *mixed integer* problem in terms of *safety*. This section covers *non-exhaustive* list of possible *Safety Margin Bloats* in our approach.

**Own Position Uncertainty Bloat:** The *sensor fusion* is precise, but not *exact* in own UAS position determination. The maximal usual disparity needs to be accounted into *Safety Margin*.

**Intruder Position Uncertainty Bloat:** The *sensor fusion* of Intruder is precise, but not *exact* in own UAS position determination. The maximal usual disparity needs to be accounted into *Safety Margin*.

**Weather bloat:** The *Weather* impact type may result to increased *safety margin*. Example: UAS is not humidity resistant, the clouds will be avoided from greater distance.

**Airspace bloat:** The *Airspace* depending on cluster or *country* may require greater separation distances, depending on circumstances. The example can be UAS directive to keep minimal separation from obstacles. The *Safety Margin* is usually overridden by UTM directive value.

**UTM Synchronization Bloat:** *Both UAS* decision times were *synchronized*. The *intruder* can be offset for *full decision frame*. This is not an assumption, but it shows critical performance. Usually safety margin is bloated for (worst case offset):

$$safetyMarginBloat = \begin{pmatrix} intruderVelocity \times \dots \\ intruderDecisionFrame \end{pmatrix} [m, ms^{-1}, s] \tag{6.156}$$

## 6.8 (R) UAS Traffic Management

The *Traffic Management* for UAS is based on existing Air Traffic Management System for manned aviation [147]. The controlled airspace segments are *static* and have one *authority for one zone* principle. The dynamic zones have been proposed in [3]. But it will be omitted for *simplification purpose*. The necessity for *UAS integration* into *National Airspace* have been outlined in [38].

The latest *Airbus blueprint* [123] outlines some functionality. The main purpose of this section is to show *Reach Set based Approach* capability to follow *Usual Air Traffic Management* commands.

The *section* is organized to introduce:

1. *UTM Architecture* (sec. 6.8.1) - centralized ATM like authority over airspace cluster.
2. *Cooperative Conflict Resolution* (sec. 6.8.2) - the model used for conflict resolution in *controlled airspace*.
3. *Non-Cooperative Conflict Resolution* (sec. 6.8.3) - the model used for conflict resolution in *non-controlled* airspace and in *emergency avoidance*.
4. *Handling Standard Collision Situations* - head on approach (sec. 6.8.4), converging situation (sec. 6.8.5), overtake (sec. 6.8.6).
5. *Position Notification* (sec. 6.8.7) - position notification design.
6. *Collision Case* (sec. 6.8.8) - calculation and handling of *collision situations*.
7. *Weather Case* (sec. 6.8.9) - definition and handling of *weather hazards*.

### 6.8.1 (R) Architecture

**UTM concept** is based on *asynchronous event-based control* [154]. *Event* in *controlled airspace* is handled in form of *cases* [155]. There are following *event sources*:

1. *Weather Information Service* (from [156]) - used to create *weather case* (tab. 6.8).
2. *Position Notification from UAS systems* (tab. 6.6) - used to create *collision cases* (new functionality) (tab. 6.7).

**Decision frame** (eq. 6.157). The *UTM* is operating in discrete decision frames which are starting on current *decision time* and ending at next *decision time*:

$$decisionFrame_i = [decisionTime_i, decisionTime_{i+1}[, \quad i \in 1, \ldots, k, k \in \mathbb{N}^+ \qquad (6.157)$$

**Event based airspace control** is collecting events in previous $decisionFrame_{i-1}$ and issuing commands in current $decisionFrame_i$.There are following phases during the *UTM frame* cycle:

1. *Planning* - the detection phase, when the hazardous situations are assessed.
2. *Fulfillment* - the monitoring phase, when the state of affairs for directives and mandates is full filled by controlled UAS systems.
3. *Acknowledgement* - the closing phase, when UTM assess and acknowledges the performance of controlled UAS systems.

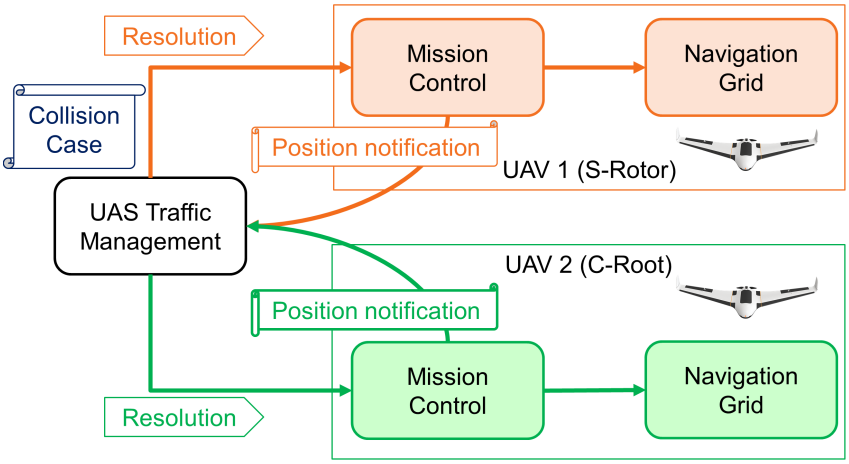



Figure 6.26: UAS Traffic Management (UTM) architecture overview.

**Architecture:** (fig. 6.26). There are multiple UAS systems equipped with standard *Mission Control* and *Navigation* procedures.

Depending on the *airspace cluster* decision time frame they are sending *periodical position notifications* (tab. 6.6).

The *UAS Traffic Management* (UTM) collects the event data from *Weather Information Service* and *Position Notifications* calculating respective *cases*.

If there is an *active collision/weather case* the *UTM* will send *resolutions* to respective airspace attendants.

### 6.8.2 (R) Cooperative Conflict Resolution

**Idea:** There is a *final decision maker* (absolute authority) in conflict resolution. This authority is *UTM* or *air traffic attendant* with higher priority. The future *UTM system* is such authority. The approach to mixed conflict resolution is mentioned in [157], based on navigation [158]. This is similar to our approach.

*Note. Open Issue:* Decentralized model with UTM as approver of directives is possible, but that is topic for own research.

**Goal:** UAS is obligated to follow up committed mission plan with given precision. There is one to five percent allowed deviations for ATM mission plans. Similar rates are achievable according to [157]. This requirement is given by [147] ICAO 4444 document for ATM operations.

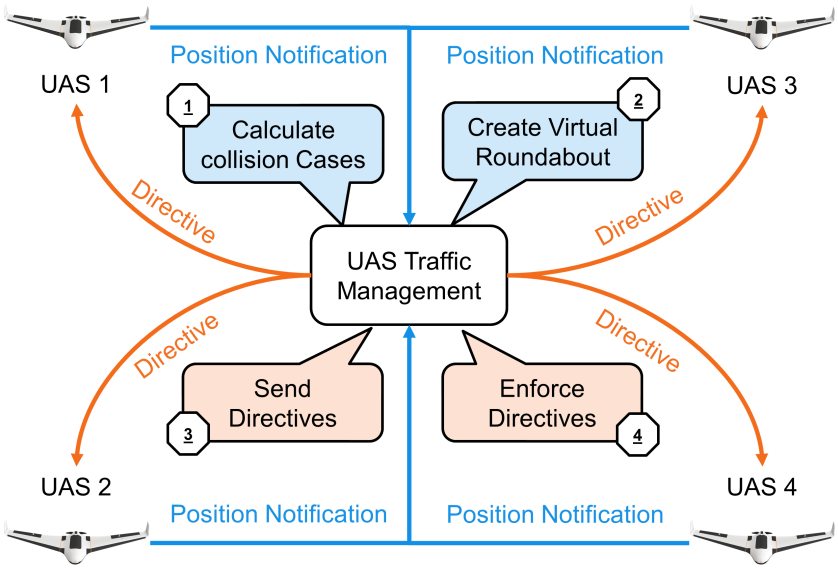



Figure 6.27: Cooperative conflict resolution via UTM authority.

**Cooperative conflict resolution** (fig. 6.27) shows functional diagram of one *UTM time-frame* there are following actors:

1. *Unmanned Autonomous System* (UAS) equipped with necessary navigation and communication modules, providing the unique *identification number*.
2. *UAS Traffic Management* (UTM) posing as central authority for given *airspace cluster*.

The following steps are executed during *Cooperative conflict resolution*:

1. $UAS_* \rightarrow UTM$ *Send position notification* - each *UAS* is notifying the authority (UTM)
2. ⟳ *UTM Calculate collision Cases* - UTM gathers data and predicts possible collisions then it tries to link them and manage the situation.
3. ⟳ *UTM Create virtual Roundabout* - active collision cases are aggregated into virtual roundabout.

4. $UTM \rightarrow UAS_*$ *Send directives* - UTM sends commands to UAS systems whom needs to change their planned trajectories.

5. $UTM \rightarrow UAS_*$ *Enforce directives* - UTM is periodically checking constraints imposed in previous *decision frames*.

### 6.8.3 (R) Non-Cooperative Conflict Resolution

**Idea:** There is *main UAS(1)* which is flying in open *non-controlled* airspace. There are other *UAS* operating in its vicinity. It is expected that they are claiming their *planned trajectories*. The *Main UAS(1)* detects the collision with other *UAS*(2-4).

There is no *final decision maker* nor *supervising authority*, all communication participants have similar level of rights.

*Note.* There is assumption that other airspace users are behaving like intruders, without intent to destroy or harm. The *adversarial behaviour* is not accounted. The response from *intruder* is not mandatory in *non-controlled* airspace.

**Goal:** Provide *mutual avoidance mechanism* in *non-controlled* airspace. Considering the equal standpoint of all airspace attendants.

**Conflict resolution:** The conflict resolution depends on current mode and *handshake* between airspace attendants. The non-cooperative behaviour have been implemented like follows:

1. *Navigation mode* - every *airspace attendant* is calculating own *collision cases* and checking the behaviour of the other (virtual UTM).

2. *Emergency avoidance mode* - is depending on communication mode:

   a *Response mode* - claiming separation methods and using avoidance mechanism (Avoidance grid with intruder model in our case).

   b *Blind mode* - every conflict side picks own strategy respecting given *rules of the air*.

*Note. Intruder Intersection model selection:* UAS based on Event detects possible collision for some reason UTM directive is out of question, then try to claim separation (body volume intruder model (sec. 6.6.3)), If separation fails, go full survival mode (uncertain intruder model (sec. 6.6.3)).

**Special cases in manned aviation:** There are IFALPA reports which can give us overview of *enforced non-cooperative* mode causes in *controlled airspace*:

1. *VFR disabled* - flying in fog or thick clouds can render pilot vision, similar to UAS cameras/LiDAR.

2. *IFR equipment broke* - the sensor malfunction is more likely to happen due the lesser redundancy in UAS systems.

3. *C2C Link disabled* - communication loss is more likely to happen, due the lesser redundancy.

4. *ATM failure* - the ground control module of UTM can also fail.

*Note.* Traffic management related fails are lesser than 0.001 cases per one flight (according to IFALPA [159]).

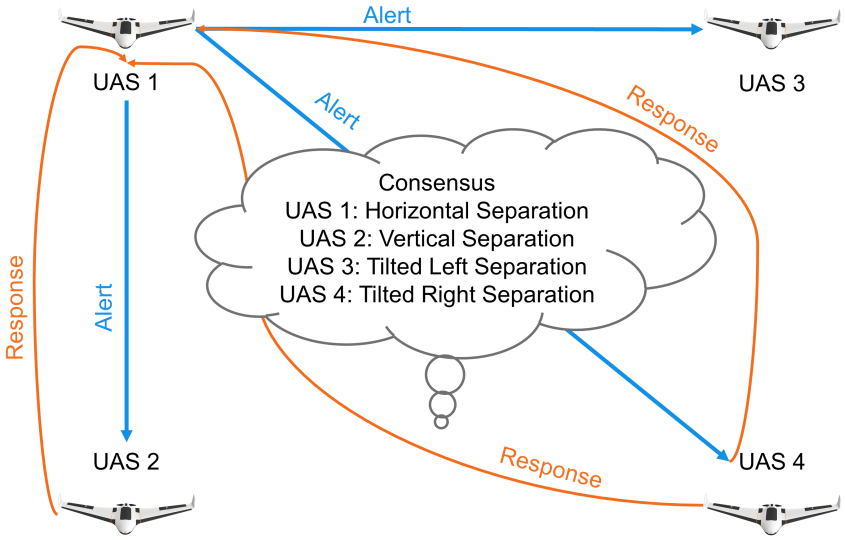



Figure 6.28: Non-cooperative conflict resolution via UAS claims.

**Response mode scenario example:** The *main UAS(1)* is going to collide with other $UAS(2\text{-}4)$:

1. $UAS(1) \rightarrow UAS(2-4)$ sends position and heading notification.
2. $\circlearrowleft UAS(2-4)$ calculates possible collisions.
3. $UAS(2-4) \rightarrow UAS(1)$ sends response to the *main UAS(1)* with claimed separation mode.
4. $\circlearrowleft UAS(1)$ acknowledges proposed *separation modes*.
5. $\circlearrowleft UAS(1-4)$ avoids each other using claimed separation mode, because every $UAS$ achieved *consensus*.

*Note.* The mutual consensus is not usually achieved via C2 communication. The most common case is *assuming separation mode*. This case is shown in (sec. 7.3.7)

### 6.8.4 (R) Handling Head on Approach

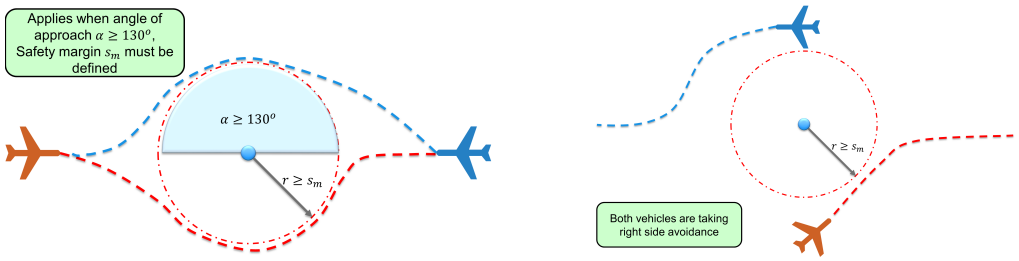
**Goal:** Identify required parameters sufficient for automatic solution of *Head on collision* situation.

**VFR:** The *Visual Flight Rules* (VFR) are specified in annex 2 [160] and there is a *Head on* approach for two or more air crafts. The definition is rather vague: "The pilot should diverge from original heading to the right to create sufficient safe space for avoidance".

**IFR:** The *Instrument Flight Rules* in annex 2. [160] and 11. [161] are defining the boundaries and events for success full *Head on resolution* in larger detail.

The parameter values are useless due the UAS scaling factor, following parameters can be used in UTM:

1. *Angle of approach* $\geq 130°$ - the minimal planar angle between aircraft positions and expected collision point is in interval $[130°, 180°]$.
2. *Minimal detection range* - the minimal detection range of head on collision is $2 \times turningRadius + safetyMargin$.
3. *Safety margin* - during avoidance all aircraft keeps mutual distance at least at value of safety margin.




(a) Detection. (b) Resolution/Closing.

Figure 6.29: Head on approach detection/resolution/Closing

**Triggering Events:** The *head on approach* (fig. 6.29) *triggering events* are following:

1. *Detection* (fig. 6.29a) - the *collision case* is open, when *collision point* with respective angle of approach is detected. This must happen until *point of no return* is achieved.
2. *Resolution* (fig. 6.29b) - the *virtual* roundabout is enforced until the closing condition is met.
3. *Closing* (fig. 6.29b) - based on condition that all vehicles are heading away from *collision point* and heir mutual heading is neutral or opposite.

**Virtual roundabout:** The *flight levels* can be abstracted as *virtual 2D surface*. The *airspace attendants* are moving on virtual routes which can cross each other. The idea is to create virtual roundabout with enforced velocity to enable smooth collision avoidance.

1. *Center* - the center defined in *airspace cluster* local coordinate system (flight level defining the horizontal placement).
2. *Diameter* - the minimal distance to *center*, accounting the *wake turbulence* and other phenomenons.
3. *Enforced velocity* - all attendants at *virtual roundabout* keeps same velocity. It helps to keep constant mutual distances.

### 6.8.5 (R) Handling Converging Maneuver

**Goal:** Identify *required parameters* sufficient for automatic solution of *Converging Maneuver*.

**VFR:** The *Visual Flight Rules* (VFR) are specified in annex 2 [160]. The rule is different from *Head on Approach* (sec. 6.8.4), because there are multiple roles depending on relative aircraft position:

1. *Avoiding Aircraft* - there is an aircraft on relative right side (blue).
2. *Right Of the Way (ROA) Aircraft* - there is an aircraft on relative left side (red).

The *avoiding aircraft* should take *right of the way aircraft* from behind, with sufficient *safety margin*, and return to original *heading* afterward. The *magnitude* of *avoidance curve* must consider *wake turbulence* and other impacts of *avionic properties*.

*Note.* This rule is applied only when the both *aircraft* belong to same *maneuverability class* [160].

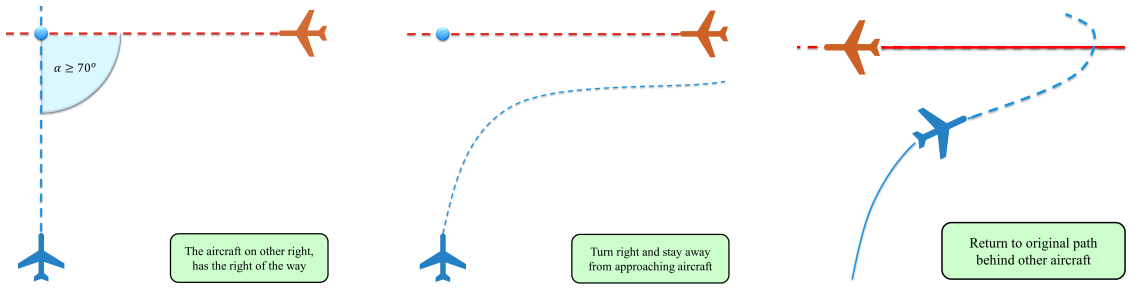
**IFR:** The *Instrument Flight Rules* in annex 2. [160] and 11. [161] are defining *converging maneuver* in detail.

The *parameters* from *head on approach* can be reused:

1. $70° \leq$ *Angle of Approach* $< 130°$ - the minimal planar angle between aircraft position and expected collision point is in interval $[70°, 130°[$.
2. *Minimal detection range* - given as $turningRadius + safetyMargin$, while *safety margin* is accounting all impact factors.
3. *Safety margin* - during avoidance all aircraft keeps mutual distance at least on value of *Safety Margin*.

*Note.* Lesser *angle of approach* induces stronger wake turbulence impact on avoiding aircraft. This results into increase of *safety margin*.

The *wake turbulence* is represented as droplet at the back of the plane. *Wake turbulence range* can be calculated based on wake turbulence cone.




(a) Detection. (b) Resolution. (c) Closing

Figure 6.30: Converging maneuver Detection/Resolution/Closing

**Triggering Events:** The *converging maneuver* (fig. 6.30) *triggering events* are following:

1. *Detection* (fig. 6.30a) - The *avoiding airplane* (blue) detects *collision point* (blue circle) which satisfy the *converging maneuver conditions*. The distance between *aircraft position* and *collision point* is lesser than *detection range*.

2. *Resolution* (fig. 6.30b) - the *Right Of the Way aircraft* (red) stays at the original course. The *avoiding aircraft* (blue) follows the *parallel* to other *plane*. The distance of *avoiding plane* to *other plane trajectory* is greater or equal to *safety margin*.

3. *Closing* (fig. 6.30c) - when both planes have opposite heading and they miss each other the converging maneuver can be closed. The *avoiding airplane* will return to *original trajectory*, while keeping the distance from *other plane* (red) at greater or equal to *safety margin*.

### 6.8.6 (R) Handling Overtake Maneuver

**Goal:** Identify *required parameters* sufficient for automatic solution of *Overtake Maneuver*

**VFR:** The *Visual Flight Rules* (VFR) are specified in annex 2 [160]. The rule states that faster air traffic attendant may overtake slower one, from right side keeping sufficient distance (*safety margin*). There are two forced roles:

1. *Overtaking* - faster aircraft with similar heading cruising in similar altitude than *overtaken* (blue). It is expected that *faster aircraft* has maneuvering capability to avoid slower aircraft.

2. *Overtaken* - slower aircraft which keeps the *Right of the way*

*Note.* This rule is applied only when both aircraft have same maneuverability class [160]. The overtake is considered *borderline emergency maneuver* in controlled airspace because the aircraft tend to keep similar velocity in similar cruising altitude. The overtake is usual in *non-controlled airspace*.

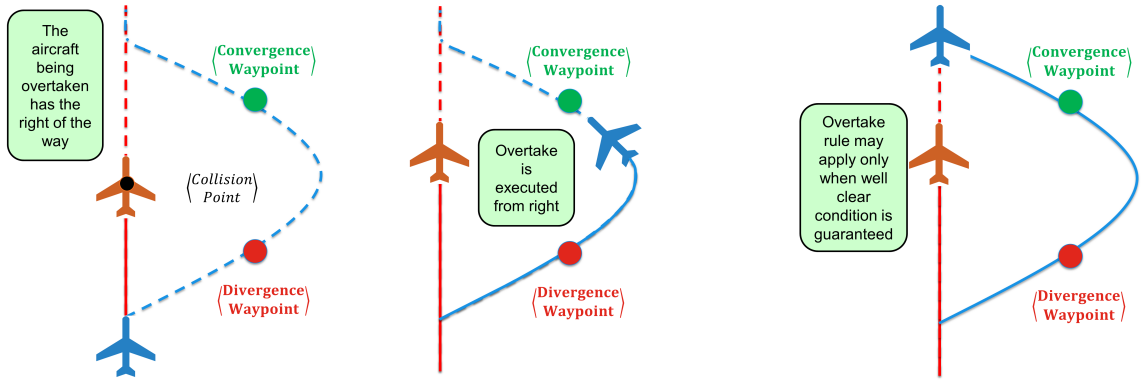
**IFR:** The *Instrument Fight Rules* in annex 2. [160] and 11. [161] are defining the converging manual in detail:

1. $0° \leq$ *Angle of Approach* $< 130°$ - the minimal planar angle between aircraft position and expected collision point is in interval $[0°, 70°[$

2. *Minimal Detection Range* - given as $2 \times reactionTime \times speedDifference$.

3. *Safety Margin* - during avoidance the overtaking aircraft keeps the minimal distance of *wake turbulence* of overtaken aircraft in own flight altitude.

*Note.* The *Safety Margin* is sufficiently small, because speed difference is usually much lesser than in case of *Head on approach*. The *Wake turbulence* can be avoided completely by taking the higher altitude level than overtaken aircraft.




(a) Detection. (b) Resolution. (c) Closing.

Figure 6.31: Overtake maneuver Detection/Resolution/Closing

**Triggering events:**

1. *Detection* (fig. 6.31a) - occurs when the distance between *overtaking* (blue) and overtaken (red) is approaching *minimal detection range* or double of *safety margin*. If the performance of *overtaking aircraft* (blue) allows to take *sharp right side overtake* the *Maneuver starts*, otherwise *overtaking aircraft* (blue slows down) and keeps at least *safety margin distance* to avoid *wake turbulence*.

2. *Resolution* (fig. 6.31b) - *overtaken* (red) is keeping same heading and *speed* during overtake maneuver. The *overtaking* (blue) projects two waypoints: *Divergence* and *Convergence* keeping the required separation minimum during overtake. Then the *overtaking* (blue) diverges heading to *Divergence waypoint*. When the *Divergence waypoint* is reached by *overtaking* (blue) aircraft it changes to *original heading*.

3. *Closing* (fig. 6.31c) - the *closing* of *Overtake* starts when *overtaking* aircraft (blue) have sufficient lead over *overtaken* aircraft (red). The *overtaking* aircraft (blue) can safely change the heading to original waypoint.

**Constant Cruising Speed:** Most of traffic attendants at same flight level have similar (close to constant) cruising speed. Lower flight levels are for slower turbo-prop planes and higher altitudes are for jet planes. It is stated that this principle will persist even when UAS will be integrated [162, 163, 164] in multiple air-traffic models.

### 6.8.7 (R) Position Notification

**Motivation:** The *position notification* (tab. 6.6) is designed for further *collision case resolution* (sec. 6.8.8). It is similar to ADS-B[4] message information.

The main purpose is to broadcast the *position notification* in *controlled aerospace*. The broadcast for *non-controlled* airspace needs to contain *intruder properties*, *preferred separation mode* and *near miss margin*.

**Position:** The position is defined in *Global Coordinate System* using GPS for latitude and longitude. The barometric altitude is required for controlled airspace, preferred for non-controlled airspace.

[4]ADS-B versions and message containment: `https://mode-s.org/decode/adsb/version.html`.

**Heading:** The *Linear Velocity* combined with heading in standard *North-East* coordinate frame is used.

**Flight Levels:** The *flight level* is notified to UTM for *collision detection* purposes. There is *main flight level* where *aircraft* belong physically. There is *passing flight level* form which/to which is aircraft emerging [147].

**Aircraft Category:** The aircraft category impacts the prioritization of *role assessment* by UTM/ATM. The following categorization is proposed by *manned aviation pilot community*, from the highest to the lowest right of the way priority:

1. *Manned aviation in distress* [160] - the aircraft with impaired capability switched to emergency mode. The emergency mode is usually acknowledged by authority in controlled airspace.

2. *Balloon* (manned) [160] - the aircraft with *altitude* control and very slow dynamics implying very low maneuverability.

3. *Glider* (manned) [160] - the aircraft with *full control* but without own *propulsion*. The overall *maneuverability* is good, but the *velocity* changes are impossible with sufficient flexibility.

4. *Aerial towing* (manned) [160] - the towing aircraft usually have *own propulsion* and full maneuverability, the only constraint is *towed load*. The towed load decreases overall maneuverability.

5. *Airship* (manned) [160] - the airship have *own propulsion* and full maneuverability, the constraint is low acceleration/deceleration and huge turning radius.

6. *Other manned aviation* [160] - containing all vehicles with required level of *airworthiness* for given operational *altitude*. They usually have required maneuverability.

7. *UAS Autonomous* (proposed) [165] - containing all autonomous UAS, the lower flexibility is expected at beginning of integration.

8. *Remotely Piloted Aerial System (RPAS)* (proposed) [165] - has lesser priority due the higher response rate of the pilot.

*Note.* This categorization reflects only Pilot community statement, the general priority rule is broken, because maneuverability and vulnerability should be always considered as key decision factor.

**Maneuverability:** The maneuverability is the real key factor in priority assessment. The components of maneuverability are *maximal/mean acceleration/deceleration*, *climb/ descent rate* and *turning ratio/radius*. The comparison can be done by solving *pursuit problem* using *Reach Sets* [46, 47].

The *Maneuverability categorization* is based on *original aircraft priority categorization* [160] accounting UAS/RPAS as equal to *manned aviation*. The ordered list from the highest to the lowest priority goes as follows:

1. *Impaired control* (Distress aircraft) - any aviation attendant in distress has the priority in case of the conflict occurrence.

2. *Altitude control/No* (Balloon, Hovering aircraft) - the balloon type crafts does not have any type of propulsion and horizontal movements follows the airflow in given altitude.

3. *Full control/No propulsion* (Gliders of any sort) - the gliders can control their horizontal position, but there are limits to altitude control and acceleration/deceleration.

4. *Full control/Linear propulsion* (Any aircraft of plane type) - the *towing aircraft's* and *airplanes* belong there, the difference is the *flexibility* of *maneuvering*.

5. *Full control/VTOL capability* (Any aircraft with VTOL) - the *other aircraft* capable to do on spot turn. The typical representative is *quad-rotor copter*.

There are other aspects like *minimal required* acceleration/deceleration/turn ratio to operate in selected segment of the *airspace*. These should be specified later by *Minimum Operational Performance Standards* (MOPS).

**Safety Margins:** The *Safety Margin* for *Well Clear Condition* value is based on *situation*. There is also an *universal safety margin* which guarantees the minimal safety for encountering intruder.

The most prevalent effect is *Wake turbulence* therefore *wake turbulence cone* angle $[0 \circ -90\circ]$ and radius.

The *safety Margin* for situation-based avoidance is given by list of supported manuevers maneuvers, there is converging (sec. 6.8.5), head on (sec. 6.8.4), overtake (sec. 6.8.6) safety margins

| | |
|---|---|
| **Position** | |
| latitude | based on GPS/IMU sensor fusion. |
| longitude | based on GPS/IMU sensor fusion. |
| altitude | barometric altitude *Above Mean Sea Level* (AMSL). |
| **Heading** | |
| orientation | orientation in standard North-East coordinate frame. |
| velocity | relative UAS velocity. |
| **Flight Levels** | |
| main | flight level, where UAS mass center belongs |
| passing | flight level, during climb/ascend, or when distance of UAS mass center to flight level boundary $\leq 250ft.$ . |
| **Registration** | |
| registration ID | is unique registration number *to be issued* by local aviation authority for UTM communications purposes. |
| flight code | or mission code is unique identification number for approved mission plan which is going to be flown by UAS. |
| UAS name | optional UAS identifier to increase human recognition. |
| **Categorization** | |
| craft category | ICAO main category, based on vehicle type. |
| maneuverability | secondary categorization specifying size class, horizontal/vertical turning radius, minimal and maximal cruising speed. |
| **Safety margins** | |
| universal | minimal safety margin for any avoidance situation |
| head on | minimal distance from other similar maneuverability class aircraft in case of head on approach. |
| converging | minimal distance from other similar maneuverability class aircraft in case of head of converging maneuver. |
| overtake | minimal distance from other similar maneuverability class aircraft in case of overtake maneuver. |
| wake angle | for wake turbulence cone. |
| wake radius | for wake turbulence cone. |

Table 6.6: Time-stamped *position notification* structure.

### 6.8.8 (R) Collision Case

**Collision Case Purpose:** There is a need for detection and tracking of possible *controlled airspace traffic attendants* collisions. The presented *collision case structure* (tab. 6.7) is minimalist reflection of *ATM* requirements. Following aspects of *collision case* life cycle are explained in this section:

1. *Base terminology* - the definition of *enforcement procedure* and difference between *Resolution* and *Mandate* from UTM authority. The *severity issue* is open.
2. *Calculation of single case for single decision frame* - step by step calculation and threat evaluation. Prequel to *life cycle*.
3. *Life cycle* gives outlook how collision case data are handled trough longer period of time, notably: *Opening*, *collision point handling*, *safety margin handling*, and, *Closure*.
4. *Merge procedure for multiple cases in single cluster* - the naive *merge procedure* to solve *multiple collision cases* via *virtual roundabout*.

**Resolution/Mandate Enforcement:** *Enforcement procedure* is consisting from *Threat detection phase* and *Mitigation phase*. The *mitigation phase* is a time interval when *UTM* decision is enforced. The decision the UTM is enforcing is delivered in form of *Resolutions* and *Mandates*.

*Resolution* is an order from the *UTM* authority which is followed by subjected UAS. The *subjected UAS* can determine own behaviour to some extent. When there is emerging threat or other destructive event, like new non-cooperative adversary, the UAS is allowed to broke *resolution*.

*Mandate* is an order from the *UTM* authority which can not be broken at any cost. The example of the *mandate*: UAS is flying in airspace, the passenger in distress needs it to safely land. The UAS must obey mandate even at event of own destruction.

**Threat Severity Evaluation:** The threat severity evaluation is omitted partialy, all threats are considered as equal. All commands from *UTM authority* will be considered as *resolutions*.

**Calculation procedure:** Collision case is calculated for two *Registered UAS systems* in *Unified UTM time-frame*. The *unified UTM time-frame* is a short period of time in future when the anticipated situations are predicted.

**1st** The *position* and *orientation* is adjusted according to *mission plan*. Our implementation uses *Movement Automaton* as a predictor:

$$\begin{gathered} adjustedPosition = Position(Trajectory(notifiedState, futureMovements)) \\ adjustedOrientation = Orientation(Trajectory(notifiedState, futureMovements)) \end{gathered} \tag{6.158}$$

For other cases standard linear prediction can be used:

$$\begin{gathered} adjustedPosition = notificationPosition \times notificationVelocity \times timeDifference \\ adjustedOrientation = notificationOrientation \end{gathered} \tag{6.159}$$

**2nd** The *maneuverability*, *craft category*, *registration ID* are taken from *position notification*.

**3rd** *Collision case check procedure* goes like follows:

1. *Operation space checks* - the controlled airspace and flight level must match for proceeding.
2. *Maneuverability/Category check* - the maneuverability and UAS category must match. If there is mismatch then right of the way is forced to vehicle with higher priority.

**4th** *Linear Intersection test* is designed to calculate *closest distance* and *time* of *linear trajectory projections*, First for given *velocity* and *position* for UAS1 and UAS2 the helper variables are calculated:

$$\begin{aligned} A &= \|velocity_1\|^2 \\ B &= 2 * (velocity_1{}^T \times position_1 - velocity_2{}^T \times position2) \\ C &= 2 \times velocity_1{}^T * velocity_2 \\ D &= 2 * (velocity_2{}^T \times position_2 - velocity_2{}' \times position_1); \\ E &= \|velocity_2\|^2 ; \\ F &= \|position_1\|^2 + \|position_2\|^2 ; \end{aligned} \tag{6.160}$$

Then the projection parameters can be calculated:

$$\begin{aligned} time &= \frac{-B - D}{2 \times A - 2 \times C + 2 \times E} \\ destination_i &= position_i + velocity_i \times time, \quad i \in \{1, 2\} \\ collisionPoint &= \frac{destination_1 + destination_2}{2} \\ collisionDistance &= \|destination_1 - destination_2\| \end{aligned} \tag{6.161}$$

If $time < 0$ the trajectories are diverging from each other (because the closest points already occurred). The procedure ends, *collision flag* is not raised.

If $time > timeMargin$ the trajectories will get close to each other, but in further future and changes are anticipated. The procedure ends, *collision flag* is not raised.

If $0 \leq time \leq timeMargin$ the trajectories are converging to each other and distance needs to be checked. If $distance \leq collisionMargin$ then *collision flag* is raised and *collision point* is set.

*Note. Collision Margin* is some number which is determined based on aircraft category and maneuverability. Our work defines collision margin as follow:

$$collisionMargin = \forall situation : \max \left\{ \begin{matrix} safetyMargin(situation, UAS1) \\ +safetyMargin(situation, UAS2) \end{matrix} \right\} \tag{6.162}$$

Where *safety margin* for every possible situation is evaluated for both *UAS*.

**5th** The *trajectory* intersection is *Movement Automaton* specific collision detection method. Its based on the assumption that *UTM* have following information from *mission plan*:

1. *UAS state* - not only *position*, *orientation*, and, *velocity* vectors, but other mathematical model parameters mandatory for *movement automaton*.
2. *Movement Automaton* - movement automaton for our UAS system, so UTM can use it in predictor mode.
3. *Future Movements set* - up to reasonable prediction horizon $timeMargin$.

The *Movement Automaton* can be used as trajectory prediction for system initial state and future movements. The prediction function (eq. 6.163).

$$Prediction : UAS \times state \times futureMovements \rightarrow [x, y, z, t] \in \mathbb{R}^4 \tag{6.163}$$

*Note.* Then prediction for UAS1 is $Prediction_1$ and for UAS 2 $Prediction_2$, the predictions are synchronized meaning that time at position $i$ is equal in both discrete trajectory matrices.

The *collision distance* for predictor (eq. 6.163) is given as minimal distance of projected synchronized trajectories for UAS1 and UAS2. In our discrete enviroment the *collision distance* is given as (eq. 6.164).

$$collisionDistance = \min \left\{ \|point_1 - point_2\| : \forall \begin{pmatrix} point_1 \in Prediction_1, \\ point_2, \in Prediction_2, \\ t_1 \sim t_2 \end{pmatrix} \right\} \tag{6.164}$$

If $collisionDistance \leq collisionMargin$ condition is met, *collision flag* is set.

The collision point is then calculated as mean of *UAS positions* in prediction at time when distance is minimal. The final collision point is arithmetic mean of two positions (eq. 6.165).

$$collisionPoint = \frac{point_1 - point_2}{2} : \begin{pmatrix} point_1 \in Prediction_1, \\ point_2, \in Prediction_2, \\ t_1 \sim t_2 \text{ at minimal distance} \end{pmatrix} \tag{6.165}$$

*Note.* Collision point is overwritten by trajectory intersection (specific) method, the *linear intersection* is considered *general collision detection method*. The collision detection method in future UTM system needs to be determined. The *Trajectory intersection* method presented in this work is one of the possible candidates.

**6th** *Role determination* phase is invoked if and only if previous conditions are met and *collision flag* with *collision point* exists.

There is *adjusted position* of each UAS used as verticals and *collision point* used as center. First step is normalization of adjusted position around collision point for both UAS:

$$normalized_i = adjustedPosition_i - collisionPoint, \quad i \in \{1, 2\} \tag{6.166}$$

Then the right hand coordinate system internal angle calculation method is used:

$$angleOfApproach = \left|\text{atan2}\begin{pmatrix} normalized_1 \times normalized_2, \\ normalized_1 \circ normalized_2 \end{pmatrix}\right| \tag{6.167}$$

Based on *angle of approach* the *scenario type* is decided like follows:

1. $130° \leq angleOfApproach \leq 180°$ - the scenario type is set as *Head On Approach* (sec.6.8.4)
2. $70° \leq angleOfApproach < 130°$ - the scenario type is set as *Converging Maneuver* (sec.6.8.5)
3. $0° \leq angleOfApproach < 70°$ and *different speed* - - the scenario type is set as *Overtake Maneuver* (sec.6.8.6)

Based on *relative position* and *scenario type*, the *avoidance role* like follows:

1. *Head On Approach* enforces following:
   a. The *avoidance role* us set as *RoundAbounting* for both UAS.
   b. None of the *UAS* does not have the *Right Of the Way.*
2. *Converging Maneuver* enforces following:
   a. *UAS* without free right side have role set as *Converging.*
   b. *UAS* with free right side have the *Right Of the Way.*
3. *Overtake Maneuver* enforces following:
   a. *Slower UAS* has *Overtaken* role with *Right Of the Way.*
   b. *Faster UAS* has *Overtaking* without *Right Of the Way.*
   c. *Faster UAS* mission plan is altered with *divergence and convergence waypoints.*

**7th** *Safety Margin Calculation* Is invoked when the collision case is *Active.* The *Active Collision Case* in this time-frame means that *Collision Flag* is raised. The *avoidance role* determines *safety margin calculation.*

If *Head On Approach* is case type of *Head collision case* then *safety margin* is calculated as maximum of sum of *default* margins or *head on* margins:

$$safetyMargin = \max\begin{Bmatrix} default(UAS1) + default(UAS2), \\ headOn(UAS_1) + headOn(UAS_2) \end{Bmatrix} \tag{6.168}$$

If *Converging Maneuver* is case type of *Head collision case* then *safety margin* is calculated based on *avoiding UAS* as maximum of opposing UAS *default margin* and avoiding *converging margin*:

$$safetyMargin = \begin{cases} uas1.role = Converging: & \max\begin{Bmatrix} default(UAS2), \\ converging(UAS1) \end{Bmatrix} \\ uas1.role = Converging: & \max\begin{Bmatrix} default(UAS1), \\ converging(UAS2) \end{Bmatrix} \end{cases} \tag{6.169}$$

If *Overtake maneuver* is case type of *Head collision case* then *safety margin* is calculated as maximumo of *default, overtaking, overtaken* margins of both UAS:

$$safetyMargin = \max \begin{Bmatrix} default(UAS1), default(UAS2), \\ overtaken(UAS_1), overtaking(UAS_2), \\ overtaking(UAS_1), overtaken(UAS_2) \end{Bmatrix} \tag{6.170}$$

**Collision Case Chaining** is procedure when multiple active collision cases for different *time-frame* are chained and creates the time ordered series of *collision cases*. There are two notable instances in the *chain*:

1. *Head Collision Case* - Collision case when the first danger was detected. The notable parameters are *collision point* and UAS *avoidance roles*, because these are enforced by *Rule engine* (sec. 6.9). The *head collision case* is first in chain.

2. *Tail Collision Case* - Collision case when the *collision danger* was not detected. The *tail collision case* is last in the chain.

*Note.* The *Chaining* of *collision cases* is rather primitive and sensitive for errors/noise.

The *Consistency of Avoidance Manuever* is ensured by enforcing *head collision case* parameters.

**Collision Cases Merge** also known as *Collision Point Adjustment Procedure* purpose it to *merge* multiple collision cases into one general collision case. The clustering is used to identify *airspace congestion events* [166]. Example of *airspace clustering* is given it [167].

The main idea is to *encapsulate multiple collision cases* into one virtual roundabout to ease *traffic load* [168]. The potential risk on *turbo roundabouts* have been outlined in [169].

There are *active collision cases* in focused *cluster* in *controlled airspace*. The multiple collision cases can pop up in different *start times* and they can be active for different *time period*.

The *Collision point* is replaced with *roundabout center* point (eq. 6.171). The *roundabout center* is calculated as weighted average of *active collision cases* collision points. The $weight \in [0, 1]$ depending on severity rating of collision case.

$$roundaboutCenter = \frac{\sum_{\in Cluster}^{\forall collisionCase} collisionCase.collisionPoint \times weight}{|collisionCase \in Cluster|} \tag{6.171}$$

*Note.* The weight in (eq. 6.171) is set to 1 for all time, the weight calculation needs to be determined in future works.

The *smallest circle problem* defined and solved in [33, 140] is used to determine safety margin in our approach. The *naive approach* determining *roundabout safety margin* is to take the maximum of all open case *safety margins* including default ones (eq. 6.172).

$$safetyMargin = \max \begin{Bmatrix} case.UAS_i.roundaboutSafetyMargin, \\ case.UAS_i.defaultSafetyMargin \end{Bmatrix}, \\ \forall case \in Cluster, \quad UAS_i \in \{1, 2\} \tag{6.172}$$

| Data for both attendants | |
|---|---|
| adjusted position | predicted from previous *position notifications* (6.6) data at time of *UTM decision frame* start. |
| adjusted orientation | predicted from previous *position notifications* (6.6), *mission plan*, and *expected velocity*. |
| velocity | proclaimed velocity for given *UTM decision time frame*. |
| registration ID | is unique registration number issued by local aviation authority |
| craft category | from *position notifications* (6.6). |
| maneuverability | from *position notifications* (6.6). |
| mission plan | is acquired from *allowed mission registers* where it has been registered prior UAS flight |
| safety margins | list of all safety margins, derived based or craft categorization or overridden by *position notifications* (6.6). |
| avoidance role | is given based on situation evaluation. |
| trajectory prediction | simulated based on *position notification* (6.6) and *mission plan*. |
| **Collision case calculated data** | |
| linear intersection | is predicted on attendants *position*, *heading*,*velocity*, based on *maneuverability* certain thresholds are applied to determine safety properties. |
| trajectory intersection | is predicted on attendants *position*, *velocity*, *heading*, and *related mission plans*, based on *maneuverability* certain thresholds are applied to determine safety properties. |
| collision point | is created if there is risk of medium/short time period collision, if head collision case has not been closed, collision point is inherited. |
| adj. collision point | is created if there exists at least one active collision case in nearby surroundings of this case collision point (cluster). |
| angle of approach($\alpha$) | is calculated based on attendants *velocity* and *position*, range is $[0°, 180°]$, it determines *primary avoidance roles*. |
| safety margin | is calculated based on *avoidance roles*, *maneuverability*, collision indicators, and *angle of approach*. |
| margin adjustment | is calculated based on *linked collision cases*, *estimation errors* and *weather*. |
| linked cases | contains list of collision cases which are active and can have impact on this *collision case*. |
| head case | is reference to collision case in time frame when it was first opened. |
| **Collision case indicators** | |
| linear intersection | indicates if there was safety breach on linear trajectories estimation with risk of direct collision. |
| trajectory intersection | indicates if there was breach on trajectory estimation, with risk of direct collision. |
| well clear breach | indicates if *linear projection* or *trajectory projection* breaches *well clear barrel* in *controlled airspace*. |
| active case | indicates if case is still open. |

Table 6.7: Collision case structure for given decision time-frame.

### 6.8.9 (R) Weather Case

**Motivation:** The weather, as defined in (eq. 4.5), impacts flight and system dynamics, therefore it impacts the *reach set* is impacted. The *weather impact* can be solved by policy application:

1. *Weather Acceptance* - for bigger *UAS* the normal weather impact does not pose significant risk. The *segmented movement automaton* (def. 21) with *Weather situation* as discrete state is used.

2. *Weather Avoidance* - all *weather* impact zones are considered as hard constraints with protective *soft constraint* around.

3. *Combined approach* - depending on type of impact and declared UAS impact resistance the zones are divined into *soft* and *hard* constraints.

*Note.* This work handles small *UAS* avoidance, these are very sensitive to any weather impact, therefore *Weather impacted areas* will be considered as *hard constraints with soft constraint protection zone.*

The original *weather impact zone* is considered as obstacle body and enforces the body margin.

The surroundings of *weather impact zone* up to *safety margin* distance is considered as *soft constraint zone* (implemented as bloated polygon).

**Purpose:** The *weather case* (tab. 6.8) is broadcast ed by *Airspace Authority* to *impacted area*, each *UAS* then change their mission according to *their maneuvering capabilities.* Each trajectory must lead away from *constrained area.* The algorithm used for intersection selected based on [144] the selected algorithm *Shamos-Hoey* [145].

**Constrained Area:** Constrained area can be defined as *static* (sec. 6.5.3) or dynamic constraint (sec. 6.6.5). The *constraint center* is defined on horizontal plane like follow:

$$ConstraintCenter = center \in [latitude, longitude] \tag{6.173}$$

The *Convex Polygon* boundary is defined on horizontal plane, contains at least 3 vertexes:

$$ConvexPolygon = \{point_i : point_i \in [latitude, longitude], i \geq 3\} \tag{6.174}$$

The *Vertical constraint* is defined as *range of barometric altitude* (Above Mean Sea Level):

$$VerticalConstraint = [startAltitude, endAltitude] \tag{6.175}$$

**Additional parameters** : Following additional parameters with additional purpose can be attached to *Weather Constraint*.

1. *Type* - defines required resistance - moisture, temperature, wind.
2. *Severity* - defines impact for each *aircraft category*, this is used in soft/hard type assessment.
3. *Duration* - start and end of *constraint* validity, if not defined valid for all *UAS mission time*.
4. *Velocity* - velocity and last position assessment time.

*Note.* Our implementation does not consider the *type* or *severity*. All *weather impact* is considered as *hard constraint*. The velocity differentiates *static* $(= 0)$/*moving* $(> 0)$ *constraints*.

**Avoidance System:** Resolve similar to *Converging/Overtake Maneuver* depending on the *angle of approach*. The *virtual roundabout* is utilized for *static constraints*, the *intruder model* is utilized for *dynamic constraints*.

| **Constrained area** | |
|---|---|
| center position | is given as geometrical *center point of boundary*. |
| boundary | is represented as *convex polygon* on latitude-longitude plane. |
| start altitude | is lover boundary barometric altitude given at above mean sea level, where given weather factor have significant impact. |
| end altitude | is upper boundary barometric altitude given at above mean sea level, where given weather factor have significant impact. |
| **Additional parameters** | |
| type(s) | lists weather events occurring in *constrained area*. |
| severity list | is recorded for each plane *category* |
| start | indicates when weather constraint was established. |
| expected end | of weather constraint. |
| velocity | indicates if weather phenomenon is moving. |
| **Miscellaneous** | |
| previous | reference to *weather constraint* decision time-frame data. |
| impacted | list of possibly impacted attendees (planes whom obtained divergence order or warning from UTM). |

Table 6.8: Static/Dynamic weather constraint for given decision time-frame.

## 6.9 (R) Rule Engine

This section is follow up of *UTM functionality definition* (sec. 6.8), outlining realization of *UTM directives* on *UAS* side (sec. 6.9.1, 6.9.2).

**Reasoning:** The *Avoidance* process and *UTM directives fulfillment* is different in every national airspace. The ICAO issues recommendation [147, 160] which are implemented by every member country, some of procedures are stricter some are implemented differently.

The *UTM* collision case calculation and procedures may be universal, but their realization by *UAS* will be heavily impacted by local legislation and procedures. The *approach* must account the need of *variable parts* of *obstacle avoidance process*. The *dynamic parts* needs to be woven to hard-coded processes.

*Note.* Please refer to *Template Programming* and *Aspect Oriented Programming* for further explanation.

**Inspiration:** There was a *Maritime Rules* implementation [170] in form of *Movement Restrictions* and *Waypoint Changes*.

### 6.9.1 (R) Architecture

**Purpose:** The *core process* of *Avoidance Grid Run* (sec. 6.7.2) and *Mission Control Run* (sec. 6.7.3) needs to be enhanced based on situation. The architecture is based on *aspect oriented approach* [171]. The key ideas and concepts are taken from rule engine implementation for multiagent navigation system [172].

**Rule Engine:** The program module to inject and run *rules* modifying standard workflow based on triggering events. The *aspect oriented* approach enables to configure rules in *run-time* via predefined process hooks - *Decision Points*.

The rules in context of this work are pieces of code which have semi-static structure consisting from following parts (fig. 6.32):

1. *Decision Point* - hook point in process where the rule can be attached/detached. If more than one rule is hooked the priority of execution needs to be defined.

2. *Context* - the *run time context* in time of *invocation* in our case the *copy* of *Mission Control*, *Avoidance/Navigation Grid* and *Collision Cases*.

3. *Parser Method* - optional helper method to parse interesting data set from *Context*. The *parsed data* have better readability.

4. *Condition Check Method* - implementation of the trigger. If the sufficient condition is met, the rule body is applied.

5. *Rule application* - calculations and data structure changes. Mainly *disabling trajectories* in *Reach Set* in our implementation.

**Configurability:** The *Rule Engine* enables real time configuration. The *Enabled Rules Table* have been implemented to enforce specific rules in specific context.

The *Rules* can be invoked from *Rule Application*, this enables effective rule chaining and piece-wise functionality split.

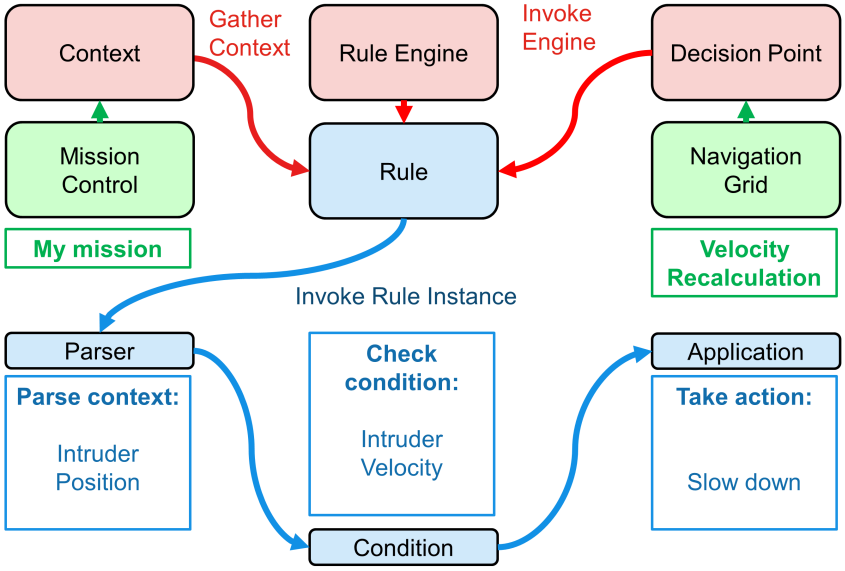



Figure 6.32: Rule engine components overview.

**Example:** The *UAS* is flying in controlled airspace. The *intruder* shows in front of *UAS*. The *UAS* is faster than *intruder*. The *UAS* tries to obtain permission for *Overtake*. The *UTM* does not allow *overtake*, because of *insufficient UAS maneuverability capability*. The *Rule* (fig. 6.32) with:

1. *Context* - UAS Mission Control, containing the actual mission goal and UAS IMU parameters.
2. *Decision Point* (Joint Point) - Navigation grid, containing projected constraints and reach set approximation.
3. *Rule is invoked:*
   a. *Parser* parses the context which is *intruder's Position Notification* containing its heading and velocity.
   b. *Condition* is checked to *relative intruder velocity*. The *evaluation* is positive, when the UAS is *pursuing intruder*.
   c. *Application* of *Rule* is last step, in this case the *UAS* will slow down.

### 6.9.2 (R) Rule Implementation

**Configuration:** The *Rule Engine Architecture* (fig. 6.32) is configured to handle *UTM functionality* for *Collision Case Resolution* (sec. 6.8.8). The overview of *Context* (Green), *Decision Points* (red) and *Rules to be Invoked* (cyan) is given in (fig. 6.33).

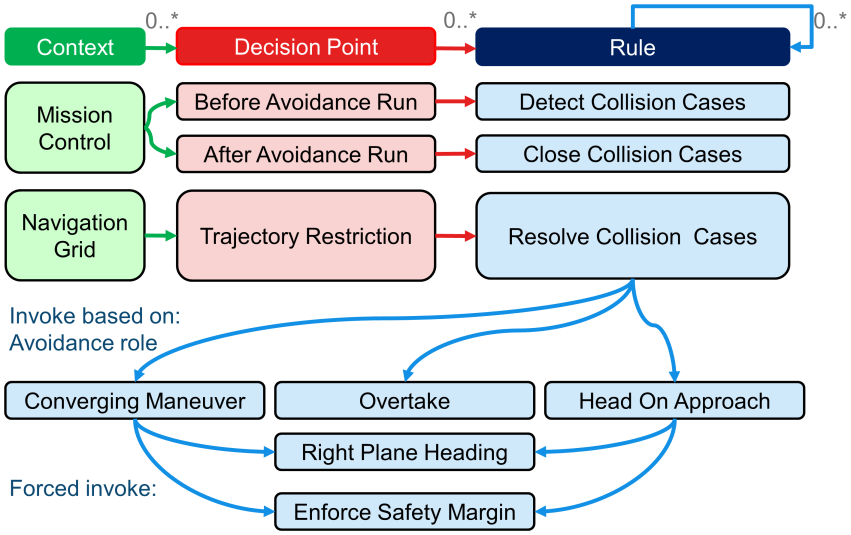



Figure 6.33: Rule engine initialization with Rules of the air.

*Note.* The *Weather Case* (sec. 6.8.9) is handled in similar manner. The mission control loop (fig. 6.25) have rules with separate *Decision Points* to enforce *hard constraints* (before step 9.) and *soft constraints* (before step 10.).

**Decision Points:** The *Decisions* are bounded to *Mission Control Run Process* (fig. 6.25) in following manner:

1. *Before Avoidance Run* (before step 7.) - Context: *Mission Control* (Received Collision Cases) - the *UTM* can send directives. It is required to find which ones are impacting our *UAS*.

2. *Trajectory Restrictions* (after step 7.) - Context: *Navigation Grid* (Trajectory Restrictions) - adaptation of *behavior* imposed by *active collision cases*.

3. *After Avoidance Run* (after step 11.) - Context: *Mission Control* (Collision Case Resolutions) - our *UAS* will update the status of *Collision Cases* then it checks the *avoidance conditions*. The *Resolution Notification* resolution notifications are sent to UTM afterwards.

**Road map:** The *implemented rules*(cyan) are separated into following categories:

1. *Management Rules* - managing collision cases (additional control flow):
   a. *Detect Collision Cases* (sec. 6.9.3) - the detection of active participation in received *collision cases* and generation of *restrictions*.
   b. *Resolve Collision Cases* (sec. 6.9.4) - the enforcement of *active avoidance roles* in *collision cases*. The one *Restriction Rule* is invoked directly.
   c. *Close Collision Cases* (sec. 6.9.5) - impact calculation and *Resolution Notification* to *UTM* authority.
2. *Restriction Rules* - restricting the *Navigation Grid* trajectories or altering *goal waypoint* based on *selected collision cases*:
   a. *Converging Maneuver* (sec. 6.9.7) implementation of *Converging Avoidance* (sec. 6.8.5).
   b. *Head On Approach* (sec. 6.9.6) implementation of *Virtual Roundabout Enforcement* (sec. 6.8.4).
   c. *Overtake* (sec. 6.9.8) implementation of *overtake maneuver* for *Overtaking plane* (sec. 6.8.6).
3. *Miscellaneous Rules* - reused pieces of code in *Head On* and *Converging Situations*:
   a. *Right Plane Heading* (sec. 6.9.9) - restrict all trajectories heading to space separated by parametric plane in *Avoidance Grid* which are heading or belonging to plane.
   b. *Enforce Safety Margin* (sec. 6.9.10) - restrict all *Trajectories Segments* which are in proximity of *Collision Point* lesser than *Enforced Safety Margin*.

### 6.9.3 (R) Rule: Detect Collision Cases

This rule is activated each *UAS avoidance run*. *UTM* sent out all related *collision cases* (6.176) based on our *UAS identifier*. Creation of *collision case* is given in sec. 6.8.8 based on air traffic periodical*position notifications* (sec 6.8.7).

$$UTM \times timeFrame \rightarrow UTMCollisionCases \tag{6.176}$$

If there are available *position notifications* (sec 6.8.7) from surrounding air-traffic, UAS will calculate own *collision cases* (6.177).

$$uasStatus \times positionNotification \times utmTimeFrame \rightarrow UASCollisionCases \tag{6.177}$$

Then UAS merges *own collision cases* with *UTM collision cases*, if there exist following disparities UAS will take action:

1. $distamce(ownCollisionPoint, utmCollisionPoint) \geq threshold$, send UTM notificaiton, use *utmCollisionPoint*
2. $utmMargin \geq ownMargin$, use safety margin from UTM.

3. $utmAvoidanceRole == active, ownAvoidanceRole == inactive$, use UTM avoidance role.

4. $utmCollisionCase == active, ownCollisionCase == uncertain$, use UTM provided collision case, not all *position notifications* are available.

5. $utmCollisionCase == inactive, ownCollisionCase == active$, notify UTM with new collision case, ignore collision case until UTM approves.

*Note. Avoidance role* is classified as *inactive* if and only if UAS has *right of the way*, it is classified as *active* otherwise.

*Safety margin* determined by UTM has priority, because not all calculations factors are available for UAS.

*Collision Case* unknown to UTM are ignored, due to safety reasons (false data spoofing), collison case is activated after UTM confirmation. If there is real intruder not confirmed by UTM it is handled via *non-cooperative* or *emergency* avoidance procedure

*Selection process* of active *collision cases* is based on UAS *avoidance role* in each *collision case.*

- If the *avoidance roles* are following: *Head On Approach, Converging Maneuver, or Overtake* in all *collision cases* UAS system will stay in cooperative mode.

- If there exists at least one *collision case* with *own avoidance role* or *intruder avoidance role* set as *emergencing*, the UAS will notify UTM and ask for *diversion order*, meanwhile it sets itself into *Emergency avoidance* mode.

- If there exist multiple *Overtake avoidance roles* or combination of *Overtake avoidance role* and *Other active role*, the UAS will decrease its cruising speed like follows:

$$UASSpeed = \max \begin{Bmatrix} minimalUASCruisingSpeed, \\ \min \{intruderSpeed\} \quad \forall activeCollisionCases \end{Bmatrix} \tag{6.178}$$

  During *slow-down* UAS switchs to *emergency avoidance mode* and asks for *divergence order* from UTM.

*Ordering of collision cases* starts if and only if the *UAS* is in *cooperative avoidance mode.* The cases are ordered for processing based on severity rating which is calculated based on:

1. *Safety Margin* - the greater safety margins are prioritized.

2. *Intruder vehicle class* - the more dangerous intruders are prioritized.

3. *Collision point distance* - closer collision points are prioritized.

4. *UAS avoidance role* - *Head on Approach* is favored upon *Converging maneuver*, due to direct collision severity.

*Rule engine invocation* for each *active collision case* is then applied on *descending severity sorted* list.

The rule is summarized in table 6.9.

*Invocation:* Every *Decision point* in *UAS main loop*

*Objective:*

1. Fetch UTM *Collision cases* for given decision time frame.
2. Create/update own *own collision cases* based on received *Position notifications* from surrounding *Intruders*.
3. Merge *Collision cases* based on *UTM priority order*.
4. Select active *collision cases* based on following conditions:
   a. *Active participation* in *collision case* where *avoidance role* $\neq$ *Right of the way*.
   b. *Collision point* is int front of UAS.
   c. *Emergency mode detection* there exist at least one non-cooperative participant.
5. Order *collision cases* based on *severity*.
6. If there is at least one *active collision case* enforce rule *Resolve collision case* (tab 6.10) for each *active collision case*.

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS Mission control, Before Avoidance Run, UTM/UAS collision cases | Clean *avoidance grid*, No emergency | Active collision case selection, Prioritization |

Table 6.9: Detect collision cases rule definition.

### 6.9.4 (R) Rule: Resolve Collision Case

*Active collision cases* are processed one by one. All collision cases are applied to *Navigation grid*. *Navigation grid* contains all possible *trajectories* in form of *Reach set*. All *trajectories* are *reachable* at the beginning of UAS *avoidance frame*. Each application of *collision case resolution* rule disables some subset of feasible *trajectories*. For this reason are *active collision cases* sorted by severity.

It is assumed that UAS is in *cooperative avoidance mode*. If previous application of this rule forced UAS into *emergency mode* the rule is not applied to save system resources. *Emergency* mode is invoked if *rule application* disable all *trajectories* in *Navigation grid*. If there is at least one *feasible trajectory* in *avoidance grid* follow-up rule is invoked based on UAS *avoidance role*.

The rule is summarized in table 6.10.

*Invocation:* This rule is invoked if exists at least one *active collision case* in given *navigation grid time-frame*, moreover *avoidance grid* must be empty and *cooperative avoidance mode* is enforced.

*Objective:* Based on *active collision case* and *UTM directives* enforce behaviour based on *own avoidance role*:

1. *Head on approach* - rule 6.12.
2. *Converging maneuver* - rule 6.13.
3. *Overtake* - rule 6.14.
4. *Emergency mode* - switch from *active avoidance mode* to *emergency mode*.

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS mission control, Trajectory restriction, Collision cases, | Active merged collision case, Resolution mandate from UTM | Enforce Rules of Air or Enforce emergency |

Table 6.10: Resolve collision case rule definition.

### 6.9.5 (R) Rule: Close Collision Cases

*Collection of rule results* detected by rule 6.9 and *resolved* by rule 6.10 is done via *context of rule engine*. For each *time-frame* and each $trajectory \in NavigationGrid$, there exists rule engine *context* query (6.179) which returns *trajectory status* and *list of applied rules on trajectory*.

$$Context(trajectory, timeFrame) \rightarrow \{State : Enabled/Disabled, Rule(s)\} \quad (6.179)$$

*Calculation of possible trajectories* in *navigation grid* is using *collected rule results* (6.179). If the *trajectory state* and linked *rule reason* is sufficient, the *trajectory* is disabled for given *time frame*. *Standard navigation algorithm* (TBD - reference to section with outer navigaiton loop) is used to select *feasible trajectory*.

*Rules of the air* and their applicaiton in *General Aviation* cases is consistent. Increasing trafic density can impose new layers of rules, which may cause the *soft deadlock* in *manuverability*. In this case *Navigation grid* will have all *possible trajectories* exhausted. Following procedure is executed:

1. UAS switch into *Non-cooperative avoidance mode* or *Emergency avoidance mode* depending on situation severity (One conflict can be handled with *vertical separation* of conflicting aircrafts).
2. UAS broadcasts *warning message* to all nearby aircrafts, and *separation message(s)* to conflicting aircraft. *Separation message* contains *expected collision point* and *preferred separation type*.Each conflicting aircraft then reacts and sends *action notification* to UTM.

3. If UAS switchs into *emergency mode*, non cooperative avoidance using *avoidance grid* is induced. Each relevant intruder is projected as *timed body volume intruder* (TBD reference to intruder probabilistic model), where *safety margin* is used as *body radius*.

*UAS* notifies *UTM* with *course change*, *planned avoidance trajectory*, *avoidance mode*. *UTM* approves planned changes or sends *plan corrections* (out of scope). The rule summary is given in table 6.11

*Invocation:* There exists at least one *active collision case* which had impact on *Navigation grid*.

*Objective:* Ensure that multiple *avoidance rules* application gives feasible *avoidance strategy*, enter into *emergency avoidance mode* otherwise. Following steps are executed:

1. *Collect rules* applied on *navigation grid* from *active collision cases*.
2. *Calculate possible trajectories for avoidance*, there may be none.
3. If there is no *feasible route*, for each *intruder* from related *collision cases*:
   a. Issue *warning message* containing *expected collision point* and *preferred separation type*.
   b. Create appropriate *intruder object* for *avoidance grid*.
   c. Calculate *evasive maneuver* based on expected *separation type*.
4. *Notify UTM* with *collision case resolution* for each *active collision case*. *Nofigy UTM* with *planned trajectory* and *avoidance mode*

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS Mission control,<br>After avoidance run,<br>Collision resolutions | At least one trajectory in<br>Navigation grid,<br>Emergency check | Force *Emergency mode*<br>OR<br>Close Collision Case |

Table 6.11: Close collision case rule definition.

### 6.9.6 (R) Rule: Head on Approach

Rule (6.12) is invoked based on *angle of approach* range condition, defined *collision case* section 6.8.8. The handling of *head on* avoidance is given in section 6.8.4.

*Virtual round-abound* for UAS and intruder is created by UTM. The center of virtual round-abound and *corrections for participants margins* are determined based on:

1. *Collision case center* - contributes to round-abound center median point.
2. *UAS and intruder maneuverability* - determines *attendants avoidance mode* and *maximal avoidance margins*.
3. *Surrounding air-traffic* - contributes to round-abound center median point, determines ideal *ideal avoidance margins* due to *wake turbulence* prevention.

*Virtual round-abound center* is calculated as *corrected median* (6.180) taking *cluster of collision cases* and calculates median of their collision points corrected by *weather* and *wake turbulence* factor.

$$correctedMedian = \sum_{c_i \in collisionCases} (c_i.center + correction) / count(collisionCases) + \\ + corrrection(Weather) + correction(WakeTurbulence) \tag{6.180}$$

*Corrected margin* needs to be calculated for each *participating aircraft*, because of the *virtual roundabout* center correction (6.180). Each *round-abound participant* is ordered based on importance (lowest maneuverability first). Then for each *round-abound participant* obtains *corrected margin* (6.181) calculated from *collision case safety margin*, corrections based on other *more important vehicles*, *weather*, *wake turbulence*.

$$correctedMargin = \min \left[ \begin{matrix} caseMargin + correction \begin{pmatrix} ImportantVehicles, \\ Weather, \\ WakeTurbulence \end{pmatrix} \\ maximalAvoidanceMargin \end{matrix} \right] \tag{6.181}$$

*Invocation:* When *UAS avoidance role* is *Head on* avoidance and *avoidance grid* is empty.

*Objective:* Ensure that *UAS body* does not enter into *intruder's well clear zone*.

1. Prevent *left-side leading* maneuvers (rule 6.15).
2. Prevent head on *safety margin* breach(rule 6.16).
3. Return to original course, when *navigation grid* is clear.
4. Prevent *wake turbulence* (by safety margin correction).
5. Enforce *Round-about* behaviour (by clustering collision cases).

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS Navigation Grid, Collision Point, Avoidance role | None | Run rules referenced in objective listing. |

Table 6.12: Head on Approach rule definition.

### 6.9.7 (R) Rule: Converging Maneuver

Rule is invoked based on *angle of approach* range defined in *collision case calculation* (sec. 6.8.8). Behaviour enforced to this rule is equal to rule 6.12 except the *intruder* stays on his original path. UAS behaviour is described in section 6.8.5. The *rule summari* is given by table 6.13.

*Invocation:* When *UAS avoidance role* is *Converging* and *avoidance grid* is empty.

*Objective:* Ensure that *UAS body* does not enter into *intruder's well clear zone.*

1. Prevent *left-side leading* maneuvers (rule 6.15).
2. Prevent head on *safety margin* breach(rule 6.16).
3. Return to original course, when *navigation grid* is clear.
4. Prevent *wake turbulence* encounter (by safety margin correction).

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS Navigation grid,<br>Collision point,<br>Avoidance role | None | Run rules from objective. |

Table 6.13: Converging maneuver rule definition.

### 6.9.8 (R) Rule: Overtake

During overtake maneuver there is our *UAS* and *Intruder* cruising at same *flight level*. *Angle of approach* ($\alpha$) is lesser than 70°. *UAS* absolute velocity is much greater than *overtaken* absolute velocity.

It is assumed that during *overtake* maneuver *overtaken* intruder will keep constant heading and velocity. If this assumption is broken *UAS* system will invoke *Emergency avoidance* procedure. *UTM* will calculate such *divergence* and *convergence* waypoints that *overtake safety condition* (6.182) is satisfied.

$$distance(uasPosition, overtakenPosition) \geq utmMargin, \forall t \in manueverTime \tag{6.182}$$

Where *utmMargin* is calculated based on *Collision case* resolution. *Main idea*is to calculate *Safe offset for Overtake maneuver*, lets have:

$$velocityDifference = \|uasVelocity - overtakenVelocity\| \quad [ms^{-1}, ms^{-1}, ms^{-1}] \tag{6.183}$$

*Decision distance* (6.184) is given as distance when *UTM mandate* takes effectiveness, its assumed that *UTM* knows *utm decision frame* [s]:

$$decisionDistance = velocityDifference \times uasDecisionFrame \quad [m, ms^{-1}, s] \tag{6.184}$$

*Overtake middle distance*(6.185) is length of hypotenuse for triangle where *positional difference* and *utm margin* for overtake are cathetuses:

$$overtakeMiddle = \sqrt{\begin{matrix}\|uasPosition - collisionPoint\|_2 + \\ + safetyMargin^2\end{matrix}} \quad [m, \vec{m}, \vec{m}, m] \tag{6.185}$$

*Safe offset* (6.186) is considered as combination of *overtake middle distance* (6.185), *decision distance* and uas *waypoint reach margin*.

$$\begin{matrix} & overtakeMiddle & \\ safeOffset = & +decisionDistance & [m, m, m, m] \\ & +waypointReachMargin & \end{matrix} \tag{6.186}$$

*Note. Waypoint reach margin* [m] is property of own *UAS navigation algorithm*. It represents maximal distance of vehicle position and waypoint at time when waypoint is considered reached.

*Local coordinate frame*: UAS and Overtaken are in Local coordinate frame heading in $X^+$ axis direction ($X^+$ front of aircrafts, $X^-$ back of vehicles, $Y^-$ right side, $Y^+$ left side, $flightLevel \rightarrow Z = 0$), Collision Point is considered as $\vec{0}$,

*Divergence point* (6.187) in local coordinates is given as right offset of *(*UTM margin) and *decision distance*:

$$divergence = \begin{bmatrix} 0 \\ -decisionDistance - utmMargin \\ 0 \end{bmatrix} \quad [\vec{m}, m, m] \tag{6.187}$$

*Convergence point* (6.188) in local coordinates is given frontal *safe offset* (6.186) and right offset of *UTM margin* and *decision distance*:

$$convergence = \begin{bmatrix} safeOffset \\ -decisionDistance - utmMargin \\ 0 \end{bmatrix} \quad [\vec{m}, m, m] \tag{6.188}$$

*Convergence* (6.189) and *Divergence* (6.190) waypoint in global coordinate frame is obtained via transformation function $R_{XYZ}$ as follow:

$$\begin{aligned} divergenceWaypoint = & collisionPoint \\ & + R_{XYZ}(overtakenOrientation, divergence) \end{aligned} \tag{6.189}$$

$$\begin{aligned} convergenceWaypoint = & collisionPoint \\ & + R_{XYZ}(overtakenOrientation, convergence) \end{aligned} \tag{6.190}$$

*Overtake rule* is summarized in table 6.14.

*Invocation:* Invoked by rule *Collision Case Resolution* (rule 6.10)

*Divergence Waypoint* (6.189): waypoint to diverge from original UAS path to ensure Intruder safety, with unchanged intruder velocity and heading.

*Convergence Waypoint* (6.190): waypoint when convergence to original UAS path is enabled, within unchanged intruder velocity and heading.

*Objective:*

1. Calculate *Divergence Waypoint* and *Convergence Waypoint*.
2. Enforce Divergence/Convergence waypoint during avoidance.

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS Navigation Grid,<br>Collision Point,<br>Avoidance Role | $UASVelocity$<br>$>>$<br>$IntruderVelocity$ | Calculate & Enforce:<br>• Divergence waypoint,<br>•Convergence waypoint |

Table 6.14: Overtake rule definition.

### 6.9.9 (R) Rule: Right Plane Heading

There is need to check if *trajectory* is heading to *right side* from *collision point*. For this purpose we need to define *separation plane in 3D environment*. *Separation plane* will be defined according to Samuelson *hyperplane separation theorem* [173].

*Separation plane* (6.191) is defined by three points in *global coordination frame*:

1. *UAS Position* which is fixed to given *time-frame*.
2. *Collision point* which is not equal to *uas position* by definition.
3. *Gravitational acceleration* vector fitted to *UAS position* and orthogonal to vector ($uasPosition \rightarrow collisionPoint$).

The properties of these three points guarantees that $scale.usasPosition \neq scale.collisionPoint \neq scale.gravitationalAcceleration$ for any linear $scale \neq 0$.

$$SeparationPlane = Plane \begin{pmatrix} uasPosition, collisionPoint, \\ loc2glob(uasPosition, gravitationalAcceleration) \end{pmatrix} \tag{6.191}$$

*Separation plane* (6.191) in *right-hand coordinate frame* where $center = uasPosition$ $X^+$ is given by vector $\vec{x^+}$ ($uasPosition$, $collisionPoint$) and $Z^-$ is given by vector $\vec{z^-}$ ($uasPosition$, $gravitationalAcceleration$). Then *right subspace* can be defined as all points where $y \leq 0$ and *left subspace* as all points where $y > 0$.

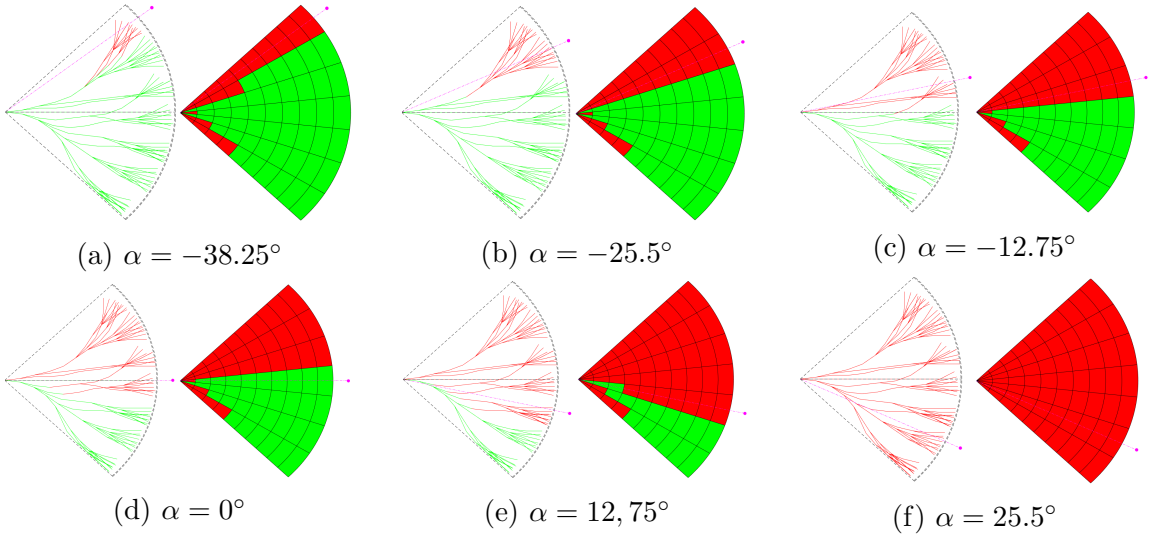
*Reach set* contains *trajectories*, minimal dataset for trajectory is time-series of *position* and *heading* regardless *underlying nonlinear model*. Let us have *transformation function* which can map UAS *position* and *heading* into *separation plane coordinate frame*.

The *first condition* (6.192) says that each trajectory *point* must lie within *right subspace*.

$$\forall position \in trajectory, \quad position \in rightSubspace \tag{6.192}$$

The *second condition* (6.193) needs to be applied for each *decision point*, when *trajectory* can be re-planned. It must be ensured that in time of reaching *decision point* vehicle is not heading into *left subspace* with given *turning time horizon*. The *minimal information* contains heading (velocity) vector. Checking if linear projection from *position* point with *heading* in given time-frame $[0, horizon]$ is sufficient.

$$\forall t \in [0, horizon], \quad (position + velocity * t) \in rightSubspace \tag{6.193}$$



(a) $\alpha = -38.25°$ (b) $\alpha = -25.5°$ (c) $\alpha = -12.75°$

(d) $\alpha = 0°$ (e) $\alpha = 12,75°$ (f) $\alpha = 25.5°$

Figure 6.34: Right plane heading rule evaluation for various angles of approach $\alpha$.

Figure 6.34. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left subfigure). These trajectories are divided according to *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. Situation is shown for various *collision point angles of approach* $\alpha$.

Rule for right plane heading check is summarized in table 6.15.

*Invocation:* Invoked by other *maneuver rules*.

*Objective:* Disable all *trajectories* in *Navigation grid's reach set* which are:

1. *Heading into collision zone*
2. *Leading into collision zone*

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS Navigation Grid, Collision point (LOC) | There are feasible trajectories in Navigation Grid. | Disable trajectories in Navigation Grid. |

Table 6.15: Right plane heading rule definition.

### 6.9.10 (R) Rule: Enforce safety margin

Rule 6.15. checks right plane heading for single mass point along *trajectiories*. Rule needs to account *body mass* of *intruder* and UAS, other factors like safe distance, regulations etc. All mentioned factors are included in *safety margin*. *Safety margin* is applied as *radius ball* around *collision point*.

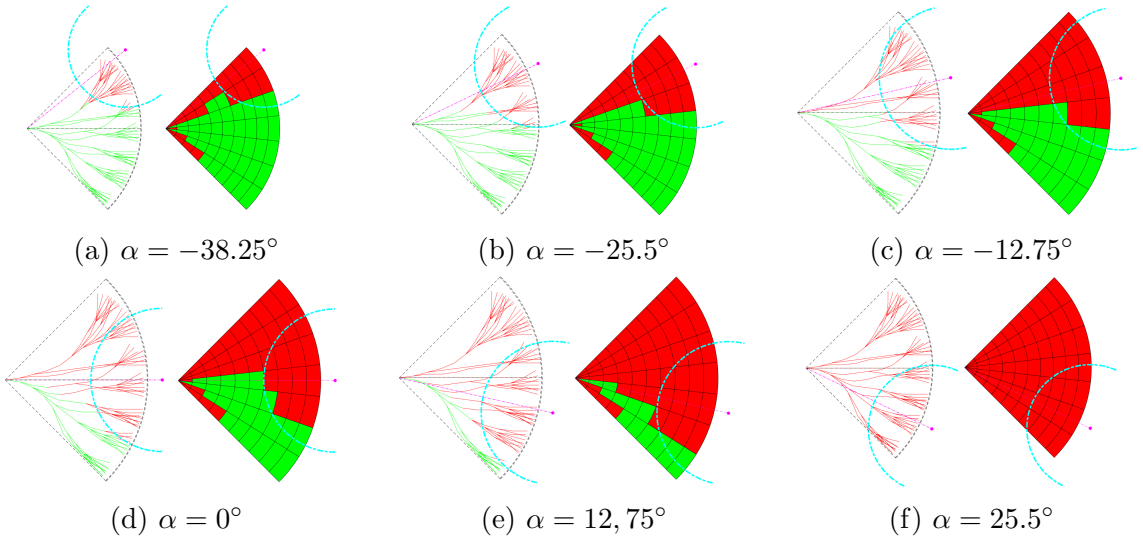
*Collision point* can be mapped from *global coordinate frame* to *reach set coordinate frame*, based on UAS *position and orientation* in *decision time*. Then comparison of distance between *collision point* and every *trajectory decision point* is trivial.

*Trajectory feasibility condition* for *non-controlled airspace* (6.194) is given as follow:

$$\forall position \in trajectory, \quad distance(position, collisionpoint) \geq safetyMargin \tag{6.194}$$

*Controlled airspace* must maintain *well clear condition*. To enforce protective barrel around *collision point* one must compare *global coordinates*. *Trajectory feasibility condition* for *controlled airspace* (6.195) is given as follow:

$$\begin{aligned} &\forall position \in trajectory, \\ &\quad XYdistance(position, collisionPoint) \geq safetyMargin \\ &\quad flightLevelStart \geq Z(position) \geq flightLevelEnd \end{aligned} \tag{6.195}$$



(a) $\alpha = -38.25°$ (b) $\alpha = -25.5°$ (c) $\alpha = -12.75°$

(d) $\alpha = 0°$ (e) $\alpha = 12,75°$ (f) $\alpha = 25.5°$

Figure 6.35: Enforce safety margin rule evaluation for various angles of approach $\alpha$.

Figure 6.35. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left subfigure). These trajectories are divided according to *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). More trajectories are disabled due to *safety margin* (teal dashed line) around the *collision point*. *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. Situation is shown for various *collision point angles of approach* $\alpha$.

Rule for safety margin check is summarized in table 6.16.

| |
|---|
| *Invocation:* Invoked by other *maneuver rules*.<br><br>*Objective:* Based on type of airspace, for given *collision point* and *safety margin* disable trajectories in:<br><br>1. Ball radius for *non-controlled* airspace (6.194).<br><br>2. Well clear barrel *controlled* airspace (6.195). |

| *Context* | *Condition* | *Application* |
|---|---|---|
| UAS Navigation Grid<br>Collision point<br>Safety Margin | There are feasible trajectories for condition application. | Disable trajectories in Navigation Grid. |

Table 6.16: Enforce safety margin rule definition.

# Chapter 7

# Simulations

The chapter presents the set of simulations developed according to a test plan (sec. 7.1). Test configuration (sec. 7.2) targets at exercising and evaluating proposed framework. The test cases are grouped in following sections:

1. *Non-cooperative test cases* (sec. 7.3).
2. *Cooperative test cases* (sec. 7.4).
3. *Test cases conclusion* (sec. 7.5).
4. *Reach set approximation performance tests* (sec. 7.6).

## 7.1 Test Plan

The *Avoidance requirements* are given in (sec. 4.5), namely:

1. *Safety Margin Enforcement* (sec. 4.5.3) - keep UAS safe depending on situation.
2. *Path Tracking* (sec. 4.5.2) - track mission given by set of *waypoints* in the manner of*Energy Efficiency* (sec. 4.5.1).

These are given as nominal behaviour (sec. 6.7.2),further enhanced by rule-based behaviour (sec. 6.9.2).

The *Navigation requirements*, out of this scope, are given in (sec. 4.6). These are satisfied by *Mission Control Run* (sec. 6.7.3).

### 7.1.1 Testing approach

The purpose of this section is to show complex scenarios, not unit testing of framework functionality. The focus is on *border line* cases for typical situations in *expected environment*. The *mode switch* between *Navigation* and *Emergency Avoidance*.

The *Tests* are designed to focus on particular functionality in specific *operational environment* with main *obstacle/weather/intruder feature* with environment induced *constraints*. There is also *UTM* factor and *Navigation penalty*.

**Operational Environment** is classified according to:

1. *Operation space* - important for *Low Altitude Operations*, the difficulty of *Avoidance Maneuvers* is proportionally increasing with *Obstacle density*. There are following main categories

   a. *Rural environment* - the relief and man-made structures are sparsely spread around the *operation space*, the UAS is operating on *very low altitude* ($\leq$ 50 feet).

   b. *Urban environment* - the concentration of the man made structures is much higher and they are more incorporated info land relief pattern, the UAS is operating on *very low altitude*.

   c. *Open air* - the concentration of ground structures is very low, the concentration of *cooperative* and *non-cooperative intruders* is increased, the UAS is operating in altitude ranging from *50 feet* to *space border*. This bring us to:

2. *Airspace category* - when *Operation Space* pattern is categorized as *Open air* and depending on *altitude above mean sea level*. The UTM is *designed authority* for controlled airspace in current *F/G class airspace*.

   a. *Controlled* - Open air where authority is present. The cases when *Authority* is not enforced due the UTM malfunction, *C*2 link loss or other cause are not considered.

   b. *Non-Controlled* - Open air operation space where is no central arbiter to determine or enforce traffic attendants behaviour.

*Static obstacles:* Static obstacles with various features detectable by main *LiDAR* sensor. The main purpose is to show avoidance capabilities combined with heavy restrictions imposed by *soft* and *hard* constraints. The original purpose of our approach was to provide robust framework for static obstacle avoidance. Three tests with increasing obstacle density and navigation complexity are delivered.

*Operational space constraints* depends mainly on *operational environment*. The standard set of constraints were taken into account for our test cases:

1. *Rural, Urban environment (low altitude)* are: geo-fencing zones, ground (hard constraints), non-controlled airspace altitudes (soft constraints).

2. *Non-controlled airspace constraints (open air)* are: geo-fencing zones (hard constraints), restricted airspace (hard constraint), weather (soft/hard constraint), controlled airspace (hard constraint), very low altitude border (soft constraints).

3. *Controlled airspace constraints (open air)* are: restricted airspace (hard constraint), weather (soft/hard constraint), non-controlled airspace boundary (hard constraints), UTM Directives (hard constraints).

**Air Traffic Attendants:**

1. *Non-cooperative UAS* (Intruder) - there are some intruders with some degree of authority, size and *severity*. There were three test cases for non-cooperative intrudes. Non-cooperative Intruders can be categorized as follow based on behaviour:

   a. *Chaotic* intruders usually have tendency to behave unpredictable, for example bird or *UAS in distress*, for this type of intruders *Maneuver Uncertainty Intersection Model* is used (sec. 6.6.4).

   b. *Harmonic* intruder usually follow long straight paths, for example UAS converging to waypoint, for this type of intruder *Body Volume Intersection Model* is used. (sec. 6.6.3).

   *Cooperative UAS* (Intruder) - there are cooperative intruders which are obeying authority (UTM) or follow *common consensus*. The work focus on *UTM* authority implementation in four test cases. These test cases are reflecting the traffic management situations essential for successful UTM collision management

**Weather** impose *soft* and *hard* space constraint, which can be moving or static. The *soft constraint avoidance* is covered by *hard constraint avoidance*. The *static constrained area* is covered by *static obstacle avoidance* capability due the *Data fusion procedure* [138]. The only case which is not covered is *Moving constrained area*, small constraints can be covered by intruder models. The ideal candidate is *storm*, because it covers quite large area, the clouds are constantly moving and severity is changing with time.

**UTM:** The *UAS Traffic Management* service should be implemented in *controlled airspace* by 2035. It is necessary to study impact of UTM services on the *Detect and Avoid* systems like ours.

The most basic service is *Identity provider* which should be implemented by 2020.

Then there is *location services*, which are necessary for coordinated collision avoidance, these were implemented in our solution up to necessary level for *Rules Of the Air* implementation.

*Mission tracking* is service tracking deviations from *declared mission plan* and *actual execution*. This statistics were used in all tests to track deviations from reference trajectory.

*Directives* for *Traffic management* and *Collision prevention* are implemented as functional life cycle of *Position notification* (sec. 6.8.7), *Collision Case* (sec. 6.8.8) for UTM. The directive handling is implemented as *Rule engine* (sec. 6.9.2) on UAS side.

**Navigation:** Navigation algorithm is depending on *Navigation mode*. UAS is usually in *Navigation mode* most of the time, despite this fact, UAS was forced into *Emergency Avoidance Mode* most of time in test cases. The navigation complexity have been divined into following categories:

1. *Open space* - UAS has visibility to goal waypoint most of the time, there are no traps.
2. *Hidden waypoint* - UAS does not have visibility to goal waypoint, most of the time, there irregular traps sometime.
3. *Maze solving* - UAS line of sight for goal waypoint is hindered by multiple obstacles, there are irregular traps often.
4. *Rule following* - UAS navigation capabilities are constrained by rule enforcement.

### 7.1.2 Test Cases Summary

*Test cases* are summarized in (tab. 7.1).

| *Test Case Name* | *Operational Environment* | *Air Traffic Attendants* | *Weather* | *UTM* | *Navigation* | *Scenario* |
|---|---|---|---|---|---|---|
| Building Avoidance | Non-controlled (Rural) $4 \times$ *buildings* | - | - | - | Open space | Fly mission around four buildings |
| Slalom | Non-controlled (Rural) $14 \times$ *buildings* | - | - | - | Hidden waypoint | Navigate to hidden waypoint |
| Maze | Non-controlled (Urban) $30 \times$ *buildings* | - | - | - | Maze structure | Solve maze with multiple curves |
| Storm | Non-controlled (Rural) $0 \times$ *buildings* | - | Storm | - | Open Space | Avoid approaching storm |
| Emergency Converging | Non-controlled (Open air) | Non-cooperative UAS (1x) | - | - | Open Space | Converging situation resolution w. o. UTM |
| Emergency Head on | Non-controlled (Open air) | Non-cooperative UAS (1x) | - | - | Open Space | Head on situation resolution w. o. UTM |
| Emergency Multiple | Non-controlled (Open air) | Non-cooperative UAS (3x) | - | - | Open Space | Multi collision case resolution w. o. UTM |
| Rule-based Converging | Controlled (Open air) | Cooperative UAS(1x) | - | Full | Follow Rules | Converging situation resolution with UTM |
| Rule-based Head on | Controlled (Open air) | Cooperative UAS(1x) | - | Full | Follow Rules | Head on situation resolution with UTM |
| Rule-based Multiple | Controlled (Open air) | Cooperative UAS(3x) | - | Full | Follow Rules | Multi collision case resolution with UTM |
| Rule-based Overtake | Controlled (Open air) | Cooperative UAS (1x) | - | Full | Follow Rules | Overtake by UAS different speed ratio |

Table 7.1: Test Cases Summary.

### 7.1.3 (R) Performance Evaluation

**Evaluation method:** *Test cases* were evaluated according to performance requirements defined in (sec. 4.5). The method was tracking critical parameter for *Safety* (sec . 4.5.3) (primary) and *Trajectory Tracking* (sec. 4.5.2) (secondary) including *Energy Efficiency* (sec. 4.5.1).

**Safety Margin Performance Evaluation:** The *safety of UAS* is main concern of *DAA system*. The common concept of *safety margin* is evaluated.

The *threat* is multidimensional, there are often multiple *static obstacles, intruders* or *weather constraints*. To reduce the multidimensional threats to one dimensional value *crash distance* concept is used:

$$crashDistance(t) = distance(UAScenter(t), threat)$$
$$\text{where } \textit{selection criterion} \text{ is:}$$
$$\min \left\{ \begin{pmatrix} distance(UAScenter(t), threat) - \dots \\ \cdots - threat.SafetyMargin \\ : \forall threat \in KnownWorld(t) \end{pmatrix} \right\} \quad (7.1)$$

The *crash distance* (eq. 7.1) for given time is evaluated as shortest distance between UAS center and threat. The threat origins from known world (sec. 4.1.10). The *threat* have safety margin. The distance to safety margin is used as prioritization criterion in our test cases (tab. 7.1).

The *safety margin* evolution over time (eq. 7.2) is calculated similar to *crash distance*. The most dangerous threat is selected based on *distance to safety margin* criterion. The value of *safety margin* property is then used.

$$safetyMargin(t) = threat.SafetyMargin$$
$$\text{where } \textit{selection criterion} \text{ is:}$$
$$\min \left\{ \begin{pmatrix} distance(UAScenter(t), threat) - \dots \\ \cdots - threat.SafetyMargin \\ : \forall threat \in KnownWorld(t) \end{pmatrix} \right\} \quad (7.2)$$

The *distance to safety margin* (eq. 7.3) is calculated as a difference between *crash distance* (eq. 7.1) and *safety margin* (eq. 7.2). The *acceptance criteria* for safety is *distance to safety margin* $\geq 0$.

$$distanceToSafetyMargin(t) = crashDistance(t) - safetyMargin(t) \geq 0 \quad (7.3)$$

*Note. On Signed Distance:* The most works are using *unsigned distance*. This work considers the *signed distance* with following intervals:

1. + (away from margin).
2. 0 (touching margin with UAS edge).
3. - (inside margin - crash/collision/broken boundary).

**Distance to Safety Margin** peaks are measured:

1. *Minimal* distance to safety margin indicates if *acceptance criterion* (eq. 7.3 is met).
2. *Maximal* distance to safety margin indicates the future *minimal detection range*. All scenarios were considered as borderline cases.

**Trajectory Tracking Evaluation** is secondary priority after safety, following parameters were checked:

1. *Waypoint reach* - the *Mission* (4.6) is considered as successfully completed if and only if $\forall$ waypoints are reached and in given order (check output of 4.7). Moreover if there is multiple UAS, each must met condition.
2. *Acceptable deviation* - for *tracking problem* (eq. 4.34) is a trajectory which in addition to *basic obstacle problem* (sec. 4.2) keeps deviation from *reference trajectory* under certain threshold (eq. 4.35).

*Trajectory tracking deviation threshold* (eq. 7.4) is defined as double of maximal distance between *goal waypoint* and *previous waypoint*.

$$trackingDeviationTreshold = 2 \times distance(goalWaypoint, previousWaypoint) \quad (7.4)$$

*Note.* If *goal waypoint* is first in *mission*, the *UAS initial condition* is considered as a *previous waypoint*.

**Computation Load:** There is theoretical definition of *intersection models* for *static obstacles and constraints* (sec. 6.5), *moving obstacles and constraints* (sec. 6.6), *avoidance run* (sec. 6.7.2), *mission control* (sec. 6.7.3) computation complexity.

The practical application requires to measure *computation load* in constrained environment. Let say that *avoidance framework* is running on stand alone embedded computer with 1.2 Ghz processor and 1GB of dedicated RAM. This is simulated by *virtual machine*.

The *simulations* were executed in *Matlab/Simulink* environment[1] using: *UTM*[2], *Navigation loop* [3], *Avoidance grid*[4] and *Reach set*[5] implementations.

The *decision frame* length is set to $1s$ which gives *computation load* (eq. 7.5). The *computation load* represents the portion of *previous decision frame* used to current decision frame calculation.

$$computationLoad = \frac{computationTime(frame)}{decisionFrameDuration} \times 100, \quad [\%; s, s] \quad (7.5)$$

*Note. computation load* is depending on actual situation, when the UAS is in *navigation mode* it should be low, when the UAS is in *clustered environment* it should be high.

Matlab implementation is quite ineffective, the Python/C++ implementation can give better results.

[1]Prototype framework implementation: `https://github.com/logomo/Feature-based-ACAS/`
[2]UTM class: `.../UavTraficManagement/UTMControl.m`
[3]Navigation Loop main class: `.../MissionControl/MissionControl.m`
[4]Avoidance Grid class: `.../AvoidanceGrid/AvoidanceGrid.m`
[5]Reach set tree class: `.../AvoidanceGrid/PredictorNode.m`

For *computational feasibility* there is *implicit* acceptance criterion (eq. 7.6): the computation of feasible path for *this time-frame* must end in *previous time-frame*.

$$\forall time \in Mission: \quad computationLoad < 100\% \tag{7.6}$$

## 7.2 Testing configuration

All *simulations* are run with configuration described in this *section*. The UAS used for the purposes is given by *model and control* (sec. 6.2).

*UAS parameters:* An *UAS system* (tab. 7.3) is modeled after small scale toy model with: maximal body radius 30 $cm$, maximal speed 4 $m.s^{-1}$,weight 450 $g.$, maximal flight duration 20 $min$, maximal turning rate 15 $deg.s^{-1}$. The *body margin* is set to $0.3m$, the *near miss radius* is double of *body margin*, thus 0.6 $m$, the *well clear radius* is set to 5 $m$. Margins can be set to any value if they are complaint with condition (7.7).

$$0 < bodyMargin \leq nearMissRadius \leq wellClearRadius \leq gridDistance \tag{7.7}$$

*Note. Safety margin* is broad term used to describe *minimal distance* between UAS and *adversarial object*. The *Safety margin* is:

1. *near miss radius* in case of *non-controlled airspace* or *emergency avoidance mode*.
2. *well clear radius* in case of *controlled airspace* and *navigation mode*.

*Decision time:* Decision time can be set by the user to any positive non-zero value (7.8). The *Decision time* is equal 1 $s$ and *Decision frames* are synchronized.

$$maxAlrogithmCalculationTime \leq decisionTome \leq \infty \tag{7.8}$$

*Speed:* For *all movements* constant speed 1 $m.s^{-1}$ is used. Speed can be changed to any value in given boundary (7.9).

$$0 \leq speed \leq \min \begin{pmatrix} 0.5 \times (navigationGrid.distance/decisionFrame) \\ 0.5 \times (avoidanceGrid.distance/decisionFrame) \end{pmatrix} \tag{7.9}$$

*Movement automaton:* The *movement set* is given in (tab. 7.2). The *movement* set contains horizontal, vertical, and, combined movements.

*Grids:* Used *Navigation grid parameters* are given in (tab. 7.4).Selected *Navigation Reach set* is *ACAS-like* with enabled horizontal/vertical separation. Used *Avoidance grid parameters* are given in (tab. 7.5). Selected *Avoidance Reach set* is *combined* because of high *coverage ratio*.

User can define own grid parameters according to the *space discretization rules* (sec. 6.3) and chose own *reach set type* according to preference (sec. 6.4).

| Movement | Roll | Pitch | Yaw |
|---|---|---|---|
| Straight | 0° | 0° | 0° |
| Left | 0° | 15° | 0° |
| Right | 0° | −15° | 0° |
| Up | 0° | 0° | −15° |
| Down | 0° | 0° | 15° |
| UpLeft | 0° | 15° | −15° |
| UpRight | 0° | −15° | −15° |
| DownLeft | 0° | 15° | 15° |
| DownRight | 0° | −15° | 15° |

Table 7.2: Movement orientations.

| UAS parameters | |
|---|---|
| speed | $1\,ms^{-1}$ |
| horizontal turning r. | $3.82\,m$ |
| vertical turning r. | $3.82\,m$ |
| body radius | $0.3\,m$ |
| near miss r. | $0.6\,m$ |
| well clear r. | $5\,m$ |

Table 7.3: *UAS* parameters.

| Navigation Grid | |
|---|---|
| type | ACAS-like |
| distance range | $0 - 10\,m$ |
| layer step | $1\,m$ |
| horizontal range | ±45° |
| horizontal cells | 7 |
| vertical range | ±30° |
| vertical cells | 5 |

Table 7.4: *Navigation Space* parameters.

| Avoidance Grid | |
|---|---|
| type | combined |
| distance range | $0 - 10\,m$ |
| layer step | $1\,m$ |
| horizontal range | ±45° |
| horizontal cells | 7 |
| vertical range | ±30° |
| vertical cells | 5 |

Table 7.5: *Avoidance Space* parameters.

| Coloring | | |
|---|---|---|
| Airc. | Executed | Planned |
| UAS 1 | blue | red |
| UAS 2 | cyan | magenta |
| UAS 3 | green | yellow |
| UAS 4 | black | green |

Table 7.6: *UAS* coloring.

## 7.3 Non-cooperative test cases

The *main* goal of this section is to show operative capabilities for *non-cooperative* avoidance mode in *emergency* and *solo situations*.

Test avoidance capabilities against *static obstacles*, *non-cooperative intruders*, *moving hard constraints* are covered.

Coverage of the *soft constraints*, *map obstacles* and *detected obstacles* are implicitly covered due the properties of *safety* and *body* margins (tab. 4.2).

1. *Building Avoidance* (sec. 7.3.1) covers *static obstacles* explicitly and *map obstacles, hard constraints, ground avoidance* implicitly.

2. *Slalom* (sec. 7.3.2) covers *open space navigation capabilities*, showing the determinism of the *avoidance loop run*, in addition to *building avoidance*.

3. *Maze* (sec. 7.3.3) covers *closed space navigation capabilities*, showing the higher level navigation properties of primitive *right-side* 2D maze solver. The main point is to show possibility to enrich the *Navigation loop algorithm* (fig. 6.25).

4. *Storm* (sec. 7.3.4) covers *hard moving constraints avoidance* explicitly and *hard static constraints*, *soft static constraints*, *soft moving constraints* implicitly.

5. *Emergency converging* scenario (sec. 7.3.5) covers *non cooperative intruder with right of the way* avoidance capability.

6. *Emergency head on* scenario covers (sec. 7.3.6) *non cooperative intruder without right of the way* avoidance capability

7. *Emergency mixed* scenario (sec. 7.3.7) covers *multiple intruders with/without right of the way* avoidance capability.

### 7.3.1 Building avoidance

**Scenario:** The *UAS* is flying the mission given by (tab. 7.7) in the *open space environment*. There exists a map of obstacles with defined *safety and body margins*. *Reference trajectory* (direct interconnection of waypoints) is going trough partially known space with some charted obstacles.

| Position | | Waypoints | | | |
|---|---|---|---|---|---|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | $\mathcal{WP}_1$ | $\mathcal{WP}_2$ | $\mathcal{WP}_3$ | $\mathcal{WP}_4$ |
| $[0, 0, 0]^T$ | $[0°, 0°, 0°]^T$ | $[100, 0, 0]^T$ | $[100, 100, 0]^T$ | $[0, 100, 0]^T$ | $[0, 0, 0]^T$ |

Table 7.7: Mission setup for *Building avoidance* scenario.

**Obstacle set:** Obstacles are discovered during a flight by *UAS LiDAR sensor*, the set of obstacle is defined in (tab. 7.8).

| Obstacle | | | Body Margin | | | Safety Margin |
|---|---|---|---|---|---|---|
| id | position | type | min. | max. | avg. | |
| 1 | $[0, 0, 0]^T$ | polygonal | 14 | 20 | 16 | 5 |
| 2 | $[100, 50, 0]^T$ | hospital | 12 | 18 | 14 | 7 |
| 3 | $[50, 100, 0]^T$ | unusual | 10 | 20 | 15 | 8 |
| 4 | $[0, 50, 0]^T$ | square | 18 | 20 | 19 | 4 |

Table 7.8: *Obstacle set* for *Building avoidance* scenario.

**Main Goal:** Show *static obstacle avoidance capability* in *open space environment*, using *LiDAR scanning* and *obstacle map* as the *information sources*.

**Acceptance criteria:**

1. Proper *algorithm mode switch*:
   a *Avoidance mode* is active when the *UAS* is in close proximity of obstacle (*distance* $(obstacleCenter, UASPosition) \leq 20m$).
   b *Navigation mode* is active when the *UAS* is further away from any obstacle (UAS is actively converging to *goal waypoint*);
2. *Minimal safety margin distance* $\geq 0m$
3. *Reach each waypoint* (tab. 7.7) in given order.

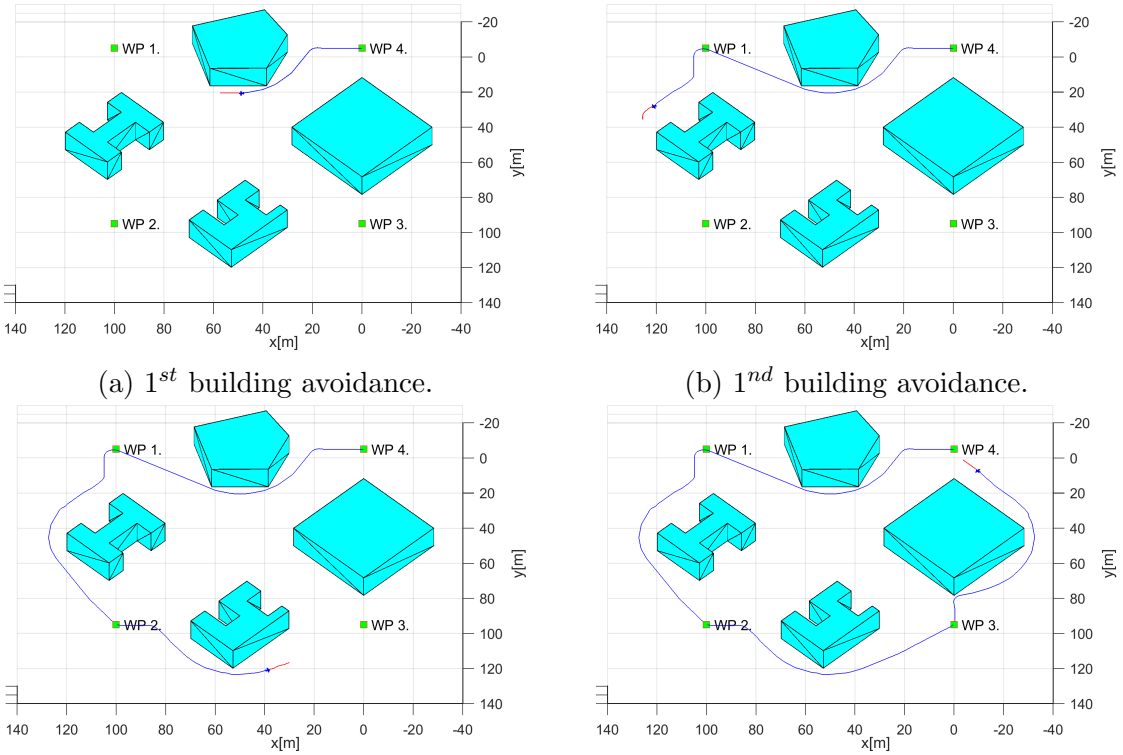
**Testing Setup:** The *standard test setup* defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Avoidance grid - type - ACAS-like* with enabled *Horizontal maneuvers*

*Note.* Enforced *safety margin* does not exceed the *avoidance grid range* (10 $m$). The concept of *Static obstacle avoidance* is in detail discussed in the *progress report* [138].

**Simulation Run:** Notable moments from the *simulation run* (fig. 7.1) are following:

1. $1^{st}$ *building avoidance.* (fig. 7.1a) - UAS avoids the building from left side, because overall trajectory cost is cheaper. The first building is convex obstacle.
2. $2^{nd}$ *building avoidance.* (fig. 7.1b) - UAS avoids the building from right side, while avoiding active non convex portion of the building.
3. $3^{rd}$ *building avoidance.* (fig. 7.1c) - UAS avoids the building from right side, missing both traps from it.
4. $4^{th}$ *building avoidance.* (fig. 7.1d) - UAS avoids the building from right side. This building is also convex obstacle.




(a) $1^{st}$ building avoidance.

(b) $1^{nd}$ building avoidance.

(c) $3^{rd}$ building avoidance.

(d) $4^{th}$ building avoidance.

Figure 7.1: Test scenario for *Building avoidance* (static ground obstacles).

**Distance to Body/Safety Margin Evolution:** The distance of *UAS* center to nearest obstacle (blue) does not broke a *safety margin* (of closest obstacle (yellow) nor *body margin* of closest obstacle (red) as it can be seen in (fig. 7.2). *Acceptance condition* for *algorithm mode switch* can be shown by UAS *active avoidance of obstacles.*

*Note.* The *body* and *safety margins* are changing depending on *UAS position and orientation*, is changing reflecting (tab. 7.8) margins.

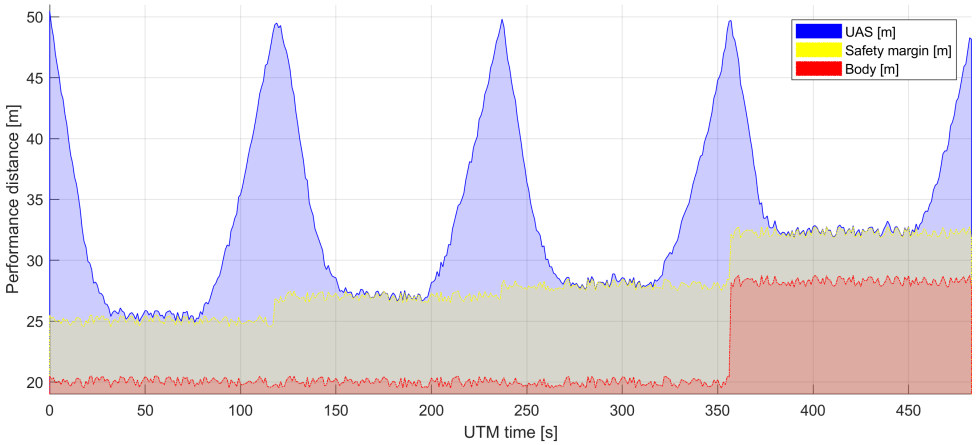



Figure 7.2: Distance to body/safety margin evolution for *Building avoidance scenario*.

**Distance to Body/Safety Margin Peaks:** Minimal distance to *safety margin* is 0.69 $m$. The *minimal distance to obstacle body* is 4.69 $m$ which is more than sufficient for tested UAS type. *Safety margin acceptance criteria* have been achieved, because minimal distance is greater than zero. The minimal *body margin distance* is 4.69 $m$ for obstacle no. 4 (tab. 7.8).

| Parameter | | UAS 1 |
|---|---|---|
| Distance to Safety Margin | min | 0.69 |
| | max | 24.98 |
| Distance to Body Margin | min | 4.69 |
| | max | 29.98 |

Table 7.9: Distance to Body/Satety Margin Peaks for *Building avoidance scenario*.

**Path Tracking Performance:** Reference path (green dashed line) is given as direct interconnection between waypoints (green numbered square). The real trajectory (blue solid line) is split into its XYZ components. *All mission waypoints* (fig. 7.3) have been reached in given order. There are some deviations on $X - Y$ horizontal axes, while the UAS was in the *avoidance mode*.

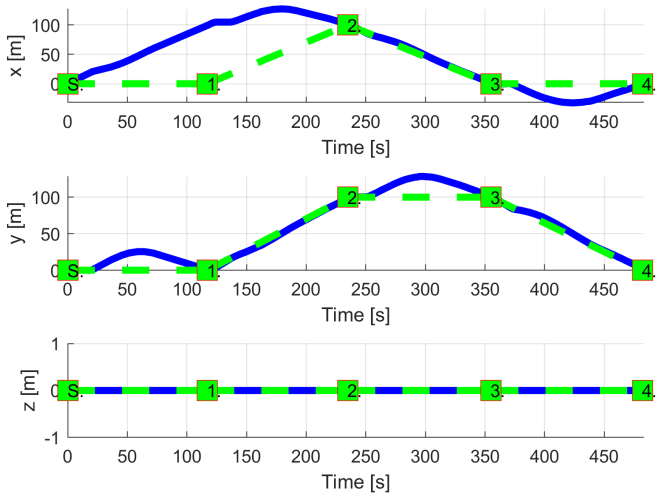



Figure 7.3: *Building avoidance* path tracking.

**Path Tracking Deviations:** Deviations (tab. 7.10) from *reference trajectory* are in expected ranges considering *mission plan* (tab. 7.7) and *obstacle properties* (tab. 7.8).

| Param. | UAS 1 | | | |
|---|---|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}$ | $\mathcal{WP}_3$ | $\mathcal{WP}_4$ |
| $\max \lvert x \rvert$ | 104 | 86 | 5.34 | 32.52 |
| $\max \lvert y \rvert$ | 25.39 | 6.59 | 28.2 | 4.55 |
| $\max \lvert z \rvert$ | 0 | 0 | 0 | 0 |
| $\max dist.$ | 107.05 | 86.2 | 28.7 | 32.84 |

Table 7.10: Path tracking for properties *Building avoidance*.

**Computation Load:** The *computation load* for *scenario* (fig.7.4) shows used time (y-axis) over decision frame (x-axis).

There is slight increase in *computation time* when UAS is in *Emergency Avoidance Mode*.

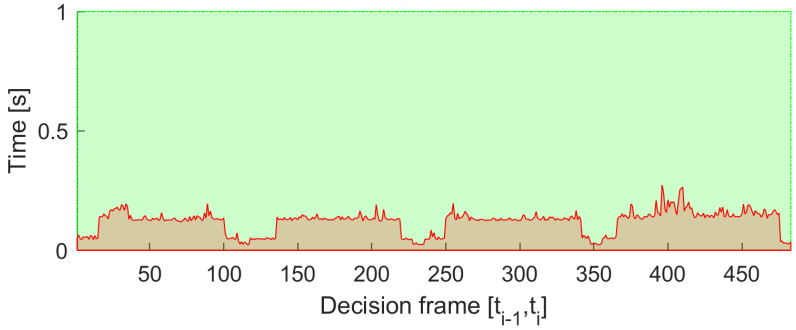



Figure 7.4: Computation time for *Building avoidance* scenario.

### 7.3.2 Slalom

**Scenario:** The *UAS* is flying the mission given by (tab. 7.11) in the *open-space environment*. A Operational space is more clustered than in case of *Building Avoidance* (sec. 7.3.1). There map of notable *buildings* with defined *safety and body margins* imposing additional flight constraints. The *UAS* is flying through partially known space with some charted obstacles.

The *goal waypoint* is hidden behind the sensors line of sight. There is multiple cost equivalent trajectories to reach the goal.

| Position | | $\mathcal{WP}_1$ |
|---|---|---|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| $[25, 5, 0]^T$ | $[0°, 0°, 90°]^T$ | $[35, 75, 0]^T$ |

Table 7.11: Mission setup for *Slalom* scenario.

**Obstacle set:** Obstacles are discovered during a flight by *UAS LiDAR Sensor*. The set of obstacles is defined in (tab. 7.12) Some obstacles does not have *Line of Sight* during a flight, which causes additional constraints during *avoidance trajectory selection* process.

| Obstacle | | Body Margin | | | Safety Margin |
|---|---|---|---|---|---|
| position | type | min. | max. | avg. | |
| multiple (4) | hospital | $[0.5, 1]$ | $[2.2, 3.1]$ | $[1.5, 3]$ | $[1, 3]$ |
| multiple (7) | unusual | $[0.3, 1]$ | $[2.3, 3.5]$ | $[2, 3]$ | $[1, 4]$ |
| multiple (3) | square | $[3, 4]$ | $[4, 5]$ | $[4, 5]$ | $[1, 4]$ |

Table 7.12: *Obstacle set* for *Slalom* scenario.

**Main goal:** Show *static obstacle avoidance* in *clustered environment* with *shorter decision frames* due the obstacle density. Show *hidden waypoint navigation capability* and Behind Line of Sight impact on decision making.

**Acceptance Criteria** are given as follow:

1. *Hidden waypoint reach* - the UAS will safely reach *goal waypoint*.
2. *Minimal safety margin distance* $\geq 0$.
3. *Hindered space* is accounted into decision making (BLOS impact).

**Testing setup:** The *standard test setup* defined in (tab. 7.2. 7.3. 7.4. 7.5. 7.6) is used with following parameter override:

1. *Avoidance grid - type - ACAS-like* with enabled *Horizontal maneuvers*

*Note.* The *vertical separation* was disabled, because *UAS* will just increase its altitude to reach *goal waypoint*.

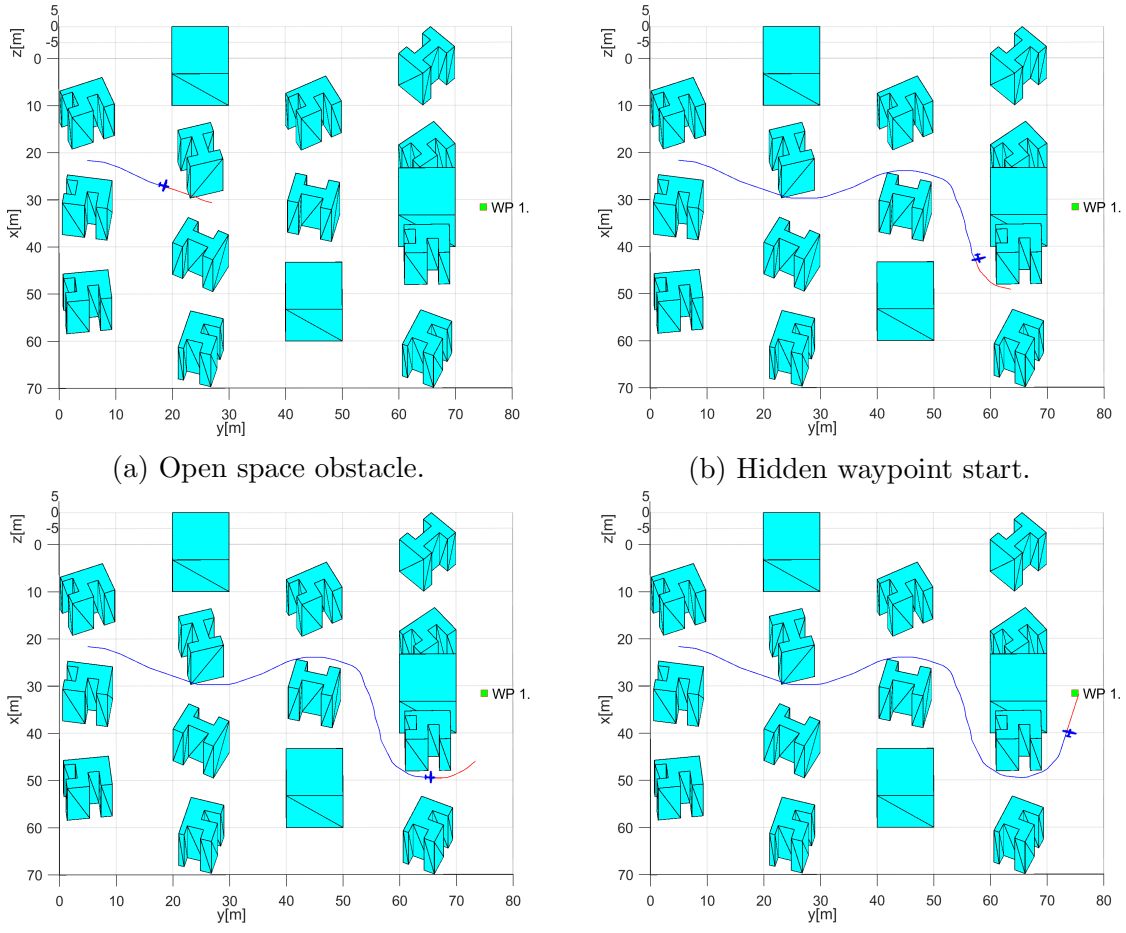
**Simulation run:** Notable moments from this *simulation run* (fig. 7.5) are following:

1. *Open space obstacle* (fig. 7.5a) - avoidance of open space obstacle, while tracking *hidden waypoint*. This is standard navigation procedure, the middle building in front of *goal waypoint* is hidden by building in front of UAS.

2. *Hidden waypoint navigation* is shown in three stages start (fig. 7.5b), middle (fig. 7.5c), and end phase (fig. 7.5d).The *hidden goal waypoint* has been reached and first acceptance criteria was fullfilled. The *Decision points* of navigational loop are placed in very high density around this area. The avoided building had following traps which were avoided:

   a. Trap (fig. 7.5b) on the left side of *UAS* was avoided, because there was no turning point inside of space.

   b. Trap (fig. 7.5c) on the left side of UAS was avoided, because it was not wide enough to be considered as trajectory space.




(a) Open space obstacle.

(b) Hidden waypoint start.

(c) Hidden waypoint middle.

(d) Hidden waypoint end.

Figure 7.5: Test scenario for *Slalom* with *hidden waypoint*.

**Distance to Body/Safety Margin Evolution:** The *UAS* (blue fill) does not break a *safety margin* (yellow fill) nor *body margin* (red fill) as you can seen in (fig. 7.6). Hindered space is accounted into decision making, because the distance to closest obstacle will never breach *safety margin* (yellow fill). If it was not, the UAS will break *safety* or *body* margin.

*Body* and *Safety margin* are changing values depending on the *nearest obstacle* and *mutual position of obstacle and UAS*. The ranges of *body* and *safety margins* are reflected in (tab. 7.12).

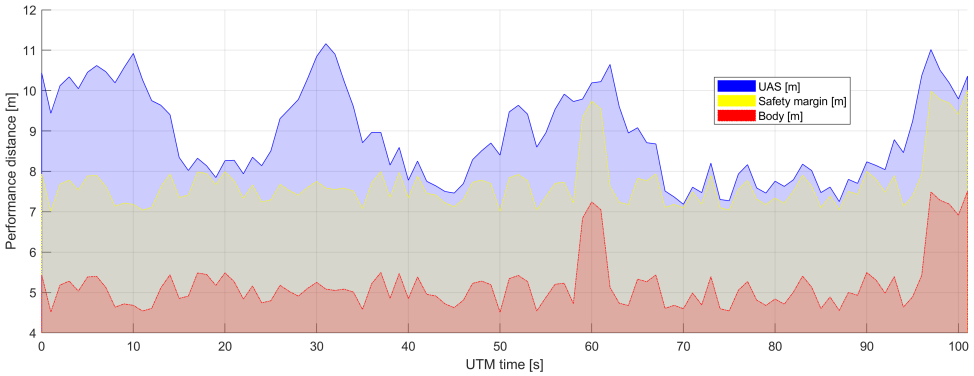



Figure 7.6: Distance to body/safety margin evolution for *Slalom scenario*.

**Distance to Body/Safety Margin Peaks:** The *UAS* distance to boundary of *safety* and *body* margin is given in (tab. 7.13). The minimal distance of *UAS border*(blue line) to *safety margin boundary* (yellow line fig. 7.6) is 0.0856 $m$ which can be considered as marginal 0. The minimal *body margin* distance is 2.5856 $m$ which is reflects *safety margin* 2.5 $m$ at that moment. The condition $safetyMarginDistance \geq 0$ holds.

The difference between minimal and maximal *safety margin distance* is $\sim 3$ $m$ which indicates that mission environment is tightly packed with obstacles.

| Parameter | | UAS 1 |
|---|---|---|
| Distance to Safety Margin | min | 0.0856 |
| | max | 3.7391 |
| Distance to Body Margin | min | 2.5856 |
| | max | 6.2391 |

Table 7.13: Distance to body/safety margin peaks for *Slalom scenario*.

**Path tracking performance:** Path tracking is given in (fig. 7.7). The line between Starting position (green square, marked S) and goal waypoint (green square marked 1) is reference trajectory (green dashed line). The flown trajectory (blue solid line) is showing evolution over mission time (Time [s]) in global coordinate frame split into three axes (x[m], y[m], z[m]). The UAS was all time in *Emergency Avoidance Mode* due the vicinity of dangerous obstacles.

The *UAS* reached final navigation waypoint, which fulfills acceptance criteria. The UAS has taken a significant detour (x[m] evolution) due to hidden *waypoint*.

The test has been run multiple times to check if *Right-Up* preference for avoidance is always selected. *Small noise* (0.5-1m) was added to obstacle positions. The algorithm always chose similar deterministic path. The higher noise levels were not possible due the obstacle original size (tab. 7.12).

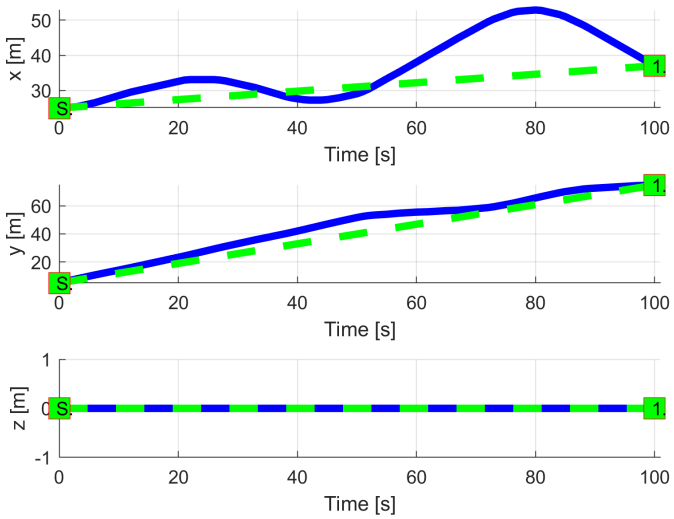



Figure 7.7: *Slalom* path tracking.

**Path Tracking Deviations:** Deviations given in (tab. 7.14) from *reference trajectory* (fig. 7.7) are in expected ranges considering *mission plan* (tab. 7.11) and *obstacle properties* (7.12).

| Param. | UAS 1 |
|---|---|
| | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 17.90 |
| $\max \lvert y \rvert$ | 12.41 |
| $\max \lvert z \rvert$ | 0 |
| $\max dist.$ | 20.06 |

Table 7.14: Path tracking properties for *Slalom* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.8) shows used time (y-axis) over decision frame (x-axis).

The UAS is moving over *semi-clustered* environment the *computation load* is almost constant.

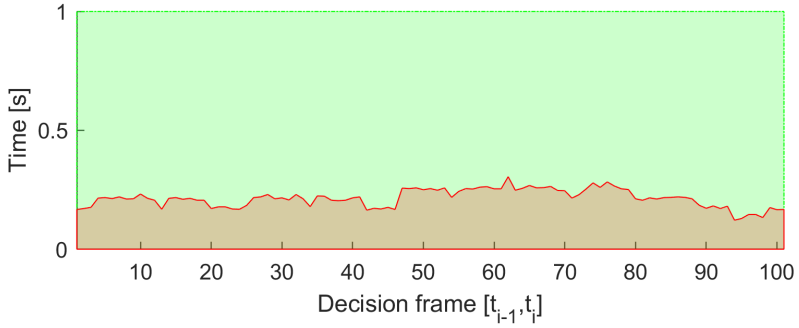



Figure 7.8: Computation time for *Slalom* scenario.

### 7.3.3 Maze

**Scenario:** The UAS is flying a mission given by (tab. 7.15) in *closed space* constrained by ground from bottom, airspace constraint from top and building from sides. The maneuverable space is *maze-like* with *hidden goal waypoint*.

There exists a *Obstacle map* with defined *safety* and *body margins*. *Reference trajectories* (direct interconnection of initial position and *goal waypoint*) is going trough *partially known space* with some charted obstacles.

| Position | | $\mathcal{WP}_1$ |
|---|---|---|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| $[15, 15, 0]^T$ | $[0°, 0°, 0°]^T$ | $[15, 75, 0]^T$ |

Table 7.15: Mission setup for *Maze* scenario.

**Obstacle set:** *Obstacles* are discovered during a flight by *UAS LiDAR* sensor. The *Obstacle set* is defined in (tab. 7.16). the obstacles are placed in *virtual grid* with *cell size* $10 \times 10m$. There are following obstacles:

1. $5\times$ *Hospital building* - H-shaped, with two open traps, with minimal body margin in range $0.5 - 1m$, with maximal body margin in range $2.2 - 3.1m$ and variable *safety margin* in range $1 - 3m$.

2. $12\times$ *Unusual trap building* - square shaped building with two traps on neighbouring side, with minimal body margin in range $0.3 - 1m$, with maximal body margin in range $2.3 - 3.5m$ and variable *safety margin* in range $1 - 4m$.

3. $6\times$ *Square building* - square shaped building with minimal body margin in range $3 - 4m$, with maximal body margin in range $4 - 5m$ and variable *safety margin* in range $1 - 4m$.

4. $7\times$ *U-shaped Trap* - thin walled U shaped trap designed to catch incoming flying objects, with minimal body margin in range $2 - 4m$, maximal body margin in range $3 - 5m$ and various *safety margin* in range $1 - 2m$.

The purpose of these *Obstacles* except *Square building* type is to create false positive path diversions. These diversions are designed to take *UAS* into unsolvable situation. *Avoidance* of traps is possible due *Reach set properties*, because many scenarios for avoidance can be evaluated at once.

| Obstacle | | Body Margin | | | Safety Margin |
|---|---|---|---|---|---|
| position | type | min. | max. | avg. | |
| multiple (5) | hospital | $[0.5, 1]$ | $[2.2, 3.1]$ | $[1.5, 3]$ | $[1, 3]$ |
| multiple (12) | unusual | $[0.3, 1]$ | $[2.3, 3.5]$ | $[2, 3]$ | $[1, 4]$ |
| multiple (6) | square | $[3, 4]$ | $[4, 5]$ | $[4, 5]$ | $[1, 4]$ |
| multiple (7) | trap | $[2, 4]$ | $[3, 5]$ | $[2, 4]$ | $[1, 2]$ |

Table 7.16: *Obstacle set* for *Maze* scenario.

**Main Goal:** Demonstrate static obstacle avoidance in closed space navigation. Focus on determinism of *avoidance run*. Demonstrate the possibilities of primitives *right-hand* maze solver incorporated into *Navigation-loop*.

**Acceptance Criteria:**

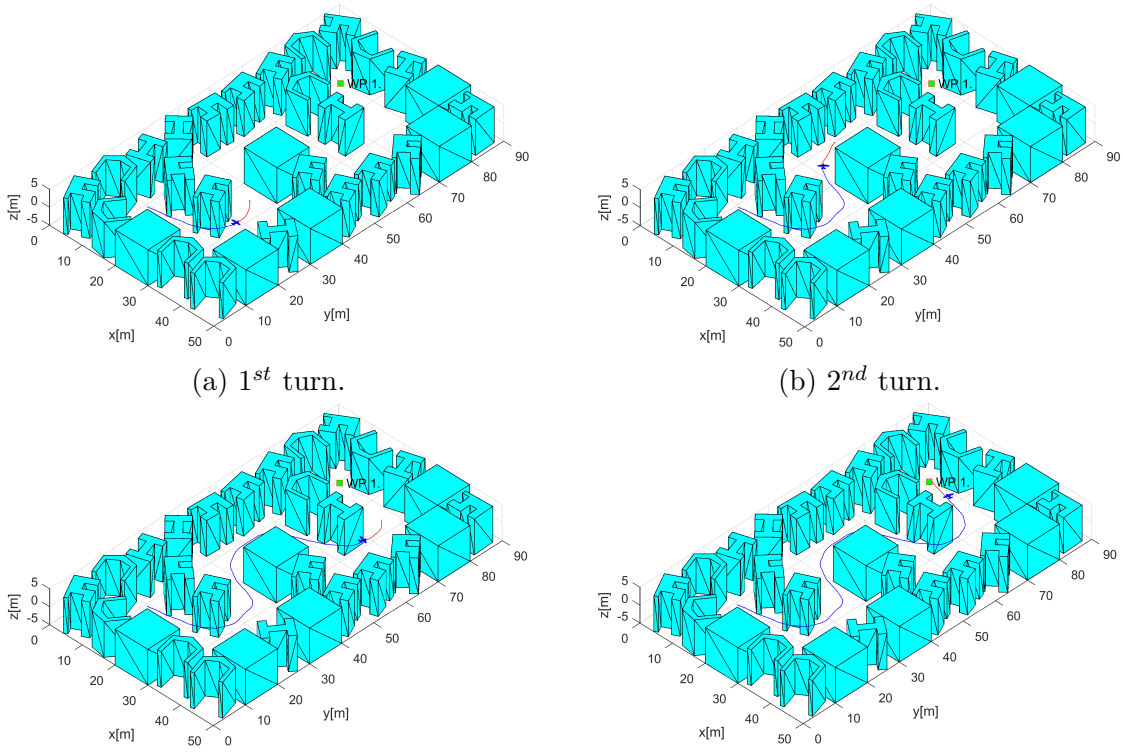
1. *Do not break top/bottom boundaries* - the *UAS* Z coordinate should not leave range $-5$ to $5m$. The boundary break occurs when there is no feasible horizontal path and UAS needs to climb up to resolve situation.
2. Minimal safety margin distance $\geq 0m$.
3. *Reach hidden goal waypoint* by solving simple maze (tab. 7.15).

**Testing Setup:** The *standard test setup* defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Avoidance grid - type - ACAS-like* with enabled *Horizontal maneuvers*

**Simulation Run:** Notable moments from the simulation run (fig. 7.9) are following:

1. *The Maze* consist from heavy constrained turns: $1^{st}$ turn (fig. 7.9a), $2^{nd}$ turn (fig. 7.9b), and $3^{rd}$ turn (fig. 7.9c). The hidden waypoint reach is given by (fig. 7.9d).
2. UAS is constantly in *Emergency Avoidance mode*, because there is always a presence of obstacle,
3. *Navigation path* is located in slim corridor with width only 3-6 meters. Mutual distance of obstacles is 20 meters and combined margins takes 14-17 meters.
4. *Maze scenario* was very close to urban environment in terms of obstacle density and computation complexity.
5. *Avoidance run* computation complexity scaled linearly with count of active obstacles in Field of View.
6. *Hidden Goal Waypoint* have been reached as shown in (fig. 7.9d). This satisfy *reach hidden waypoint* acceptance criterion.




(a) $1^{st}$ turn. (b) $2^{nd}$ turn.

(c) $3^{rd}$ turn. (d) Waypoint reach.

Figure 7.9: Test scenario for *Maze*.

**Distance to Body/Safety Margin Evolution:** The evolution of *body and safety margin* over time (x-axis, sec) given in meters distance (y-axis, m) is given in (fig. 7.10).

The *UAS* center distance to nearest obstacle (blue line) does not break any *Safety Margin* (yellow line) of closest obstacle. *Body Margin* of closest obstacle (red line) has not been break, because it always lies below of *Safety Margin* (yellow).

For *UTM time period* 37 to 68 sec there is a *margin spike* due avoidance of bloated *Rectangle buildings* (fig. 7.9b) during $2^{nd}$ turn. The *acceptance criterion* for *Safety Margin* is satisfied.

*Note.* The *body* and *safety margin* are changing depending on *UAS position* and *orientation*. The changes are reflected in (tab. 7.17).

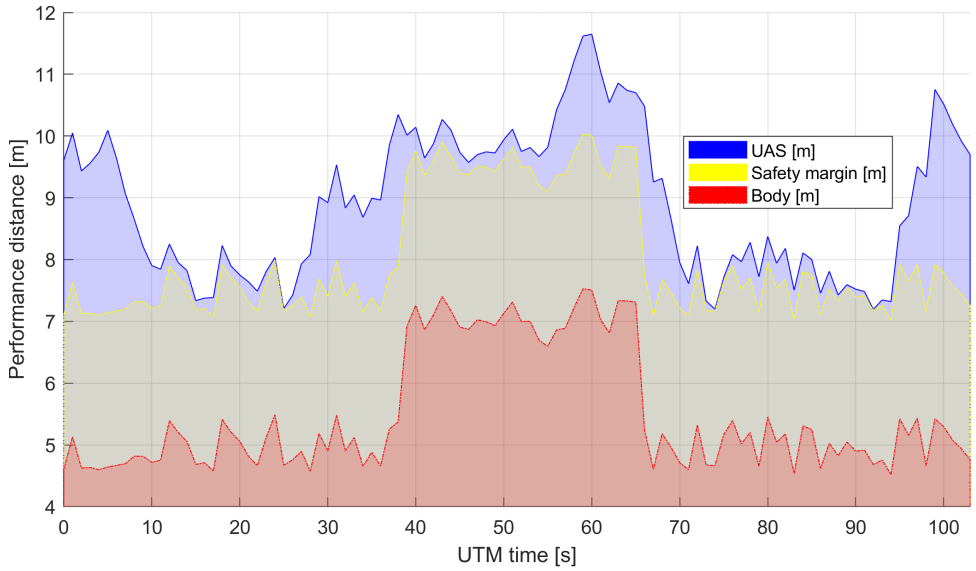



Figure 7.10: Distance to body/safety margin evolution for *Maze scenario*.

**Distance to Body/Safety Margin Peaks:** The minimal and maximal values for *UAS distance* to *safety margin* based on performance (fig. 7.10) is summarized in (tab. 7.17).

The *minimal distance to safety margin* is $0.0131m$ which can be taken as $\sim 0m$ due the numerical error. The *maximal distance to safety margin* is $2.9513m$ which is $5\times$ *UAS radius*. The safety margin distance is $\leq 3m$ which means the scenario is tightly packed with obstacles. The *UAS* never left *Emergency Avoidance Mode* because condition: $safetyMarginDistance \geq avoidanceGridLength$ was never satisfied.

The *minimal body* distance is $5.0131m$, while the *maximal body* distance is $8.7117m$. The difference between minimal and maximal body distance is $\sim 4m$ which also indicates scenario packed with obstacles.

| Parameter | | UAS 1 |
|---|---|---|
| Distance to Safety Margin | min | 0.0131 |
| | max | 2.9513 |
| Distance to Body Margin | min | 5.0131 |
| | max | 8.7117 |

Table 7.17: Distance to body/safety margin peaks for *Maze scenario*.

**Path Tracking Performance:** Reference path (green dashed) line is given as direct interconnection of *initial position* (green square with S marker) and *hidden waypoint* (green square with 1 marker). The *UTM Reference Time* is given on x-axis. The evolution of real trajectory (blue solid line) for each axis is given as follow:

1. *X-axis path tracking* - reflects the maneuvering in the curves of the maze.
2. *Y-axis path tracking* - shows horizontal progress to the *hidden goal waypoint*. The expected linear tracking is not achievement due the manuevering delays on X-axis.
3. *Z-axis path tracking* - shows perfect linear tracking of reference trajectory. The *altitude acceptance criterion*: $-5m \leq altitude \leq 5m$ have been fulfilled.

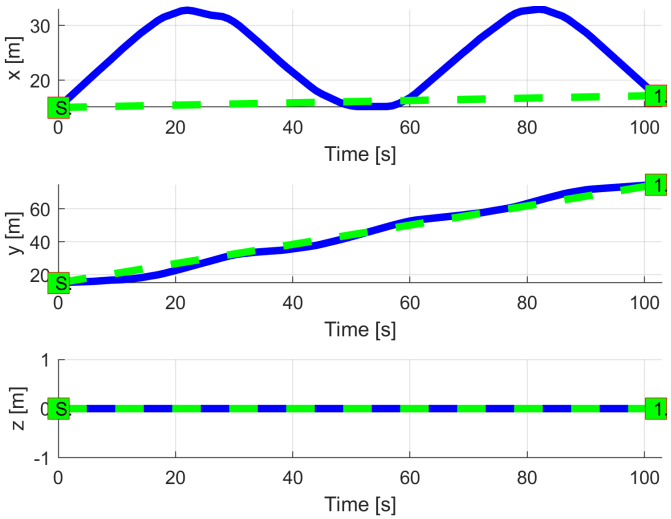



Figure 7.11: *Maze* path tracking.

**Path Tracking Deviations:** Deviations (tab. 7.18) from *reference trajectory* are in expected ranges considering *mission plan* (tab. 7.15) and *obstacle properties* (tab. 7.16).

| Param. | UAS 1 |
|---|---|
| | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 27.32 |
| $\max \lvert y \rvert$ | 2.41 |
| $\max \lvert z \rvert$ | 0 |
| $\max dist.$ | 28.06 |

Table 7.18: Path tracking properties for *Maze* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.12) shows used time (y-axis) over decision frame (x-axis).

The UAS is constantly in *Emergency Avoidance Mode*, the *operational environment* is *clustered* with obstacles. This causes very high *computation load*.

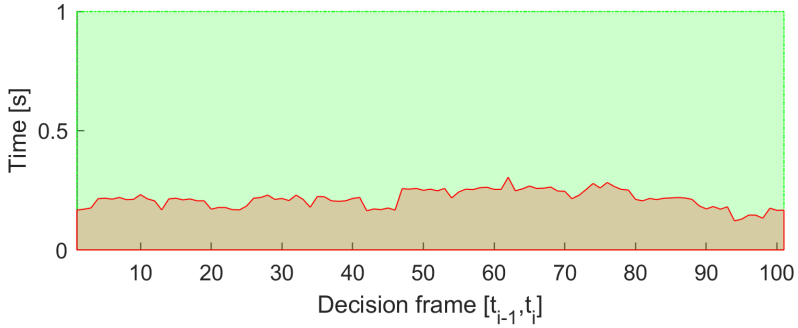



Figure 7.12: Computation time for *Maze* scenario.

### 7.3.4 Storm

**Scenario:** Small UAS is flying in open space in uncontrolled airspace ($\leq$ 500 feet AGL (Above Ground Level)). A *Weather Service* notices UAS about *Dangerous Weather zone* (virtual constraint s. 6.5.3) which is moving in UAS direction. The *UAS* is executing mission given by (tab. 7.19).

| Position | | $\mathcal{WP}_1$ |
|---|---|---|
| $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| $[0, 0, 0]^T$ | $[0°, 0°, 90°]^T$ | $[0, 60, 0]^T$ |

Table 7.19: Mission setup for *Storm* scenario.

**Constraints:** The *storm* is modeled as a *virtual constraint* with parameters given in (tab. 7.20). A constraint is modeled as a *convex polygon* for *horizontal boundary* and altitude for the *vertical boundary*.

The *Storm* is moving trough an *operaitonal region* with linear velocity $0.5ms^{-1}$. The *storms center* was first detected at *decision frame* 0 at position $[0, 50, 0]^T$.

| Constraint | | | Body Margin | | | Safety Margin |
|---|---|---|---|---|---|---|
| i. position | velocity | type | min. | max. | avg. | |
| $[0, 50, 0]^T$ | $[0, -0.5, 0]$ | polygon | 9 | 10 | 9.5 | 5 |

Table 7.20: *Constraint set* for *Storm* scenario.

**Assumption:** Every *avoidable moving constraint* is usually slower than an *Approaching UAS*, or its radius is smaller than turning radius of an *Approaching UAS*.

*Note. Manned aviation* receives permit to operate in a *controlled airspace* only if it has capability outmaneuver every known threat in requested airspace.

The *Constrained space portion* is usually very large, therefore in majority of cases the assumption $uasSpeed >> constraintSpeed$ holds.

**Main Goal:** Show dynamic moving constraint avoidance capability in an *uncontrolled airspace*.

**Acceptance criteria:**

1. *Hard constraint avoidance* - the *UAS* must not cross the body margin: $distance(stormCenter, UAS) \geq bodyMargin$.
2. *Soft constraint avoidance* - the *UAS* can not cross the safety margin to get into close proximity of *Storms surrounding area*: *distance(stormCenter, UAS)* $\geq$ *safetyMargin*.

**Testing setup:** The *standard test setup* defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Avoidance grid - type - ACAS-like* with enabled *Horizontal maneuvers*.

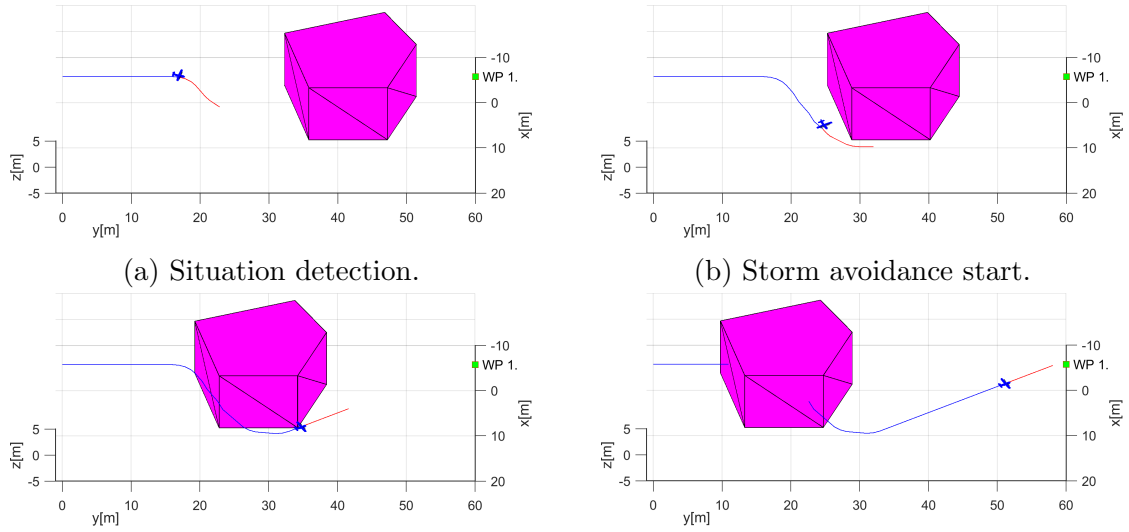
**Simulation run:** *Notable moments* from a *simulation run* (fig. 7.13) are following:

1. *Detection* (fig. 7.13a) - the *Storm* (magenta polygon) is detected prior the engagement (retrieved from associated weather service). The *UAS* (blue) stays in *Navigation mode*. *Trajectories* in *Navigation grid* are constrained by rule *Enforce safety margin* (tab. 6.16). The *Planned trajectory* (red) changes to avoid *Storm*.

2. *Avoidance start* (fig. 7.13b) - when *optimal avoidance distance* is reached by UAS, the *navigation reach set* is constrained, forcing UAS to perform evasive maneuver.

3. *Avoidance end* (fig. 7.13c) - navigation space is no longer constrained when the *minimal safe distance/heading* is achieved.

4. *Waypoint reached* (fig. 7.13d) - standard waypoint navigation procedure was used in this case.




(a) Situation detection.

(b) Storm avoidance start.

(c) Storm avoidance end.

(d) Waypoint reach.

Figure 7.13: Test scenario for *Storm* (Dynamic hard constraint).

**Distance to Body/Safety Margin Evolution:** The *body margin* (red line) and *safety margin* (yellow line) and *UAS distance to storm center* (blue line) evolution over *UTM time* (x-axis) are given in (fig. 7.14) The *body* and *safety* margin was changing according to mutual position of the *storm* and the *UAS* (see tab. 7.20).

The acceptance criteria for the *hard constraint avoidance* and *soft constraint avoidance* have been fulfilled.

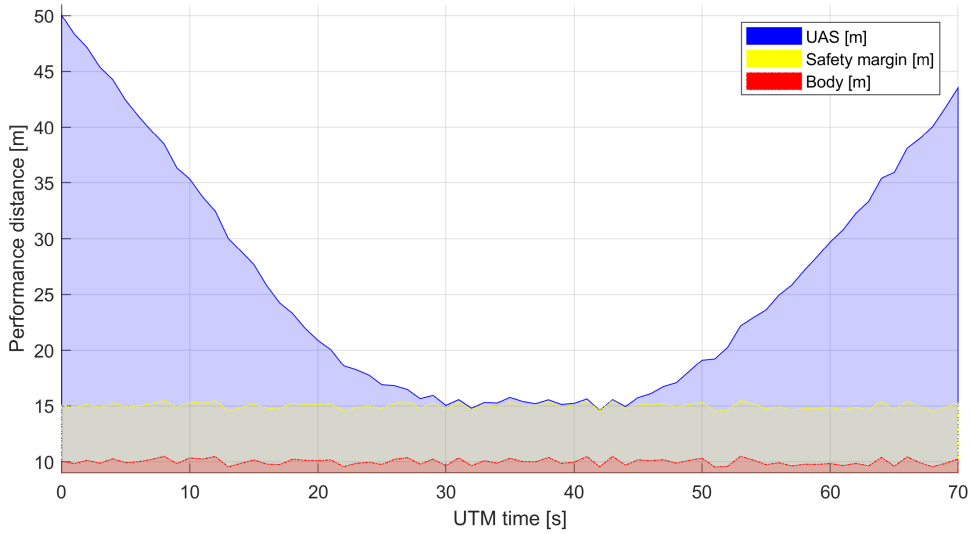



Figure 7.14: Distance to body/safety margin evolution for *Storm scenario*.

**Distance to Body/Safety Margin Peaks:** A *hard constraint* of *body margin* was not breached, because the *distance(UAS(t),stormBody(t))* was all time greater than *0*. Thus the *UAS* stayed well clear from *Storm*. The summary (tab. 7.21) shows that the *minimal body margin distance* was 5.0335 $m$, which proves *avoidance of hard constraint*.

A *soft constraint* represented as *safety margin* (protective coating around storm body) was not breached, because the *distance(UAS(t), stormBody(t)) - safetyMarginm(t)* was all time greater than *0*. The summary (tab. 7.21) show that the *minimal safety margin distance* was 0.0355 $m$, which proves *avoidance of soft constraints*.

| Parameter | | UAS 1 |
|---|---|---|
| Distance to Safety Margin | min | 0.0355 |
| | max | 34.9934 |
| Distance to Body Margin | min | 5.0355 |
| | max | 39.9934 |

Table 7.21: Distance to body/safety margin peaks for *Storm scenario*.

**Path Tracking Performance:** The *path tracking* (blue solid line) of *reference trajectory* (green dashed line) between *starting waypoint* (green square marked "S") and *final waypoint* (green square marked "1") is portrayed in (fig. 7.15). The *UAS* executes *horizontal right-side avoidance* of the *Storm* as is preferred.

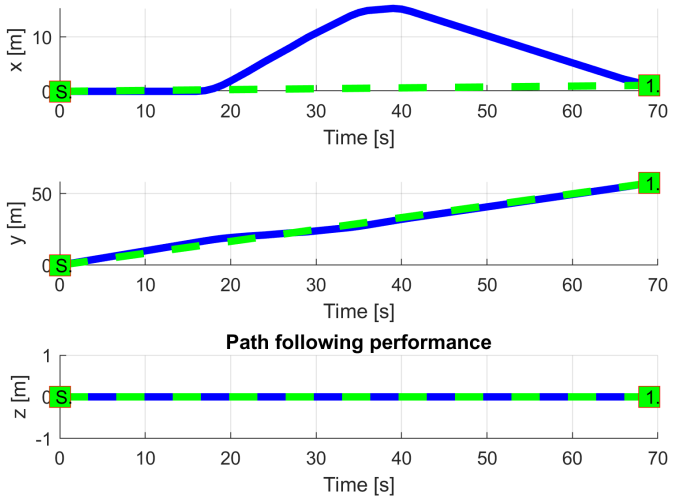



Figure 7.15: *Storm* path tracking.

**Path Tracking Deviations:** *Deviations* (tab. 7.22) are in expected ranges considering the mission plan (tab. 7.19) and *body* and *safety* margins (tab. 7.20).

| Param. | UAS 1 |
|---|---|
| | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 15.26 |
| $\max \lvert y \rvert$ | 1.32 |
| $\max \lvert z \rvert$ | 0 |
| $\max dist.$ | 15.76 |

Table 7.22: Path tracking properties for *Storm* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.16) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is low, it only increases slightly during avoidance maneuver.

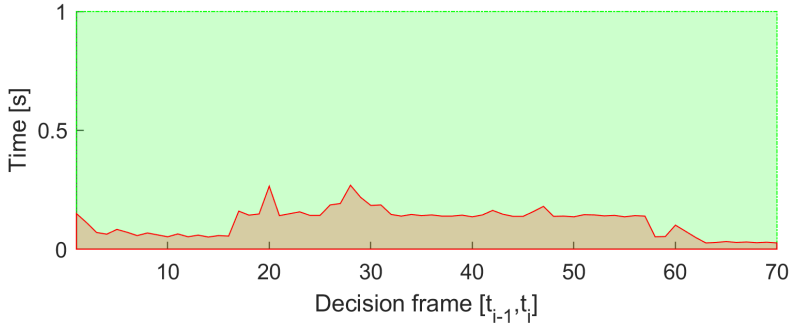



Figure 7.16: Computation time for *Maze* scenario.

### 7.3.5 Emergency Converging

**Scenario:** Two *UAS* are flying in an *uncontrolled airspace* (altitude $\leq$ 500 ft. Above the Ground Level) with missions defined in (tab. 7.23). Both *UAS* are in the *Navigation mode* with active *ADSB-In/Out*, receiving position notification from each other. Cruising altitude is sufficient for horizontal separation (50-100 ft. Above the Ground Level). *Horizontal separation* is preferred separation type for both *UAS*.

| UAS | Position | | $\mathcal{WP}_1$ |
|---|---|---|---|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[40, 20, 0]^T$ |
| 2 | $[20, 0, 0]^T$ | $[0°, 0°, 90°]^T$ | $[20, 40, 0]^T$ |

Table 7.23: Mission setup for *Emergency converging* scenario.

*Note. Collision point* is expected at $\mathcal{C} = [20, 20, 0]^T$. The *angle of approach* is 90° which classifies situation as *Converging maneuver* (fig. 6.31).

**Main Goal:** Show two *non-cooperative* UAS avoidance capability for *Converging maneuver* scenario in *uncontrolled airspace*.

**Acceptance criteria:**

1. *Proper mode invocation* - when an intruder intersects the UAS with *Right of the Way* navigation grid, both UAS will swith into *Emergency Avoidance Mode*.
2. *Minimal safety margin distance* $\geq 0m$.
3. *Each UAS* will reach own goal waypoint (tab. 7.23).

**Testing setup:** The *standard test setup* for each UAS defined in (tab 7.2, 7.3, 7.4, 7.5, 7.6) is used with following without parameter override.

*Intruder intersection* model has been chosen depending on UAS (tab. 7.24). Each UAS is equipped with *ADS-B In/Out* sensor obtaining/distributing following information:

1. *Position* - in operational section coordinate frame.
2. *Velocity* - vector representation in given coordinate frame.
3. *Class size* - class body radius based on UAS propulsion and size.
4. *Safety margin set* - set of safety margins for different collision cases.

*Avoidance parameters* for *Emergency converging scenario* are given in (tab. 7.24). Each UAS has same speed set to $1ms^{-1}$. Second UAS has the *Right of The Way*.

*Safety margin* is considered as sum of both participants *near miss margins*. In this case default safety margin is considered as 1.2 $m$.

| UAS | Parameters | | | Margins | | Separation |
|---|---|---|---|---|---|---|
| | velocity | intruder model | ROW | body | safety | |
| 1 | 1 | body + spread | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body + spread | true | 0.3 | 0.6 | horizontal |

Table 7.24: Avoidance parameters for *Emergency converging* scenario.

*Note.* Both UAS are using body (sec. 6.6.3) and spread (sec 6.6.4) intersection models, reflecting both body volume and maneuverability of intruder. Both UAS have preferred separation mode as *horizontal*, typical for planes.

**Simulation Run:** Notable moments from the simulation run (fig. 7.17) are following

1. *Detection* (fig. 7.17a) - Intruder (UAS2 cyan) is approaching (UAS 1 blue) from right side, Intruder (UAS2 cyan) has the right of the way, because $70° \leq angleOfApproach < 130°$. *Intruder intersection model* (for UAS 2) is created and propagated in *avoidance grid* (for UAS 1).

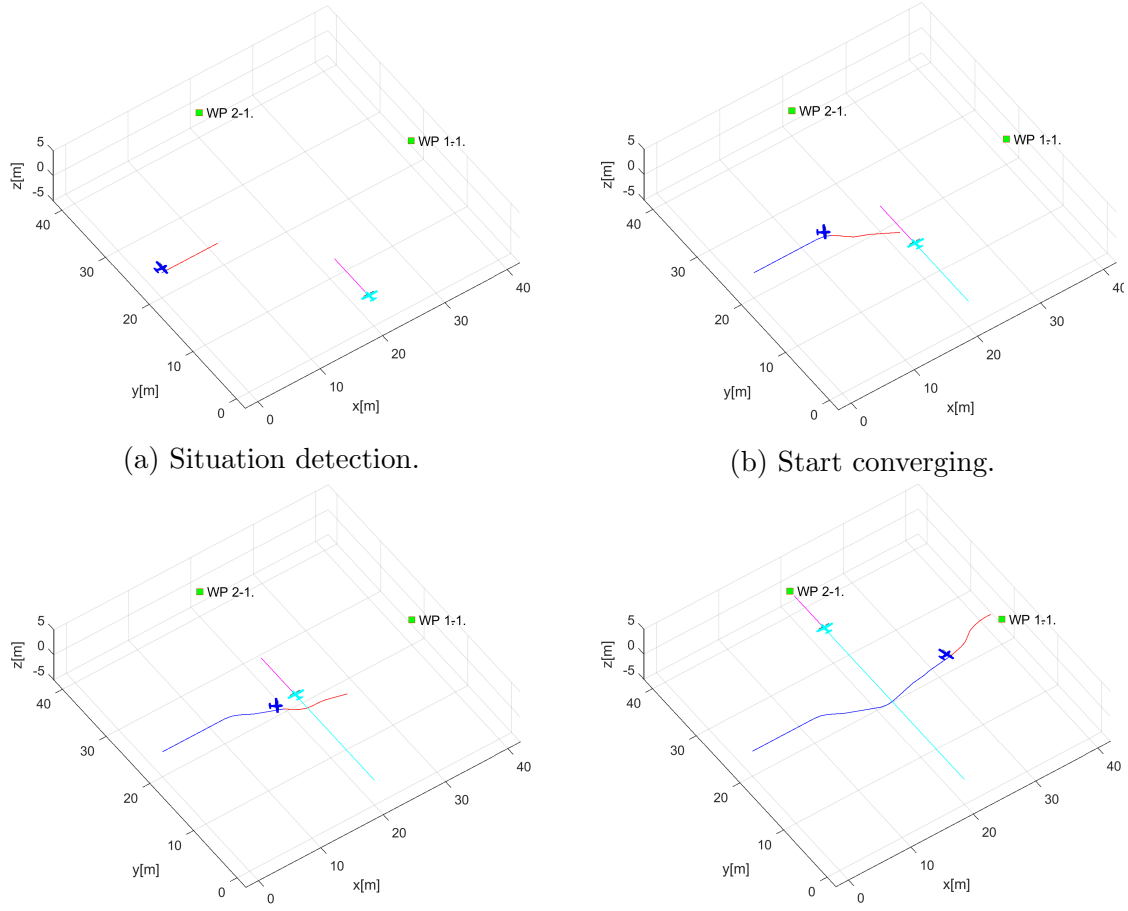
2. *Start Converging* (fig. 7.17b) - when *UAS 2 (cyan) parametric intruder intersection model* disables *trajectories*, converging maneuver for UAS 1 (blue) starts.

3. *Near miss case* (fig. 7.17c) - UAS 1 (blue) to UAS 2 (cyan) closest distance. The safety margin for *near miss* has not been breached. The safety margin for *well clear* in uncontrolled airspace is invalid.

4. *Waypoint reached* (fig. 7.17d) - the intruder intersection model for *UAS 2* (cyan) is removed from UAS 1 (blue) *avoidance grid* after *converging maneuver competition*, standard navigation procedure is applied afterwards.

5. Note that *UAS 2* (cyan) has *Right of the way* in (tab. 7.24).

6. Note that *UAS 1* (blue) used only horizontal separation (priority) in (fig. 7.19a).




(a) Situation detection.

(b) Start converging.

(c) Near miss case.

(d) Waypoints reach.

Figure 7.17: Test scenario for *Emergency converging* (Intruder avoidance).

**Distance to Safety Margin Evolution:** There is need to compare mutual distance between both UAS (y-axis [m]) and its evolution over UTM time (x-axis [s]). The *mutual* distance of *UAS 1* to *UAS 2* is given by *blue line*. The *Safety margin* value is denoted by red line at *constant value* of 1.2 $m$.

The *Proper avoidance Invocation* is shown when UAS systems are getting closer to each other and they enter (Emergency Avoidance Mode) to provide *active separation*. The *Mutual distance evolution* (blue line) does not cross *safety margin* (red line).

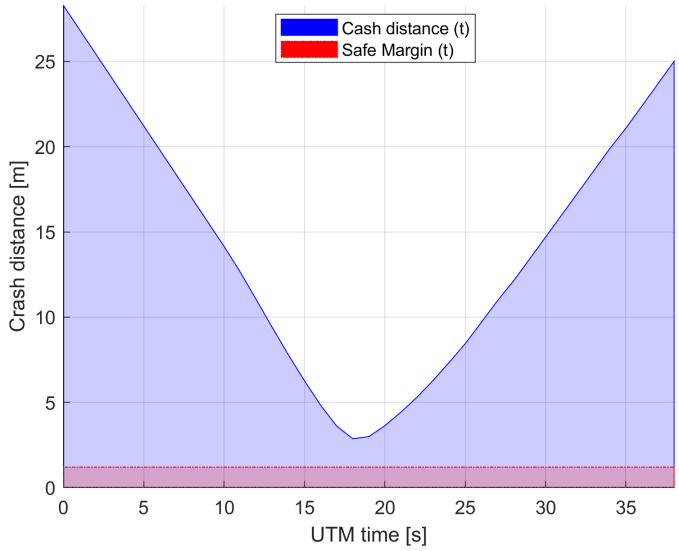



Figure 7.18: Distance to safety margin evolution for *emergency converging scenario*.

**Distance to Safety Margin Peaks:** Minimal and Maximal mutual distance to safety margin is summarized in (tab. 7.25). The closest to collision are UAS systems when the distance to safety margin is $1.6676m$.

The *minimal distance to safety margin* $\geq$ *0* which means that the *safety acceptance criterion* is fullfilled.

| UAS: | | 1-2 |
|---|---|---|
| Distance to Safety Margin | min | 1.6676 |
| | max | 27.0843 |

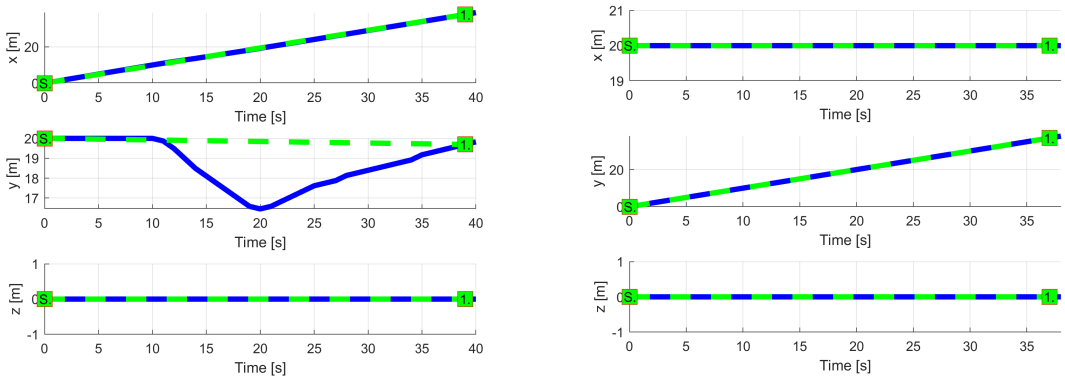
Table 7.25: Distance to safety margin peaks for *emergency converging scenario*.

**Path Tracking Performance:** All waypoints (green numbered squares) for both UAS have been reached (fig. 7.19). *Reference trajectories* (green dashed lines), between initial position (green square marked S) and goal waypoint (green square marked 1) are split into three XYZ values with respective figures. The tracked value is on y-axis [m] and time on x-axis [s]. The blue lines represents real parameter evolution over time.

Following observations can be made from path tracking (fig. 7.19) and preferred separations (tab. 7.24):

1. UAS 1 (fig. 7.19a) is using *horizontal separation* (y-axis). The UAS diverges from reference trajectory to minimal necessary time.

2. UAS 2 (fig. 7.19b) has the right of the way and is not using any active avoidance mechanism.




(a) UAS 1.

(b) UAS 2.

Figure 7.19: *Trajectory tracking* for *Emergency converging* test case.

**Path Following Deviations:** *Deviations* (tab. 7.26) are in expected ranges considering the *mission plans* (tab. 7.23) and *separation safety margin* (tab. 7.24).

| Param. | UAS 1 | UAS 2 |
|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 0 | 0 |
| $\max \lvert y \rvert$ | 3.25 | 0 |
| $\max \lvert z \rvert$ | 0 | 0 |
| $\max dist.$ | 3.25 | 0 |

Table 7.26: Path tracking properties for *Emergency converging* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.20) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is increased only for UAS 1 during avoidance period. The UAS 2 remains unaffected because it has a right of the way.

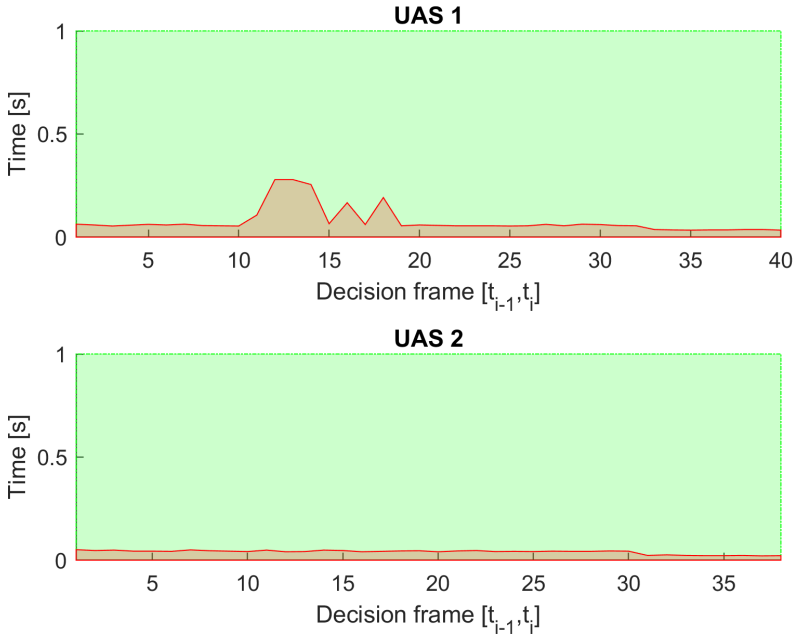



Figure 7.20: Computation time for *Emergency converging* scenario.

### 7.3.6 Emergency Head on

**Scenario:** Two *UAS* systems are flying in an *uncontrolled airspace* (altitude $\leq$ 500 ft. Above the Ground Level) with missions defined in (tab. 7.27). Both *UAS* are in the *Navigation mode* with active *ADSB-In/Out*, receiving position notifications from each other. Cruising altitude is sufficient for horizontal separation (50-100 ft. Above Ground Level). *Horizontal separation* is preferred mode for both *UAS*.

| UAS | Position | | $\mathcal{WP}_1$ |
|---|---|---|---|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[40, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0°, 0°, 180°]^T$ | $[0, 20, 0]^T$ |

Table 7.27: Mission setup for *Emergency head on* scenario.

*Note. Collision point* is expected at $\mathcal{C} = [20, 20, 0]^T$. The *angle of approach* is 180° which classifies situation as *Head on maneuver* (fig. 6.29).

**Main Goal:** Show two *non-cooperative* UAS avoidance for *Head-on approach scenario* in *uncontrolled* airspace.

**Acceptance criteria:**

1. *Proper mode invocation* - when an intruder intersects the opposing *UAS* Navigation grid, bot intruder and *UAS* will switch to *Emergency Avoidance Mode*. None of the *UAS* have *Right Of the Way*.
2. *Minimal Safety Margin distance* $\geq 0m$. That means the mutual distance of both *UAS centers* does not go below given *safety margin*.
3. *Both UAS* will reach own goal waypoint (tab. 7.27).

**Testing setup:** The *standard test setup* for each UAS defined in (tab 7.2, 7.3, 7.4, 7.5, 7.6) is used with following without parameter override.

*Intruder intersection* model has been chosen depending on UAS (tab. 7.28). Each UAS is equipped with *ADS-B In/Out* sensor obtaining/distributing following information:

1. *Position* - in operational section coordinate frame.
2. *Velocity* - vector representation in given coordinate frame.
3. *Class size* - class body radius based on UAS propulsion and size.
4. *Safety margin set* - set of safety margins for different collision cases.

*Avoidance parameters* for *Emergency head-on scenario* are given in (tab. 7.28). Each UAS has same speed set to $1ms^{-1}$. None of them have the *Right of The Way*.

*Safety margin* is considered as sum of both participants *near miss margins*. In this case default safety margin is considered as 1.2 $m$.

| UAS | Parameters | | | Margins | | Separation |
|---|---|---|---|---|---|---|
| | velocity | intruder model | ROW | body | safety | |
| 1 | 1 | body (timed) | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body (timed) | false | 0.3 | 0.6 | horizontal |

Table 7.28: Avoidance parameters for *Emergency head on* scenario.

*Note.* Both UAS are using body (sec. 6.6.3) intersection model, reflecting both body volume along expected trajectory. Both UAS have preference for *horizontal* separation mode, typical for planes.

**Simulation Run:** Notable moments from the simulation run (fig. 7.21) are following:

1. *Situation detection* (fig. 7.17a) - UAS 1 (blue) is approaching UAS 2 (cyan) with $130° \leq angleOfApproach \leq 180°$, this is considered head on approach. Head on approach give the *right of the way* neither to *UAS 1* nor *UAS 2*. *Intruder intersection model* for opposite UAS is created in respective *avoidance grids*. *Head on emergency avoidance* starts independently in each UAS without intruders coordination. First *avoidance maneuver* is invoked when the *intruder intersection model* constraints any trajectory in the *avoidance grid*. When this happens *Navigation mode* switch to the *Emergency avoidance mode*.

2. *Before near miss* (fig. 7.21b) - both *UAS* are in *emergency avoidance mode*, sticking to right side avoidance maneuver.

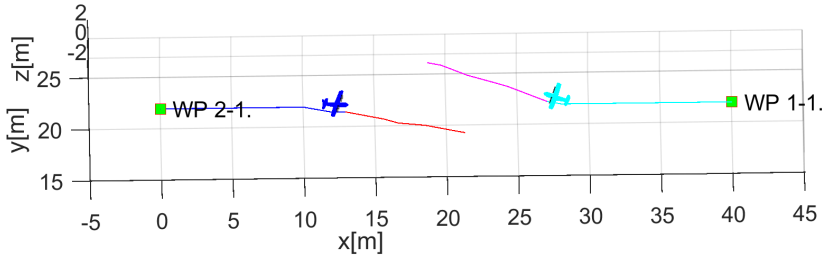
3. *Near miss case* (fig. 7.21c) - UAS 1 to UAS 2 closest distance. The safety margin for *near miss* has not been breached. The safety margin for *well clear* in uncontrolled airspace is invalid. Both UAS are using also *Horizontal separation* to avoid each other, *Emergency avoidance mode* is switched to the *Navigation mode* when risk of an *aerial clash* is voided.

4. *After near miss* (fig. 7.21d) - both *UAS* are tracking back to respective waypoint, correcting *altitude* (Z-axis in (fig. 7.23)) first.

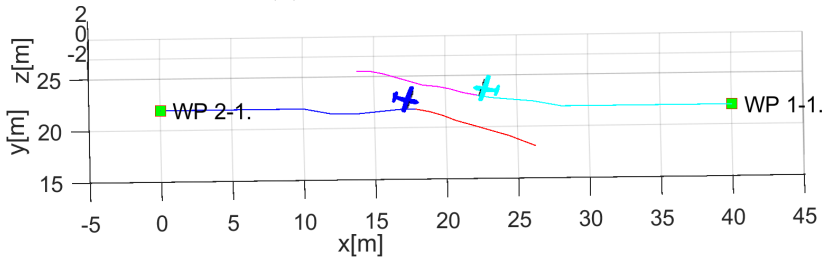
5. Note *Collision point* was expected at $\mathcal{C} = [20, 20, 0]^T$

6. Note *Both UAS* used *horizontal* (primary), *vertical* (secondary) separation (fig 7.23).

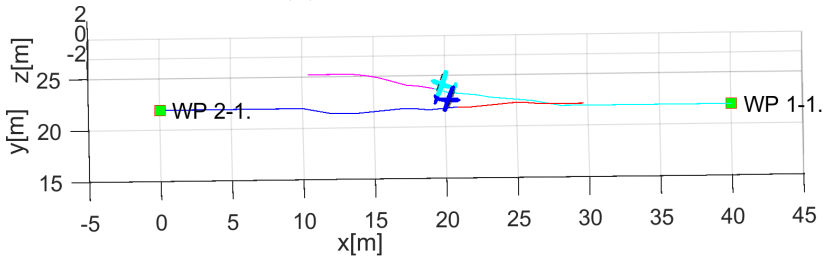
7. Note *Both UAS* decision times were *synchronized*, this is not an assumption, but it shows critical performance. Usually safety margin is bloated for (eq.6.156).



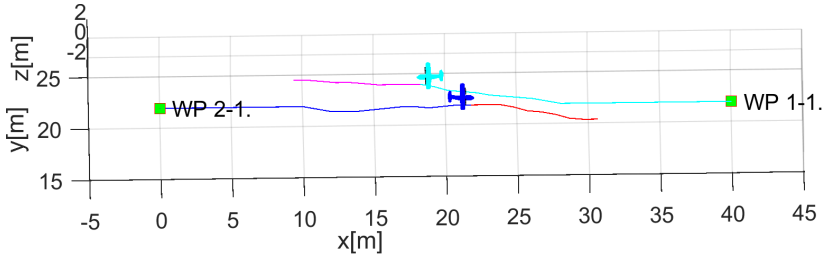

(a) Situation detection.




(b) Before near miss.




(c) Near miss.




(d) After near miss.

Figure 7.21: Test scenario for *Emergency head on approach* (Intruder avoidance).

**Distance to Safety Margin Evolution:** There is need to compare mutual distance between both UAS (y-axis [m]) and its evolution over synchronized *UTM time* (x-axis [s].) The *mutual distance* between bodies of *UAS 1*, *UAS 2* (blue line) compared to *Safety Margin* (red line) is given in (fig. 7.22). The *Safety Margin* value was constant for all time at value 1.2 *m* which is double of *Near Miss Margin for UAS 1 UAS 2*.

The proper *Avoidance Invocation* is shown when *UAS* systems are getting closer to each other and they starts their *separation phase* (Emergency Avoidance Mode switch). The mutual distance (blue line) does not cross *safety margin* (red line).

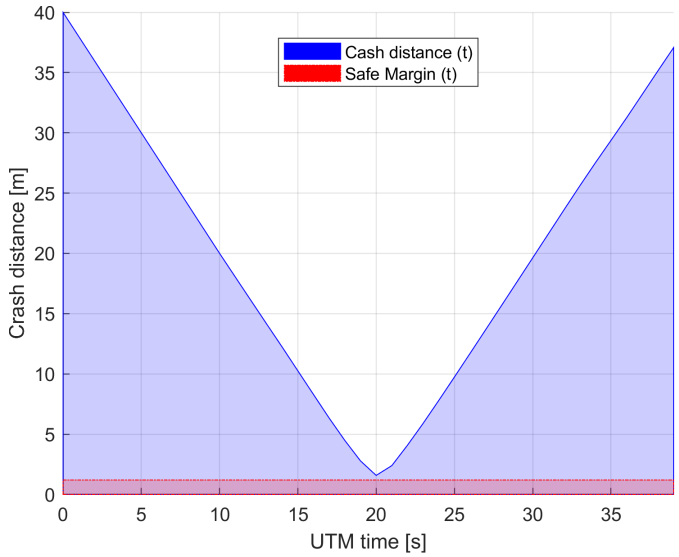



Figure 7.22: Distance to safety margin evolution for *emergency head on scenario*.

**Distance to Safety Margin Peaks:** Minimal and Maximal mutual distance to safety margin is summarized in (tab. 7.29). The closest to collision are UAS systems when *distance to safety margin* is $0.3824m$.

The *minimal distance to safety margin* $\geq 0$ which means that the *safety acceptance criterion* is full filled.

| UAS: | | 1-2 |
|---|---|---|
| Distance to Safety Margin | min | 0.3824 |
| | max | 38.8000 |

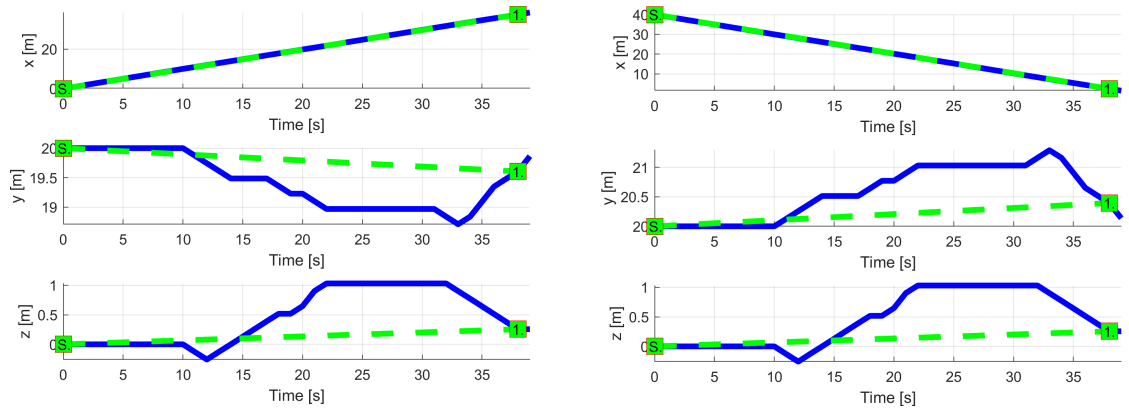
Table 7.29: Distance to safety margin peaks for *Emergency head on scenario*.

**Path Tracking Performance** All waypoints (green numbered squares) for both UAS have been reached (fig. 7.23). *Reference trajectories* (green dashed lines), between initial position (green square marked S) and goal waypoint (green square marked 1) are split into three XYZ values with respective figures. The tracked value is on y-axis [m] and time on x-axis [s]. The blue lines represents real parameter evolution over time.

Following observations can be made from path tracking (fig.7.23) and preferred separations (tab. 7.28):

1. UAS 1 (fig. 7.23a) is using horizontal separation going to the right (y-axis) and a little bit up (z-axis).

2. UAS 2 (fig. 7.23b) is using horizontal separation going to the right (left in GCS, y-axis) and little bit up (z-axis).




(a) UAS 1. (b) UAS 2.

Figure 7.23: *Trajectory tracking* for *Emergency head on* test case.

**Path Following Deviations:** *Deviations* (tab. 7.30) are in expected ranges considering the *mission plans* (tab. 7.27) and *separation safety margins* (tab. 7.28).

| Param. | UAS 1 | UAS 2 |
|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 0.05 | 0.06 |
| $\max \lvert y \rvert$ | 1.37 | 1.48 |
| $\max \lvert z \rvert$ | 1.03 | 1.05 |
| $\max dist.$ | 1.39 | 1.52 |

Table 7.30: Path tracking properties for *Emergency head on* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.20) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is increased only during *avoidance phase*. The *load* is symetric for both UAS systems.

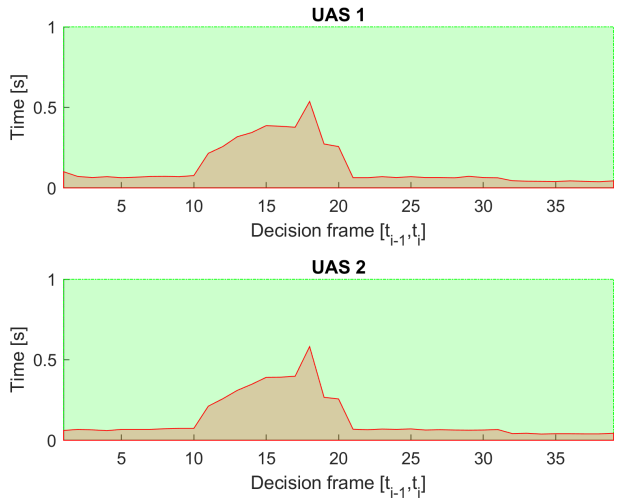



Figure 7.24: Computation time for *Emergency head on* scenario.

### 7.3.7 Emergency Mixed Head on with Converging

**Scenario:** *Four UAS* are flying in an *uncontrolled airspace* (altitude $\leq$ 500 ft. Above the Ground Level) missions defined in (tab. 7.31). All UAS are in the *Navigation mode* with active *ADS-B In*, receiving *position notifications* from each other. Cruising altitude is sufficient for *horizontal separation* (50-100 ft. Above the Ground Level).

| UAS | Position | | $\mathcal{WP}_1$ |
|---|---|---|---|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[45, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0°, 0°, 180°]^T$ | $[-5, 20, 0]^T$ |
| 3 | $[20, 0, 0]^T$ | $[0°, 0°, 90°]^T$ | $[20, 45, 0]^T$ |
| 4 | $[20, 40, 0]^T$ | $[0°, 0°, -90°]^T$ | $[45, 20, 0]^T$ |

Table 7.31: Mission setup for *Emergency mixed* scenario.

*Note. Collision point* is expected at $\mathcal{C} = [20, 20, 0]^T$

**Main Goal:** Show *multiple non-cooperative intruders avoidance capability* in *uncontrolled* airspace.

**Acceptance criteria:**

1. *Proper avoidance mode invocation* - when an *intruder intersection model* impact the *Avoidance Grid*, UAS system will switch to an *Emergency avoidance mode*.
2. *Minimal safety margin distance* $\geq 0m$.
3. Each *UAS* will reach own goal waypoint (tab. 7.31).

**Testing setup:** The *standard test setup* for each UAS defined in (tab 7.2, 7.3, 7.4, 7.5, 7.6) is used with following without parameter override.

*Intruder intersection* model has been chosen depending on UAS (tab. 7.32). Each UAS is equipped with *ADS-B In/Out* sensor obtaining/distributing following information:

1. *Position* - in operational section coordinate frame.
2. *Velocity* - vector representation in given coordinate frame.
3. *Class size* - class body radius based on UAS propulsion and size.
4. *Safety margin set* - set of safety margins for different collision cases.

*Avoidance parameters* for *Emergency mixed scenario* are given in (tab. 7.32). Each UAS has different *intruder model* and separation combination. Each UAS has same speed set to $1ms^{-1}$. None of UAS has the *Right of The Way*.

*Safety margin* is considered as sum of both participants *near miss margins*. In this case default safety margin is considered as 1.2 $m$.

| UAS | Parameters | | | Margins | | Separation |
|---|---|---|---|---|---|---|
| | velocity | intruder model | ROW | body | safety | |
| 1 | 1 | body + spread | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body (timed) | false | 0.3 | 0.6 | vertical |
| 3 | 1 | body (timed) | false | 0.3 | 0.6 | horizontal |
| 2 | 1 | body + spread | false | 0.3 | 0.6 | vertical |

Table 7.32: Avoidance parameters for *Emergency mixed* scenario.

*Note.* Each *UAS* use different intruder intersection models and primary *separations* (defined in tab. 7.32). UAS reactions are based on primary *Separation* mode, intruders intersection models this is reflected on major axial deviations in (fig. 7.27) and summarized in *path tracking* deviation (tab. 7.34).

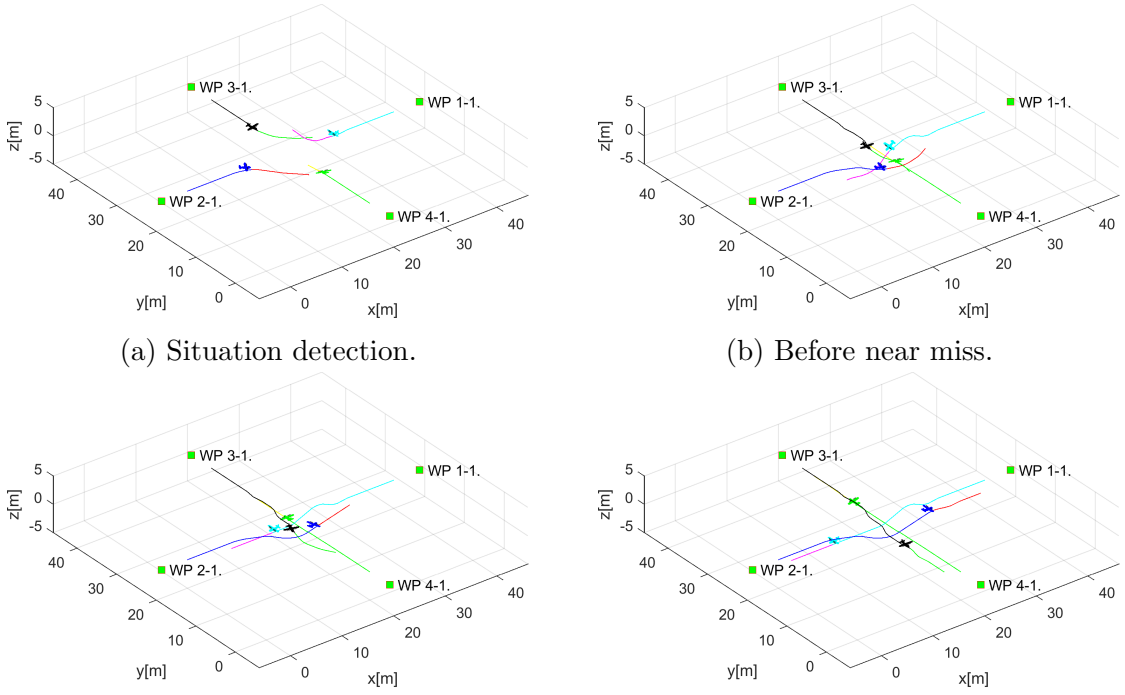
**Simulation Run:** Notable moments from the simulation run (fig. 7.25) are following:

1. *Situation detection* (fig. 7.25a) - UAS 1 (blue) is detecting UAS 2 (cyan), UAS 3 (green), and UAS 4 (black) as possible intruders. There are multiple converging and head on approaches depending on mutual positions (UAS and *angle of approach*). There exist at least one *converging case* where each UAS has *Right of the way*. Each UAS creates intruder intersection models depending on the intruder configuration (tab. 7.32). Each UAS enters into the *Emergency avoidance mode* independently, when at least one trajectory is constrained in the *avoidance grid*.

2. *Before near miss* (fig. 7.25b) - all *UAS* are in *emergency avoidance mode*, using various *separation modes* and *intruder intersection models*. Each UAS is performing its own avoidance maneuver, constantly checking other intruders. If same separation and same intruder model was used, there will be virtual roundabout.

3. *After near miss* (fig. 7.25c) - all *UAS* avoided each other which is covered in *safety margin performance* (fig. 7.26) and (tab. 7.33).

4. *Situation resolution* (fig. 7.25d) - all *UAS* returns to *Navigation mode* correcting *altitude* first and continuing to assigned waypoints.




(a) Situation detection.

(b) Before near miss.

(c) After near miss.

(d) Situation resolution.

Figure 7.25: Test scenario for *Emergency mixed* situation with *self-separation mode*.

**Distance to Safety Margin Evolution:** There is need to compare mutual distance between each UAS. The graph (fig. 7.26) shows six figures for each *UAS systems* mutual distance (blue line) in this scenario. The *Safety Margin* (red line) (1.2 $m$) was not breached for any pair (case).

The *Proper avoidance invocation* is shown when UAS systems are getting closer to each other and then they start separation phase (Emergency avoidance mode).

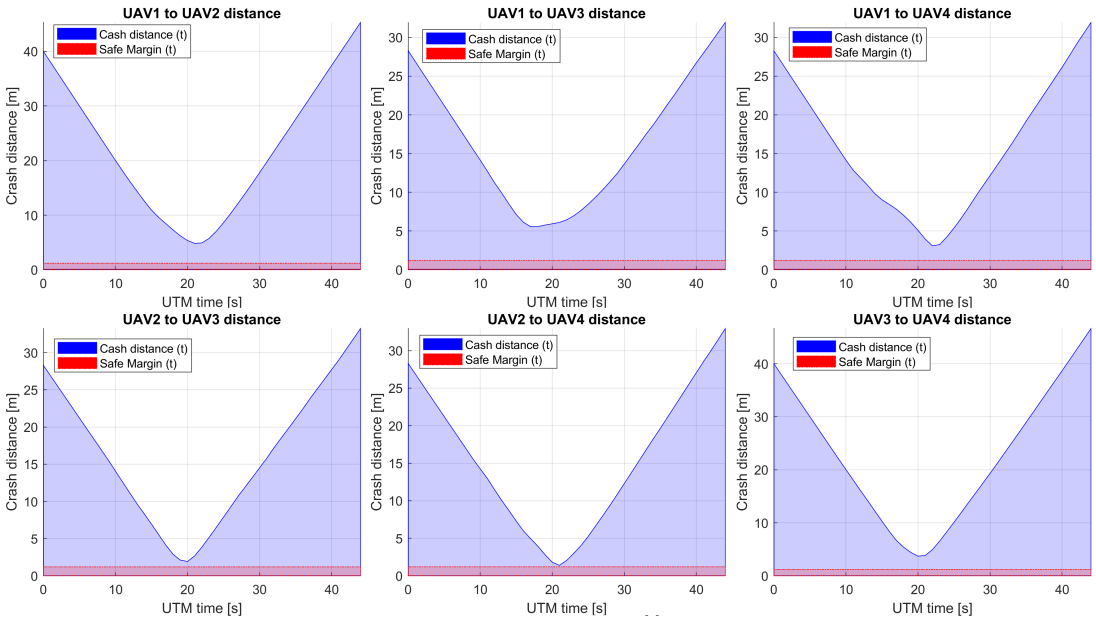



Figure 7.26: Distance to safety margin evolution for *emergency mixed scenario*.

**Distance to Safety Margin Peaks:** Minimal and Maximal mutual distance to safety margin is summarized in (tab. 7.33). There is no detected breach for any combination.

The *closest to collision* is UAS pair $2-4$ with mutual safety margin only 0.2019 $m$. On the other side is UAS pair $1-3$ with mutual safety margin 4.3721 $m$.

The *minimal distance to safety margin* $\geq 0$ which means that the *safety condition* is fulfilled.

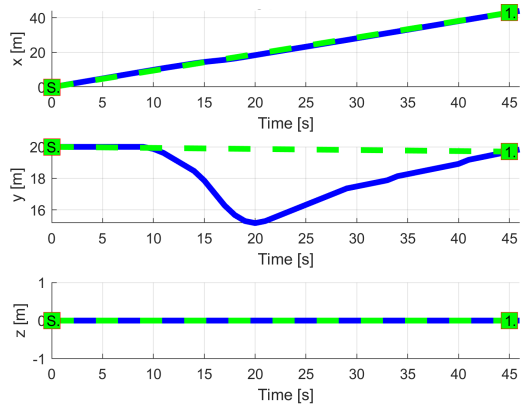
| UAS: | Distance to Safety Margin | | |
|---|---|---|---|
| | min | max | breach |
| 1-2 | 3.6231 | 44.0831 | false |
| 1-3 | 4.3721 | 30.7300 | false |
| 1-4 | 1.8959 | 30.7331 | false |
| 2-3 | 0.7331 | 32.0266 | false |
| 2-4 | 0.2019 | 31.7282 | false |
| 3-4 | 2.5171 | 45.4257 | false |

Table 7.33: Distance to safety margin peaks for *emergency mixed scenario*.

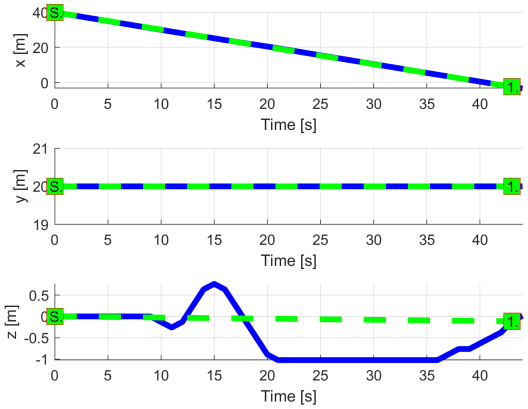
**Path Tracking Performance:** All waypoints (Green numbered squares) for all UAS have been reached (fig. 7.27). *Reference trajectories* (green dashed line) have been tracked by *UAS real path* (blue solid line) almost all time.

Following observations can be made from *path tracking* (fig. 7.27) and *preferred separations* (tab. 7.32):

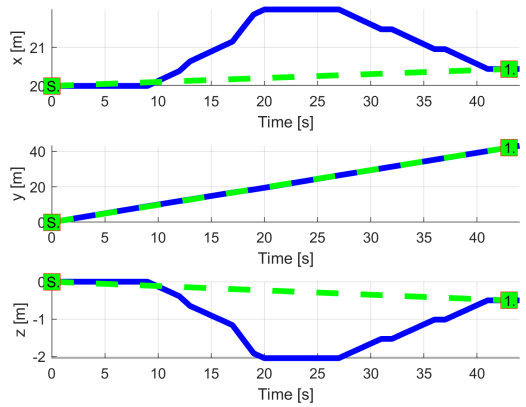
1. UAS 1 (fig. 7.27a) is using *horizontal separation* (y axis right) having *preferred horizontal separation*.
2. UAS 2 (fig. 7.27b) is using vertical separation (z axis up-down), having preferred vertical separation.
3. UAS 3 (fig. 7.27d) is using horizontal/vertical separation (x right, z down), having preferred horizontal separation. This UAS has used other than preferred separation type.
4. UAS 4 (fig. 7.27c) is using horizontal separation (x-axis right/left), having preferred vertical separation. This UAS has used opposite separation type, than preferred.



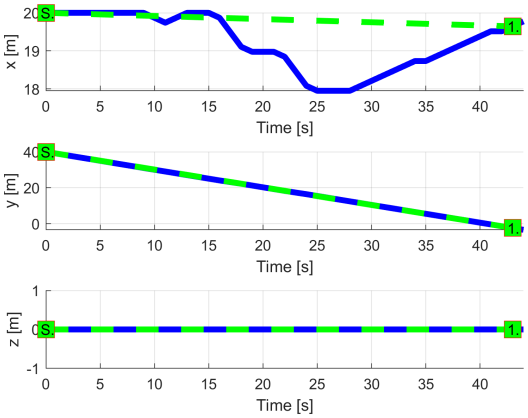

(a) UAS 1.




(b) UAS 2.




(c) UAS 3.




(d) UAS 4.

Figure 7.27: Trajectory tracking for *Emergency mixed* situation test case.

**Path Tracking Deviations:** *Deviations* (tab. 7.34) are in expected ranges considering the *mission plans* (tab. 7.31) and *separation safety margins* (tab. 7.32).

| Param. | UAS 1 | UAS 2 | UAS 3 | UAS 4 |
|---|---|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 0 | 0 | 1.98 | 2.05 |
| $\max \lvert y \rvert$ | 4.84 | 0 | 0 | 0 |
| $\max \lvert z \rvert$ | 0 | 1.23 | 2.43 | 0 |
| $\max dist.$ | 4.84 | 1.23 | 3.45 | 2.05 |

Table 7.34: Path tracking properties for *Emergency mixed* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.28) shows used time (y-axis) over decision frame (x-axis).

The *computation time* increases during periods of *active avoidance*. The *shortest* period of avoidance has UAS 1 and the longest period of avoidance has UAS 4.

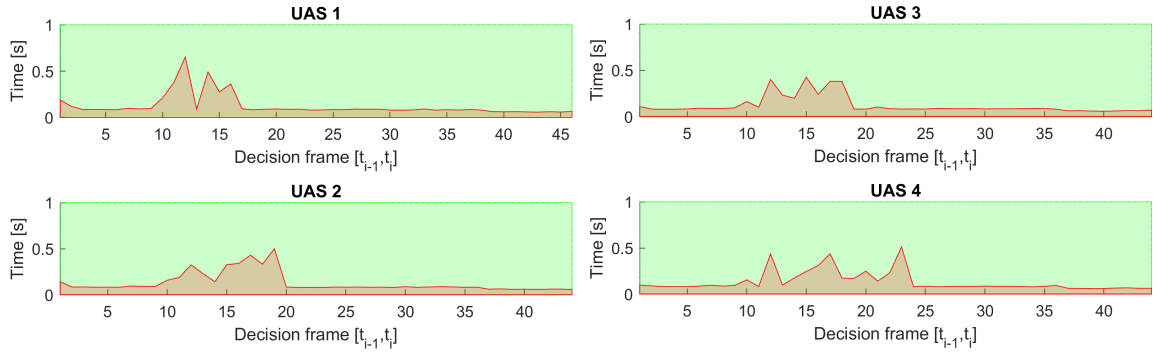



Figure 7.28: Computation time for *Emergency multiple* scenario.

## 7.4 Cooperative test cases

The *main goal* of this section is to show the operational capabilities of *approach* under *UTM supervision*. The minimal UTM functionality set (sec. 6.8) has been implemented, including *position notifications mechanism, collision case calculation, resolution enforcement* components.

Test cases covers *well clear breach prevention*, *situation based avoidance*, and *rules of the air enforcement*.

Coverage of *near miss situations*, *clash incidents* is given implicitly by *safety* and *body* margins (tab. 4.1).

1. *Rule based converging* (sec. 7.4.1) covers *well clear breach* and *converging rule of the air*, showing determinism and *UTM resolution execution*.

2. *Rule based head on* (sec. 7.4.2) covers *well clear breach* and *head on rule of the air*, showing determinism and *UTM resolution execution*.

3. *Rule based mixed head on with converging* (sec. 7.4.3) covers *well clear breach* and *head on and converging rules of the air*. The main focus is on *virtual roundabout* concept, when multiple collision cases are clustered into one avoidance maneuver.

4. *Rule based overtake* (sec. 7.4.4) covers *well clear breach* during *overtake* by faster UAS.

### 7.4.1 Rule based Converging

**Scenario:** Two *UAS* are approaching an *airway intersection* at *same time* in *controlled airspace* (over 500 feet Above the Ground Level). The mutual position of *UAS* can be classified as *Side approach*. Following *collision hazards* are present:

1. *Active Converging Collision Hazard* - There is an *UAS* approaching from the *right side*, which give him *Right of the Way* and invokes need to actively avoid *Intruder*.
2. *Passive Converging Collision Hazard* - There is an *UAS* approaching from the *left side*, which gave us *Right of the Way* and imposes an obligation of *active avoidance* on other *UAS*.

*Collision Hazards* must be addressed by *UTM* service in the following manner:

1. *Each UAS* in particular *Controlled Space* periodically sends synchronized *Position Notification* messages (tab. 6.6).
2. *UTM* service receives *Position Notifications* and manages *Collision Case* (tab. 6.7) in *Controlled Space*.
3. *UTM* detects *Converging Collision Case* with *Collision Point* in vicinity.
4. *UTM* service Sends *Mandate* to UAS without *Right of the Way* and implements *Normative Directive* on all *UAS* in area.

*Mission parameters* for both UAS systems are defined in (tab. 7.35).

| UAS | Position | | $\mathcal{WP}_1$ |
|---|---|---|---|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[40, 20, 0]^T$ |
| 2 | $[20, 0, 0]^T$ | $[0°, 0°, 90°]^T$ | $[20, 40, 0]^T$ |

Table 7.35: Mission setup for *Rule based converging* scenario.

**Assumptions:** Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover airworthy *UAS* has capability to precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for (UAS $\leftrightarrow$ UAS) and (UAS $\leftrightarrow$ UTM) communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.
3. *Decision frame synchronization with UTM* - necessary in discrete C2 environment otherwise *safety margins* needs to be *bloated*.
4. *Both UAS have identical cruising speed* - simplification impacting *UTM* service implementation. *Obstacle Avoidance Framework* can comprehend various intruders speed, with proper *UAS* directives.

**Main Goal:** Show possibility of *Converging situation resolution* with *forced safety margin* by *UAS Traffic Management* system. The *Obstacle Avoidance Framework based on Reach Sets* is used as *Navigation Module*.

**Acceptance Criteria:** Following criteria must be met:

1. *Well Clear Condition valid for both UAS* - Both *UAS* must have *minimal required distance* from *other UAS* for all *Converging Maneuver* enforcement time.
2. *Fulfillment of UTM Directives* - Both UAS must stay in *Navigation mode* for all *Converging Maneuver* enforcement time. *UAS without Right Of the Way* must stay away for necessary time, before returning to *Original Navigation waypoint* $\mathcal{WP}_1$ following.

**Testing Setup:** The *standard test setup* for each UAS defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Navigation grid - type - ACAS-like* with enabled *Horizontal maneuvers*

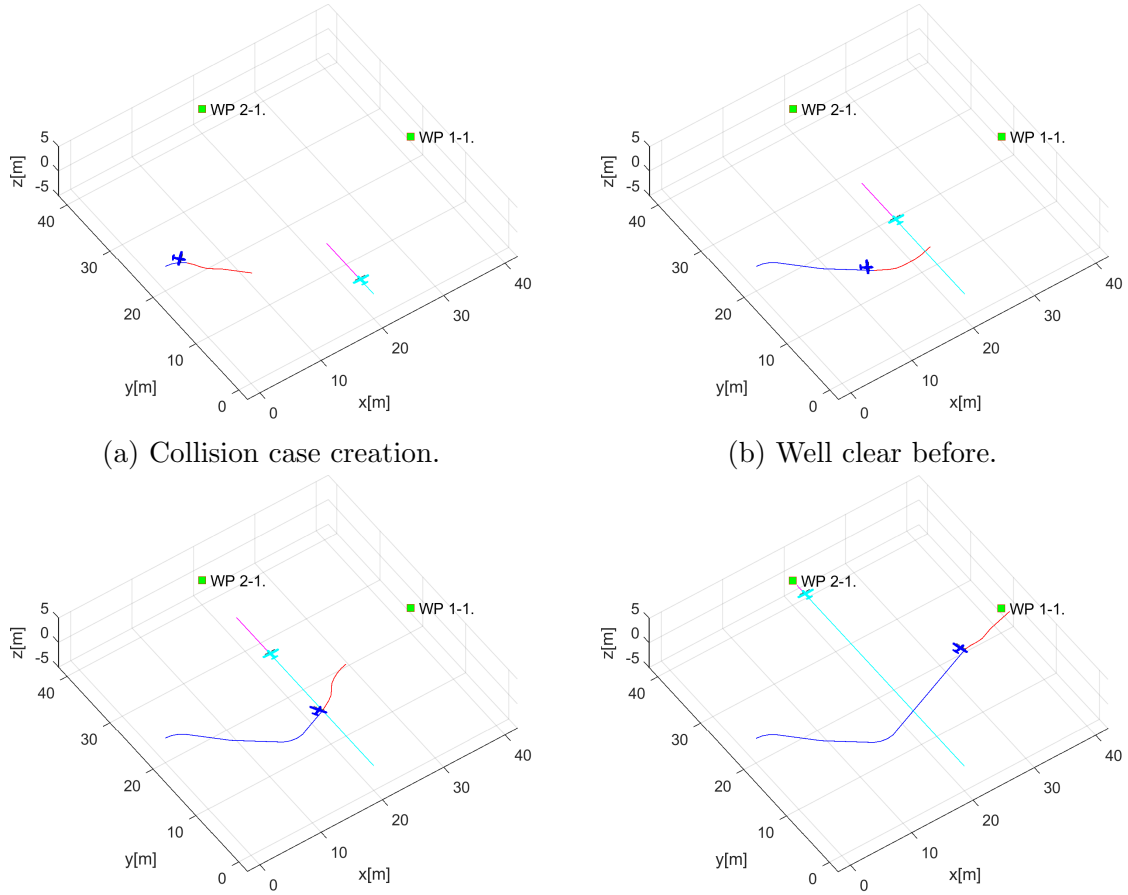
This *configuration* is based on assumption that every UAS is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for *climb or descent maneuver*. *Rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.33).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.7) based on incoming *UAS position notifications* (tab. 6.6).

**Simulation Run:** Notable moments from *simulation run* (fig. 7.29) are following:

1. *Collision Case creation* (fig. 7.29a) following events happens in this step:
   a. Two *UAS* are approaching *airway intersection*: UAS 1 (blue) from left and UAS 2 (cyan) from bottom.
   b. They are going to *collide* at point $\mathcal{C} = [20, 20, 0]^T$ of *Flight Level* (elevation is 45, 000 feet Above Mean Seal Level).
   c. UTM service notices future *Collision Situation* and creates *Collision Case*.
   d. *Converging Directive* for 8 m from *Collision point* is issued for UAS 1 (blue), because UAS 2 (cyan) has *Right Of the Way*.
   e. *Keep Velocity/Heading Directive* is issued for UAS 2 (cyan) to ensure avoidance maneuver success.
   f. UAS 1 (blue) corrects its heading according to *UTM* directive.
   g. UAS 2 (cyan) stays on claimed course and if its necessary adjust its speed.
2. *Well clear before* (fig. 7.29b) UAS 1 (blue) checks the *Collision Point* distance and keeps safe distance given by safety margin. UAS 2 (cyan) checks if there is no intruder in *Avoidance Grid* and if not, stays in *Navigation Mode*.
3. *Well clear after* (fig. 7.29c) UAS 2 (cyan) is *after Collision Point*, it can start negotiations of new speed and heading with UTM. UAS 1 (blue) is still enforced to follow *Converging Maneuver* directive, until the outer boundary of *Collision Zone* is reached.

4. *Waypoints reach* (fig. 7.29d) UAS 1 (blue) leaves outer boundary of *Collision zone*. Leaving *Converging Maneuver Directive*. UTM closes *Collision Case*.




(a) Collision case creation.

(b) Well clear before.

(c) Well clear after.

(d) Waypoints reach.

Figure 7.29: Test scenario for *Rule based converging*.

**Collision Case Calculation:** For test scenario in (fig. 7.29) where UAS 1 (blue) is converging to avoid UAS 2 (cyan) the *Collision Case* (tab. 7.36) have been calculated.

The *Collision point* is at $[20, 20, 0]$ in *Flight Level* $FL450$ coordinate frame.

The *angle of approach* was evaluated as 90° which indicates *converging maneuver* in range $70° \leq angleOfApproach < 130°$.

The *mutual position* of UAS 1 (blue) and UAS 2(cyan) is giving the roles: *Right Of the Way* for UAS 2 (cyan) and *Converging* for UAS 1 (blue).

The *safety margin* for *Well Clear* was determined as $3m$ for UAS 1 and $5m$ for UAS 2. (Note: Well Clear Margin is usually much greater than Near Miss margin). The *Combined Case* margin which was enforced was $8m$. The mutual distance can not go below this threshold.

| Collision Case | | | | | | Margins | |
|---|---|---|---|---|---|---|---|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Converging | $[20, 20, 0]^T$ | 90° | Converging | 3 | 8 |
| | 2 | Right o. W. | | | | 5 | |

Table 7.36: Collision case for *Rule-based converging* scenario.

**Distance to Safety Margin Evolution:** The safety margin values (well clear) (fig. 7.30) in controlled airspace are much greater than in non-controlled airspace (near miss) (fig. 7.18)

The enforced rule was (rule 6.13) with parameters: Collision Point $[20, 20, 0]^T$ and *Safety Margin* 8 $m$ as given by Collision Case (tab. 7.36).

The mutual *UAS distance* (blue line) does not go over *Safety Margin* (red line), which means UAS 1 well clear margin of 3 $m$ and UAS 2 well clear margin of 5 $m$ are not broken (fig. 7.30).

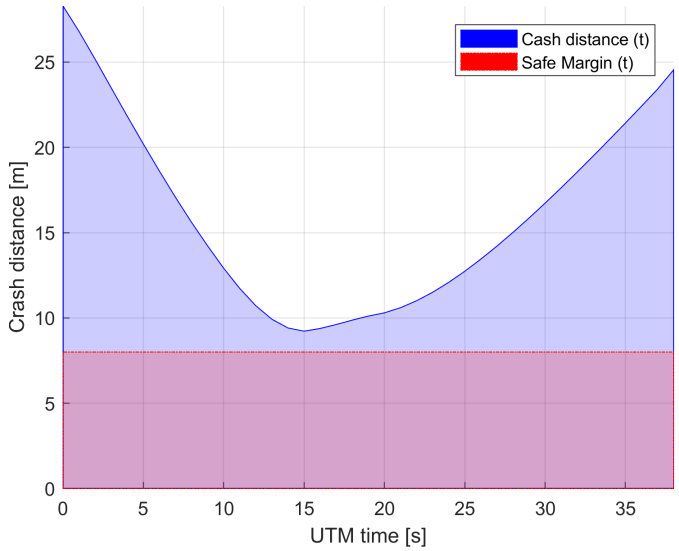



Figure 7.30: Distance to safety margin evolution for *rule based converging scenario.*

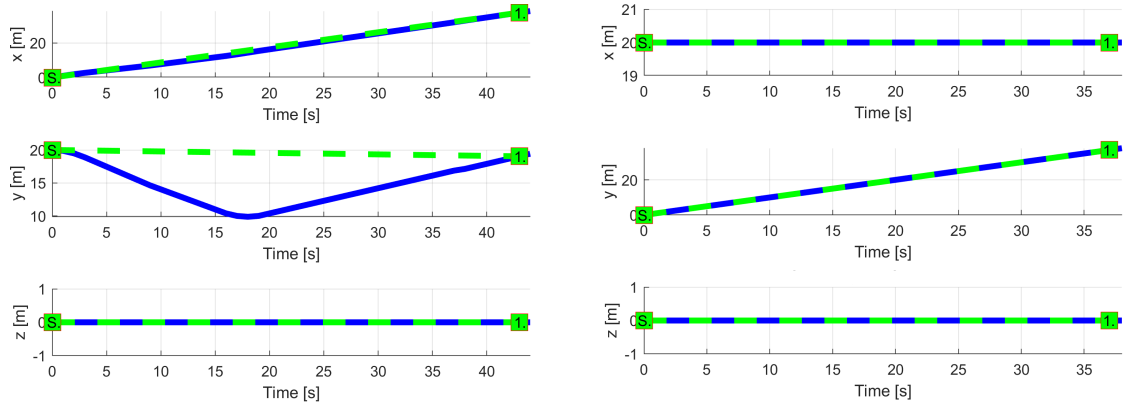
**Distance to Safety Margin Peaks:** *Distance to safety margin peaks* (tab. 7.37) represent the proximity on UAS mutual distance to *breach of well clear condition* (safety margin). The *breach of well clear condition* was not achieved. The *minimal distance to safety margin* was 1.2240 $m$. The *maximal distance to safety margin* was 20.2843 $m$ which represents distance in time of *Collision Case Creation.*

| UAS: | Distance to Safety Margin | | |
|---|---|---|---|
| | min | max | breach |
| 1-2 | 1.2240 | 20.2843 | false |

Table 7.37: Distance to safety margin peaks for *Rule based converging scenario.*

**Path Tracking Performance:** *Path tracking* is displayed in (fig. 7.31). The *UAS* trajectory is divined into *X, Y, Z axis tracking over UTM Time*. The *Reference Trajectory* (green dashed line) interconnect starting position of UAS (green square marked S) an goal waypoint (green square marked 1). The *Executed Trajectory* (blue solid line) reflects real UAS trajectory.

1. UAS 1. (fig, 7.31a) do steady right side *converging maneuver* (y-axis).
2. UAS 2. (fig. 7.31b) follows the reference trajectory precisely, because it has *Right Of the Way*.




(a) UAS 1. (b) UAS 2.

Figure 7.31: *Trajectory tracking* for *Rule based converging* test case.

**Path Tracking Deviations:** Deviations (tab. 7.38) are in *expected ranges*, considering the *mission plans* (tab. 7.35) and *Collision Case* safety margin of $8m$.

The minimal deviation distance was expected at value of *safety margin* ($8m$). The maximal deviation was $10.22m$ which is acceptable due the space discretization, UAS dynamic, and, *dynamic decision time*.

| Param. | UAS 1 | UAS 2 |
|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 0 | 0 |
| $\max \lvert y \rvert$ | 10.22 | 0 |
| $\max \lvert z \rvert$ | 0 | 0 |
| $\max dist.$ | 10.22 | 0 |

Table 7.38: Path tracking properties for *Rule based converging* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.32) shows used time (y-axis) over decision frame (x-axis).

The *computation time* is slightly increased for avoiding UAS 1 during avoidance. The initial increase of computation time UAS 2 is caused by UTM communication demand.

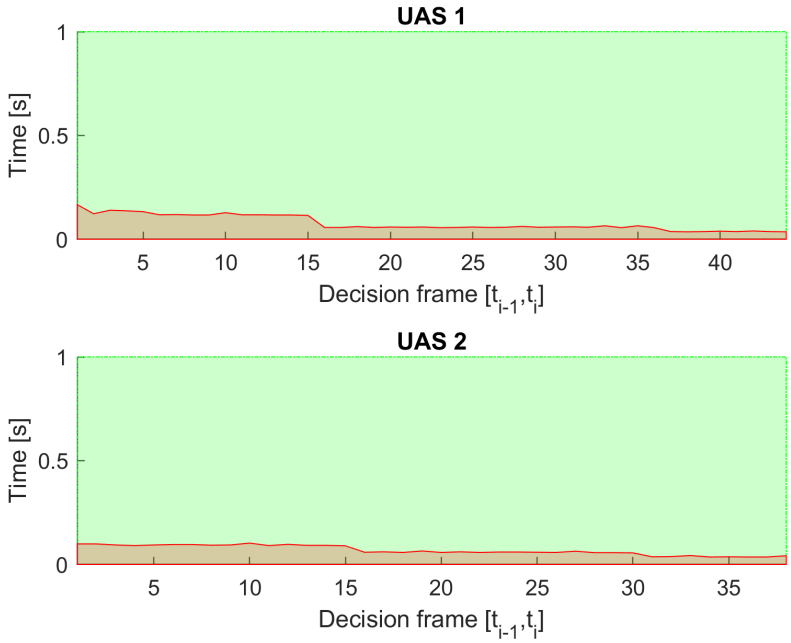



Figure 7.32: Computation time for *Rule-based converging* scenario.

### 7.4.2 Rule based Head on

**Scenario:** Two *UAS* are going on same *airway* in same *flight level* in opposite direction in *controlled airspace* (over 500 feet Above the Ground Level). The *mutual position* of UAS can be classified as *Side Approach*. Following *collision hazard* is present:

1. *Head on Collision Hazard* - There is an *UAS* approaching from opposite direction which invokes need to actively avoid *Collision Point*.

*Head on Collision Hazard* must be addressed by *UTM* service in the following manner:

1. *Each UAS* in particular *Controlled Space* periodically sends synchronized *Position Notification* messages (tab. 6.6).
2. *UTM* service receives *Position Notifications* and manages *Collision Cases* (tab. 6.7) in *Controlled Space*.
3. *UTM* detects single *Head on Collision Cases* with *Collision Point* in vicinity.
4. *UTM* service creates *Virtual Roundabout* and implements *Normative Directive* on both *UAS*.

*Mission parameters* for four UAS systems are defined in (tab. 7.39).

| UAS | Position | | $\mathcal{WP}_1$ |
|---|---|---|---|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[45, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0°, 0°, 180°]^T$ | $[-5, 20, 0]^T$ |

Table 7.39: Mission setup for *Rule based head on* scenario.

**Assumptions:** Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover airworthy *UAS* has capability to precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for (UAS $\leftrightarrow$ UAS) and (UAS $\leftrightarrow$ UTM) communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.
3. *Decision frame synchronization with UTM* - necessary in discrete $C2$ environment otherwise *safety margins* needs to be *bloated*.
4. *Both UAS have identical cruising speed* - simplification impacting *UTM* service implementation. *Obstacle Avoidance Framework* can comprehend various intruders speed, with proper *UAS* directives.

**Main Goal:** Show possibility of *Head on situation resolution* with *forced safety margin* by *UAS Traffic Management* system. The *Obstacle Avoidance Framework based on Reach Sets* is used as *Navigation Module*.

**Acceptance Criteria:** Following criteria must be met:

1. *Well Clear Condition valid for both UAS* - Both *UAS* must have *minimal required distance* from *other UAS* for all *Virtual Roundabout* enforcement time.

2. *Fulfillment of UTM Directives* - Both UAS must stay in *Navigation mode* for all *Virtual Roundabout* enforcement time. Both *UAS* must stay on *Virtual Roundabout* for necessary time, before leaving for *Original Navigation waypoint* $\mathcal{WP}_1$.

**Testing Setup:** The *standard test setup* for each UAS defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Navigation grid - type - ACAS-like* with enabled *Horizontal maneuvers*

This *configuration* is based on assumption that both UAS is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for *climb or descent maneuver*. *Rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.33).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.7) based on incoming *UAS position notifications* (tab. 6.6).

**Simulation Run:** Notable moments from the *simulation run* (fig. 7.33) are following:

1. *Collision Case creation* (fig. 7.33a) following events happens in this step:

   a. Two UAS are on same airway approaching each other from opposite direction, UAS 1 (blue) from the left, UAS 2 (cyan) from the right.

   b. They are going to *collide* at point $\mathcal{C} = [20, 20, 0]^T$ of *Flight Level* (Elevation is 45, 000 feet Above Mean Sea Level).

   c. UTM service notices future *Collision Situation* and creates *Collision Case*.

   d. *Virtual Roundabout* is created at *collision point* with radius $10m$. UTM issues directive for both UAS to avoid collision point from different sides.

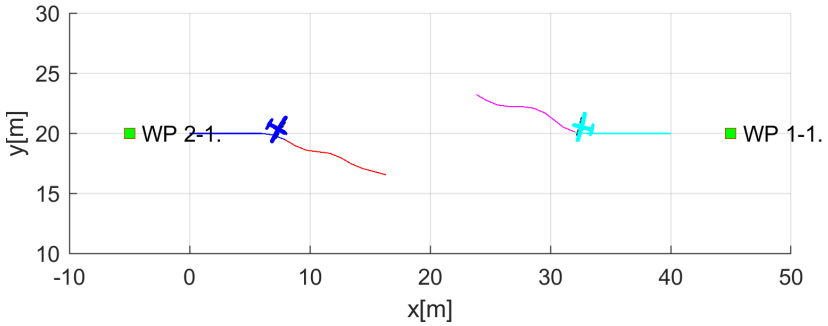
   e. UAS 1 (blue) receives directive to avoid *Collision Point* from *right side* (Down side in GCS). UAS 2 (cyan) receives directive to avoid *Collision Point* from *right side* (Up side in GCS).

   f. Both UAS enters into *Virtual Roundabout*.

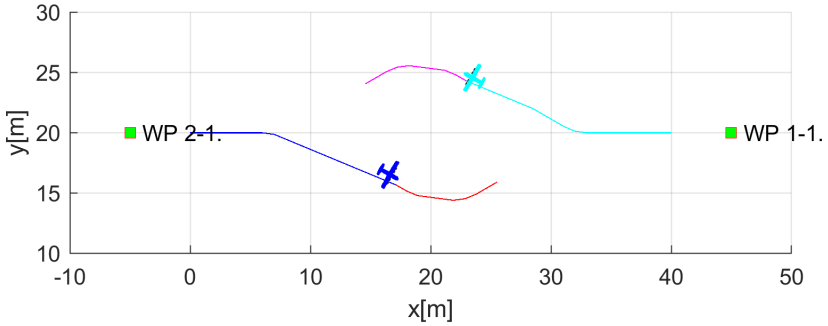
2. *Well clear before* (fig. 7.33b) UAS 1 (blue) is keeping *enforced safety margin* (10 m) from *collision point* and *UAS 2 position*. The *Virtual Roundabout* is enforced until the (*Collision point*) is reached by both UAS. Both UAS stays in *Navigation Mode*.

3. *Well clear after* (fig. 7.33c) UTM notices that *Collision point level* has been reached by both UAS. UTM renounce *Directives* and enables a return to *Original Waypoint* $\mathcal{WP}_1$. Both UAS starts to converging to *Original waypoint* (because possible collision was averted).

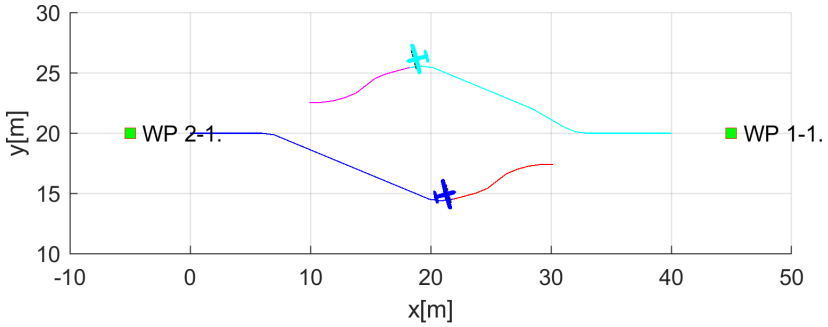
4. *Waypoint reach* (fig. 7.33d) Both UAS reaches respective goal points.



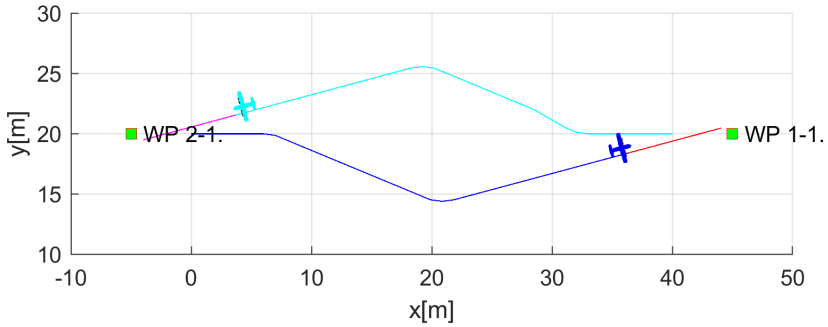

(a) Collision case creation.




(b) Well clear before.




(c) Well clear after.




(d) Waypoints reach.

Figure 7.33: Test scenario for *Rule based head on approach* (virtual roundabout).

**Collision Case Calculation:** For test scenario in (fig. 7.33) where UAS 1 (blue) have head on collision with UAS 2 (cyan), *Collision Case* have been calculated (tab. 7.40).

The *Collision point* is at $[20, 20, 0]^T$ in Flight Level $FL450$ coordinate frame.

The *angle of approach* was evaluated as 180° which indicates *Head on Approach* due the $130° \leq angleof Approach \leq 180°$ condition.

The *mutual position* of UAS 1 (blue) and UAS 2 (cyan) is giving the roles of *Round-*

*about* to *both* UAS.

The *safety margin* for *Well Clear* was determined as $5m$ for UAS 1 and UAS 2. The combined *Case Margin* is 10 m, which is sum of both. The *mutual distance* can not go below this threshold.

| Collision Case | | | | | | Margins | |
|---|---|---|---|---|---|---|---|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Roundabout | $[20, 20, 0]^T$ | 180° | Head on | 5 | 10 |
| | 2 | Roundabout | | | | 5 | |

Table 7.40: Collision case for *Rule-based head on* scenario.

**Distance to Safety Margin Evolution:** The safety margin values (well clear) (fig. 7.34) in controlled airspace are much larger than in non-controlled airspace (near miss) (fig. 7.22).

The enforced rule was (rule 6.12) with parameters: Collision Point $[20, 20, 0]^T$ and *Safety Margin* 10 $m$ as given by Collision Case (tab. 7.40).

The mutual *UAS distance* (blue line) does not go over *Safety Margin* (red line) which means both UAS well clear margins are not broken by any means (fig. 7.33).

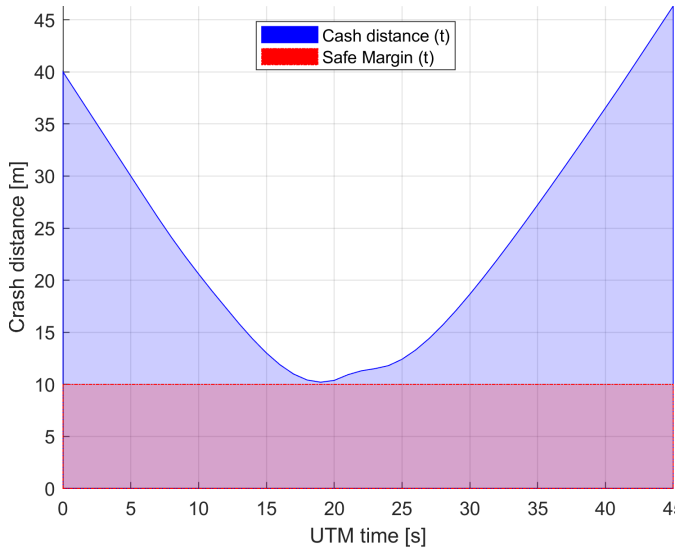



Figure 7.34: Distance to safety margin evolution for *rule based head on scenario.*

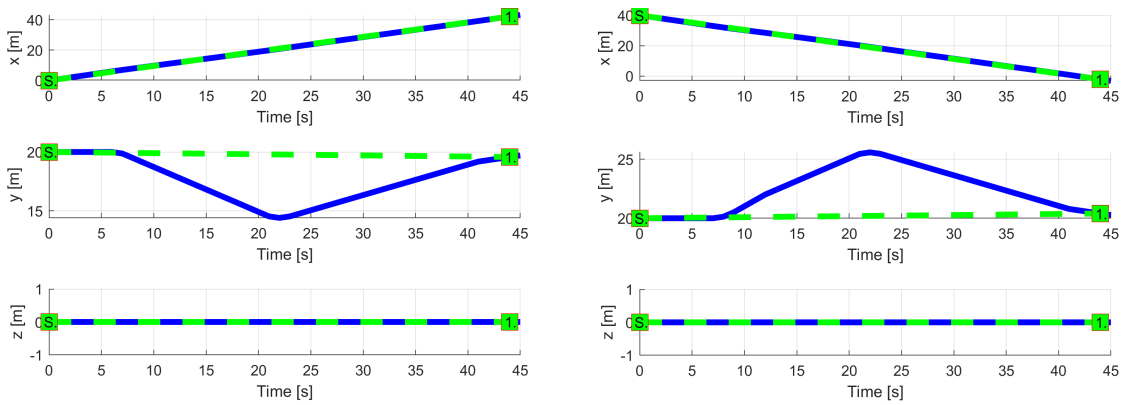
**Distance to Safety Margin Peaks:** Given by (tab. 7.41) represents the proximity on UAS mutual distance to *well clear condition* breach. The breach of *well clear condition* was not achieved. The *minimal distance to safety margin* was 0.2084 $m$. The *maximal distance to safety margin* was $36.3253m$ which represents distance at *Collision Case* closing.

| UAS: | Distance to Safety Margin | | |
|---|---|---|---|
| | min | max | breach |
| 1-2 | 0.2084 | 36.3253 | false |

Table 7.41: *Rule based head on* safety margin distances.

**Path Tracking Performance:** *Path tracking* is displayed in (fig. 7.35). The *UAS* trajectory is divined into *X, Y, Z axis tracking over UTM Time*. The *Reference Trajectory* (green dashed line) interconnect starting position of UAS (green square marked S) an goal waypoint (green square marked 1). The *Executed Trajectory* (blue solid line) reflects real UAS trajectory.

1. UAS 1. (fig, 7.35a) do steady right side *roundabout maneuver* (y-axis).
2. UAS 2. (fig. 7.35b) do steady right side *roundabout maneuver* (y-axis).




(a) UAS 1. (b) UAS 2.

Figure 7.35: *Trajectory tracking* for *Rule based head on* test case.

**Path Tracking Deviations:** Deviations (tab. 7.42) are in *expected ranges*, considering the *mission plans* (tab. 7.39) and *Collision Case* safety margin of $10m$.

| Param. | UAS 1 | UAS 2 |
|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 0 | 0 |
| $\max \lvert y \rvert$ | 5.40 | 5.40 |
| $\max \lvert z \rvert$ | 0 | 0 |
| $\max dist.$ | 5.40 | 5.40 |

Table 7.42: Path tracking properties for *Rule based head on* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.36) shows used time (y-axis) over decision frame (x-axis).

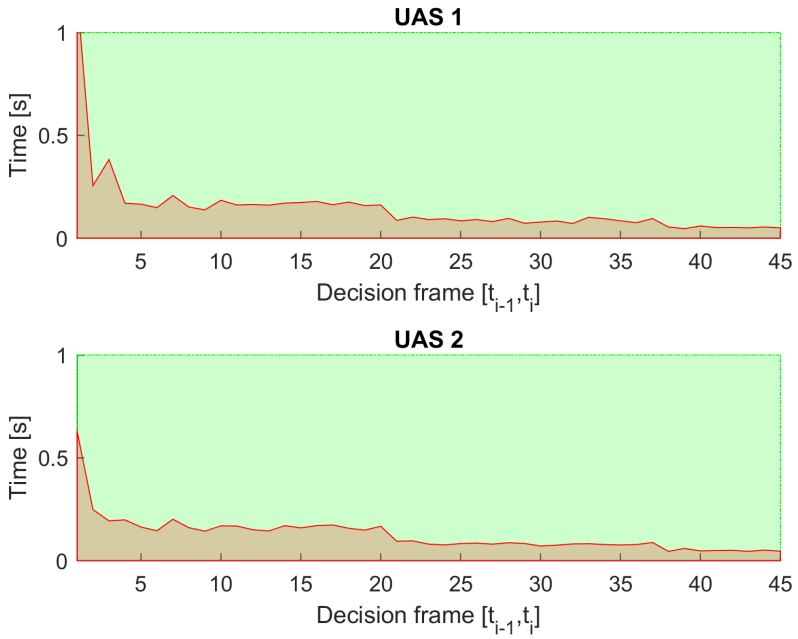



Figure 7.36: Computation time for *Rule-based head on* scenario.

### 7.4.3 Rule based Mixed Head on with Converging

**Scenario:** Four *UAS* are approaching an airway *intersection* at *same time* from *opposite direction* in *controlled airspace* (over 500 feet Above Ground Level). Each *UAS* have following *Collision Hazards*:

1. *Head on Collision Hazard* - There is an *UAS* approaching from opposite direction which invokes need to actively avoid *Collision Point*.
2. *Active Converging Collision Hazard* - There is an *UAS* approaching from the *right side*, which give him *Right of the Way* and invokes need to actively avoid *Intruder*.
3. *Passive Converging Collision Hazard* - There is an *UAS* approaching from the *left side*, which gave us *Right of the Way* and imposes an obligation of *active avoidance* on other *UAS*.

*Note.* Presented scenario is *the worst possible situation* in current *manned aviation ATM*.

*Mentioned Collision Hazards* must be addressed by *UTM* service in the following manner:

1. *Each UAS* in particular *Controlled Space* periodically sends synchronized *Position Notification* messages (tab. 6.6).
2. *UTM* service receives *Position Notifications* and manages *Collision Cases* (tab. 6.7) in *Controlled Space*.
3. *UTM* detects multiple *Collision Cases* with *Collision Points* in vicinity.
4. *UTM* service creates *Virtual Roundabout* and implements *Normative Directive* on all *UAS* in area.

*Mission parameters* for four UAS systems are defined in (tab. 7.43).

| UAS | Position | | $\mathcal{WP}_1$ |
|---|---|---|---|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[0, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[45, 20, 0]^T$ |
| 2 | $[40, 20, 0]^T$ | $[0°, 0°, 180°]^T$ | $[-5, 20, 0]^T$ |
| 3 | $[20, 0, 0]^T$ | $[0°, 0°, 90°]^T$ | $[20, 45, 0]^T$ |
| 4 | $[20, 40, 0]^T$ | $[0°, 0°, -90°]^T$ | $[45, 20, 0]^T$ |

Table 7.43: Mission setup for *Rule based mixed* scenario.

**Assumptions:** Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover airworthy *UAS* has capability to precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for (UAS $\leftrightarrow$ UAS) and (UAS $\leftrightarrow$ UTM) communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.

3. *Decision frame synchronization with UTM* - necessary in discrete C2 environment otherwise *safety margins* needs to be *bloated*.

4. *Every UAS have identical cruising speed* - simplification impacting *UTM* service implementation. *Obstacle Avoidance Framework* can comprehend various intruders speed, with proper *UAS* directives.

**Main Goal:** Show possibility of *Virtual Roundabout* invoked by *UTM* directives where *Obstacle Avoidance Framework based on Reach Sets* is used as a *Navigation Module*.

**Acceptance Criteria:** Following criteria must be met:

1. *Well Clear Condition valid for every UAS* - Each *UAS* must have *minimal required distance* from *other UAS* for all *Virtual Roundabout* enforcement time.

2. *Fulfillment of UTM Directives* - Each UAS must stay in *Navigation mode* for all *Virtual Roundabout* enforcement time. Each *UAS* must stay on *Virtual Roundabout* for necessary time, before leaving for *Original Navigation waypoint* $\mathcal{WP}_1$.

**Testing Setup:** The *standard test setup* for each UAS defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Navigation grid - type - ACAS-like* with enabled *Horizontal maneuvers*

This *configuration* is based on assumption that every UAS is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for *climb or descent maneuver*. *Rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.33).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.7) based on incoming *UAS position notifications* (tab. 6.6).

**Simulation Run:** Notable moments from the *simulation run* (fig. 7.37) are following:

1. *Collision cases created* (fig. 7.37a) following events happens in this step:

   a. Four *UAS* are approaching airways intersection: *UAS 1* (blue) from left, *UAS 2* (cyan) from right, *UAS 3* (green) from bottom, *UAS 4* (black) from top.

   b. They are going to collide at point $[20, 20, 0]^T$ of *Flight level* (elevation is 45, 000 feet Above Mean Sea Level).

   c. *UTM service* notices future *Collision Situations* and creates *Collision Cases*.

   d. There are many *Collision Cases* in near vicinity. The *Virtual Roundabout* is created with *Safety margin* 15 $m$.

   e. The *UTM* service then sends a new *Roundabout Directives* to involved *UAS* systems.

   f. Each *UAS* starts *Roundabout Entry Maneuver* by correcting own *Heading* and *Speed* (if its necessary).

2. *Roundabout entry* (fig. 7.37b) - Each *UAS* enters into *Virtual Roundabout* while sending *Roundabout Entrance Notification* to *UTM service*.

3. *Roundabout leave* (fig. 7.37c) following events happens in this step:

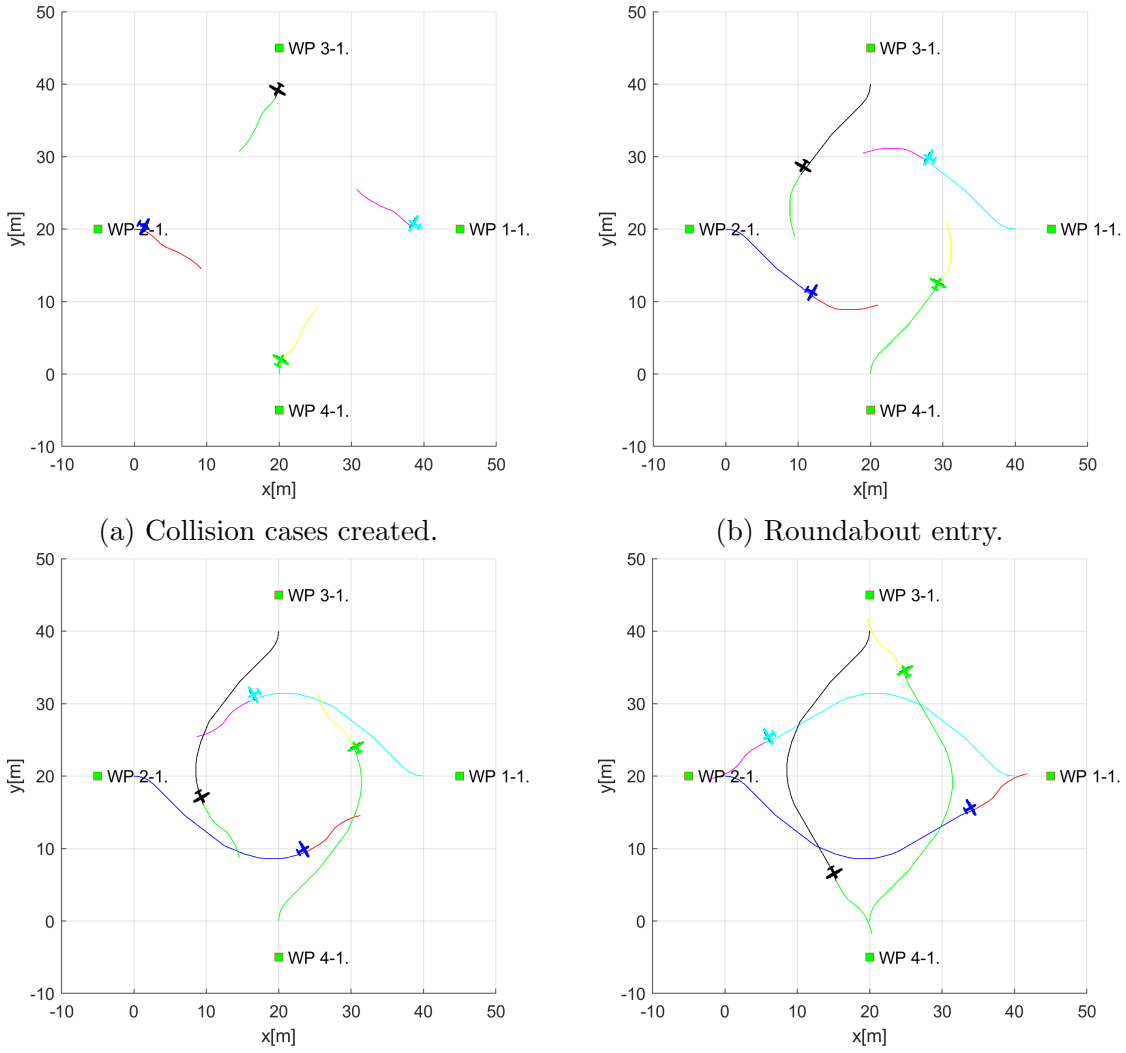
   a. Each *UAS* when is going to approach level of *Original Goal Waypoint* sends *Roundabout Leave Request*.

   b. UTM system will check if there is *Sufficient Free Space* to leave *Virtual Roundabout*.

   c. The *UTM Service then issues Virtual Roundabout Leave Approval*.

   d. Each *UAS* will correct own heading and speed in range of received permit.

4. *Situation resolution* (fig. 7.37d) - Each *UAS* is heading away from *Roundabout Center*, there is no active user of *Virtual Roundabout*. *UTM* will remove *Virtual Roundabout* and closes underlying *Collision Cases*. Each *UAS* will reach respective *Original Goal Waypoint*.




(a) Collision cases created.

(b) Roundabout entry.

(c) Roundabout leave.

(d) Situation resolution.

Figure 7.37: Test scenario for *Rule based mixed* situation with *self-separation mode*.

**Collision Cases Calculation:** The set of original *Collision cases* is given in (tab. 7.44).

Each *UAS* has one *Head on*, *Converging passive*, *Converging active* collision hazard. For example: *UAS 1* have *head on* with *UAS 2*, *converging passive* with *UAS 4*, *converging active* with *UAS 3*. For *UAS 2-4* check *role* in respective *Collision Cases*.

*Note. Collision case* calculated by *UTM* are symmetric, which means that collision case for *UAS X, UAS Y* is identical to collison case calculated for *UAS Y, UAS X*, $X \neq Y$.

*Safety margin* representing *Well Clear Margin* for single *UAS* in *Collision Case* ranges $5 - 8$ $m$. *Case margin* representing minimal mutual distance between two *UAS systems* to remain well clear, ranges $12 - 15$ $m$.

*Merged Collision Case* is oversimplified for demonstration purposes. *Merge Case Procedure* is out of scope of this work due to its extent. Every *Collision Case* share same *Collision Point* $[20, 20, 0]^T$ in flight level coordinate frame. *Merged Collision Case* type was set as *Roundabout*, due the number of collision case *attendants* is greater than 2. Each *UAS role* has been set as *Roundabout*. Enforced *safety margin* is equal to 15 $m$, which is maximum of all *single collision case combined margins*.

| Collision Case | | | | | | Margins | |
|---|---|---|---|---|---|---|---|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Roundabout | $[20, 20, 0]^T$ | 180° | Head on | 8 | 15 |
| | 2 | Roundabout | | | | 7 | |
| 1-3 | 1 | Converging | $[20, 20, 0]^T$ | 90° | Converging | 8 | 15 |
| | 3 | Right o.W. | | | | 5 | |
| 1-4 | 1 | Right o.W. | $[20, 20, 0]^T$ | 90° | Converging | 8 | 15 |
| | 4 | Converging | | | | 5 | |
| 2-3 | 2 | Right o.W. | $[20, 20, 0]^T$ | 90° | Converging | 7 | 12 |
| | 3 | Converging | | | | 5 | |
| 2-4 | 2 | Converging | $[20, 20, 0]^T$ | 90° | Converging | 7 | 12 |
| | 4 | Right o.W. | | | | 5 | |
| 3-4 | 3 | Roundabout | $[20, 20, 0]^T$ | 180° | Head on | 7 | 14 |
| | 4 | Roundabout | | | | 7 | |
| Merged cases | | | | | | Safety Margin | |
| id | UAS | role | collision point | | type | | |
| 1-2-3-4 | 1 | Roundabout | $[20, 20, 0]^T$ | | Roundabout | 15 | |
| | 2 | Roundabout | | | | | |
| | 3 | Roundabout | | | | | |
| | 4 | Roundabout | | | | | |

Table 7.44: Collision cases for *Rule-based mixed* scenario.

**Distance to Safety Margin Evolution:** *Merged Collision Case Safety Margin* is 15 $m$ and it is valid for all *UAS mutual distances*. The simple condition for *Remain Well Clear* is:

$$crashDistance(UAS_X, UAS_Y, t) \geq 15m, X \neq Y \in \{1, 2, 3, 4\}, t \in utmTime$$

*Safety Margin Performance* is given in (fig. 7.38). The mutual distance (Crash Distance [m]) between two UAS is denoted as *blue line*. The enforced safery margin for *Remain Well Clear* condition is denoted as red line.

*Note. Evolution of mutual crash distance* is symmetric. In any case the mutual distance goes under *safety margin*. *Acceptance criterion* for *Well Clear condition* is fulfilled.

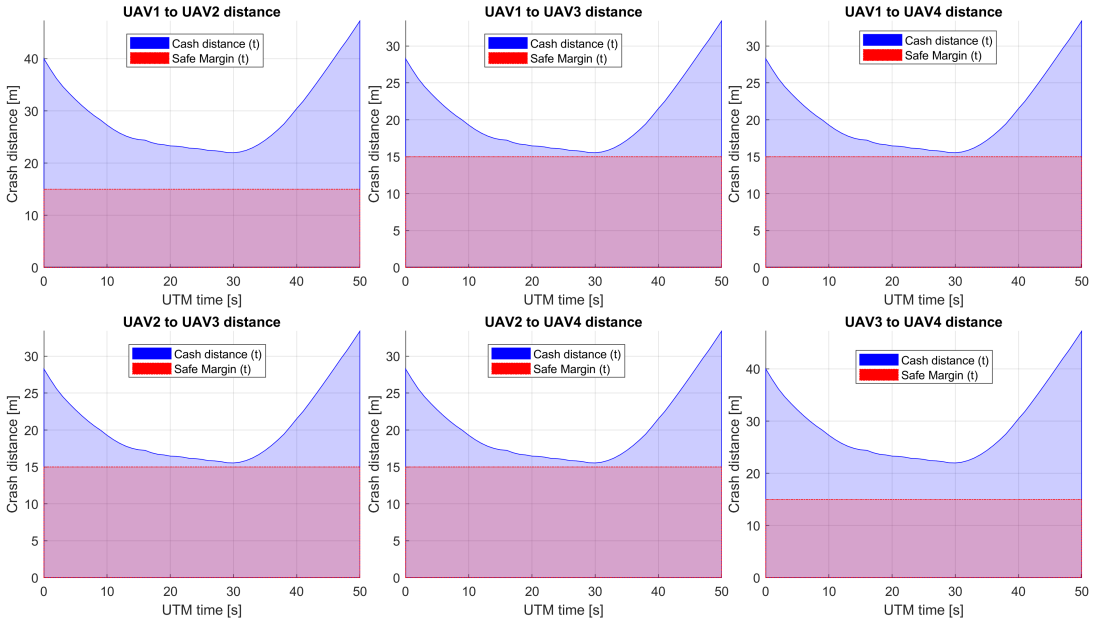



Figure 7.38: Distance to safety margin evolution for *rule based mixed scenario*.

**Distance to Safety Margin Peaks:** *Distance to Safety Margin Peaks* (tab. 7.45) represents the proximity of *UAS mutual distance to breach well clear condition*. The *breach condition* was not fulfilled in any combination.

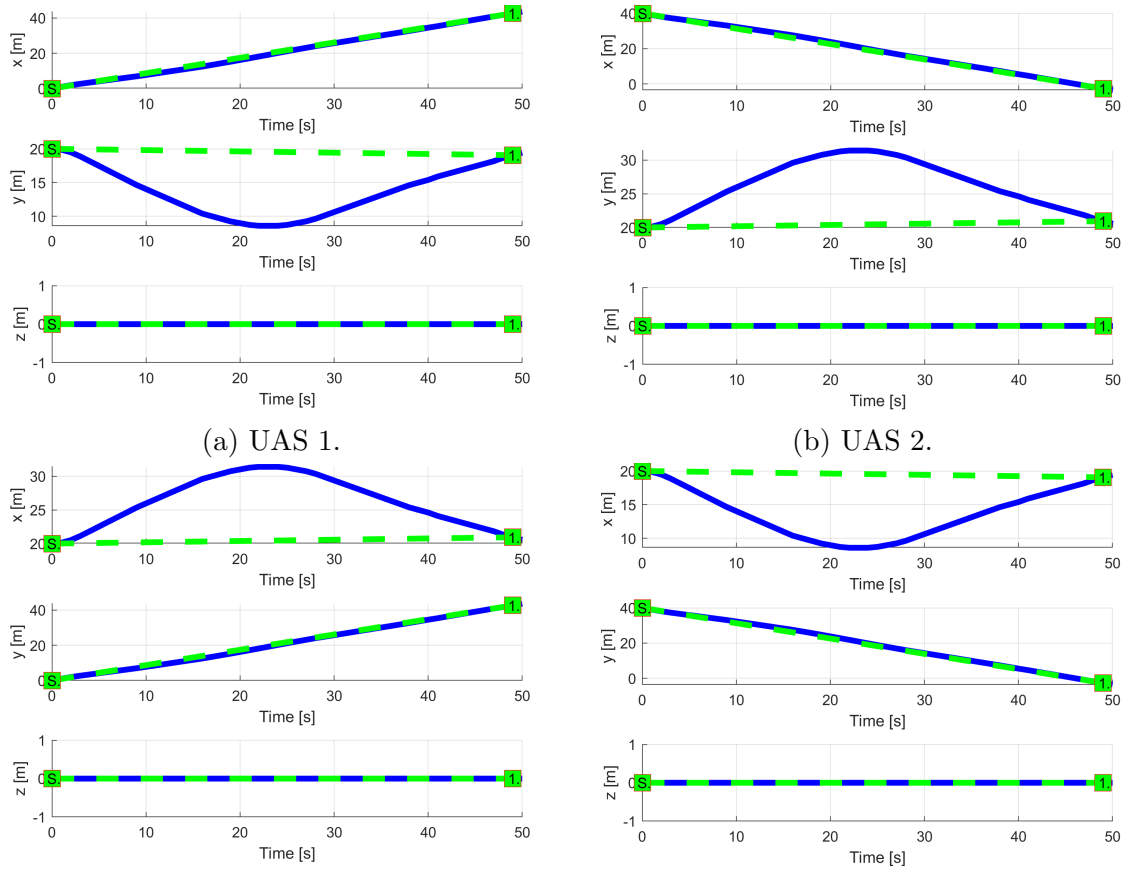
The *minimal distance to safety margin* was 0.5438 $m$ between all four *UAS* systems. The *maximal distance to safety margin* ranges between *18 - 32 m* which shows advantages of *virtual roundabout*.

| UAS: | Distance to Safety Margin | | |
|---|---|---|---|
| | min | max | breach |
| 1-2 | 6.9823 | 32.2369 | false |
| 1-3 | 0.5438 | 18.4015 | false |
| 1-4 | 0.5438 | 18.4015 | false |
| 2-3 | 0.5438 | 18.4015 | false |
| 2-4 | 0.5438 | 18.4015 | false |
| 3-4 | 6.9823 | 32.2369 | false |

Table 7.45: Distance to safety margin peaks for *rule based mixed scenario*.

**Path Tracking Performance:** Path tracking is displayed in (fig. 7.39). The UAS trajectory is divined into *X, Y, Z axis tracking over UTM Time*. The *Reference Trajectory* (green dashed line) is represented as interconnection between *Start Waypoint* (green square marked S) and *Goal Waypoint* $\mathcal{WP}_1$ (green square marked 1). The *Executed trajectory* (blue solid line) reflects real *UAS* movement.

1. *UAS 1* (fig. 7.39a) is using bottom portion of *Virtual Roundabout* (-Y values), sticking to the boundary of the *Virtual Roundabout*.
2. *UAS 2* (fig. 7.39b) is using upper portion of the *Virtual Roundabout*. (+Y values), sticking to the boundary of the *Virtual Roundabout*.
3. *UAS 3* (fig. 7.39c) is using right portion of the *Virtual Roundabout*. (+X values), sticking to the boundary of the *Virtual Roundabout*.
4. *UAS 4* (fig. 7.39d) is using left portion of the *Virtual Roundabout*. (-X values), sticking to the boundary of the *Virtual Roundabout*.




(a) UAS 1.

(b) UAS 2.

(c) UAS 3.

(d) UAS 4.

Figure 7.39: Trajectory tracking for *Rule based mixed* situation test case.

**Path Tracking Deviations:** *Deviations* (tab. 7.46) are in expected ranges, considering the mission plans (tab. 7.43) and *Merged Case Safety Margin* (15 $m$).

| Param. | UAS 1 | UAS 2 | UAS 3 | UAS 4 |
|---|---|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ | $\mathcal{WP}_1$ |
| $\max \lvert x \rvert$ | 0 | 0 | 11.40 | 11.40 |
| $\max \lvert y \rvert$ | 11.40 | 11.40 | 0 | 0 |
| $\max \lvert z \rvert$ | 0 | 0 | 0 | 0 |
| $\max dist.$ | 11.40 | 11.40 | 11.40 | 11.40 |

Table 7.46: Path tracking properties for *Rule based mixed* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.40) shows used time (y-axis) over decision frame (x-axis).

The *computation time* for each UAS has same evolution. The *load* is higher during avoidance maneuver on *virtual roundabout*.

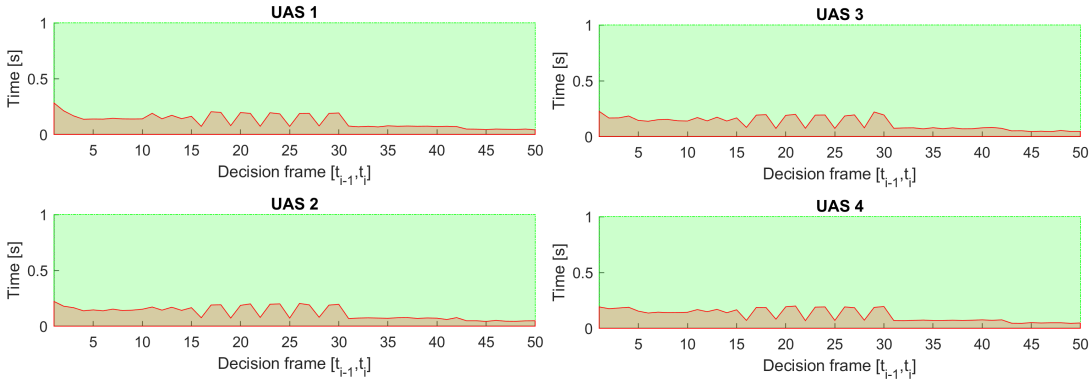



Figure 7.40: Computation time for *Rule-based multiple* scenario.

### 7.4.4 Rule based Overtake

**Scenario:** Two UAS are flying in the *controlled airspace* (over 500 feet Above Ground Level) on the *airway* (in same direction). *Slower UAS* is in front of *Faster UAS*. There is possibility of a *collision* or a *near miss incident* or a *well clear breach*. The *Faster UAS* (Overtaking) must contact *UTM* service and ask for *overtake permission*. Scenario steps:

1. *Faster UAS* (Overtaking) notices *UTM* service about *Slower UAS* (Overtaken). (This step is Optional.)
2. *UTM* service issues *Directives* to all *UAS* in area.
3. *Overtake Directive* is received by *Faster UAS* (Overtaking) and *Slower UAS* (Overtaken).
4. *Faster UAS* (Overtaking) mission plan is altered to reflect *Overtake directive*, *Divergence Waypoint* and *Convergence Waypoint* are added.
5. *Faster UAS* (Overtaking) safely overtakes *Slower UAS* (Overtaken) without breaking *Well clear* condition.

*Mission parameters* for both *UAS* systems are defined in (tab. 7.47).

| UAS | Position | | $\mathcal{WP}_1$ |
|---|---|---|---|
| | $[x, y, z]$ | $[\theta, \varpi, \psi]$ | |
| 1 | $[-40, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[110, 20, 0]^T$ |
| 2 | $[-20, 20, 0]^T$ | $[0°, 0°, 0°]^T$ | $[80, 20, 0]^T$ |

Table 7.47: Mission setup for all *Rule based overtake* scenarios.

**Assumptions:** Following assumptions are valid for this test:

1. *Controlled Airspace Airworthiness* - UAS system is equipped with necessary controlled airspace equipment like ADS-B In/Out, Radar, Transponder, etc. Moreover airworthy *UAS* has capability to precisely follow *UTM directives* (max. 5 % deviation).
2. *C2 (Command & control) Link Established* - necessary for (UAS $\leftrightarrow$ UAS) and (UAS $\leftrightarrow$ UTM) communication. If *C2* link is lost the *UAS* will enter into *Emergency avoidance mode*.
3. *Decision frame synchronization with UTM* - necessary in discrete C2 environment otherwise *safety margins* needs to be *bloated*.

**Main Goal:** Show possibility of *Overtake Maneuver* invoked by the *UTM Directive* (event based flight constraint).

**Acceptance Criteria:** Following criteria must be met:

1. *Proper passing of Divergence/Convergence Waypoint* - minimal distance of *UAS trajectory* to *Divergence/Convergence waypoint* must be below passing threshold. Waypoints needs to be passed in given order (Divergence $1^{st}$, Convergence $2^{nd}$).

2. *Slower UAS (Overtaken) keeps Right of the Way* - the UAS with lesser maneuverability does not stand a chance in avoidance situation, it needs to keep its *Right of the Way*.

3. *Both UAS does not breach Well Clear (safety) Margin* - mutual distance does not get trough *calculated Safety Margin*.

**Testing Setup:** The *standard test setup* for each UAS defined in (tab. 7.2, 7.3, 7.4, 7.5, 7.6) is used with following parameter override:

1. *Navigation grid - type - ACAS-like* with enabled *Horizontal maneuvers*

This *configuration* is based on assumption that every UAS is in *controlled airspace* in *FL450* (flight level 45000 feet Above Sea Level), without permission for *climb or descent maneuver*. *Rule engine* is initialized in standard *Rules of the air* configuration (fig. 6.33).

There is *UTM* service for given *airspace cluster* calculating *collision cases* (tab. 6.7) based on incoming *UAS position notifications* (tab. 6.6).

**Simulation Run:** Notable moments from the *simulation run* (fig. 7.41) are following:

1. *Collision case creation* (fig.7.41a) - *Faster UAS* (blue) receives *UTM Directive* to invoke *Overtake Rule* (tab. 6.14). *Slower UAS* (magenta) receives *UTM Directive* to keep *Right of the Way* and warning that is going to be *Overtaken*. *Faster UAS* (blue) creates two *virtual waypoints*:

   a. *Divergence waypoint* at position $[0, 14, 0]^T$.

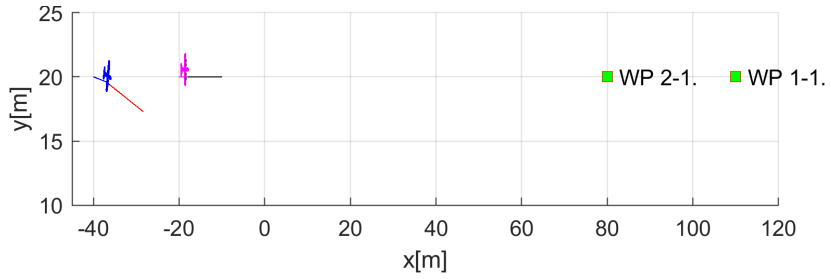
   b. *Convergence waypoint* at position $[24, 14, 0]^T$.

   *Faster UAS* then sets *Divergence waypoint* as *Goal waypoint* and It starts overtake maneuver while checking mutual distance.

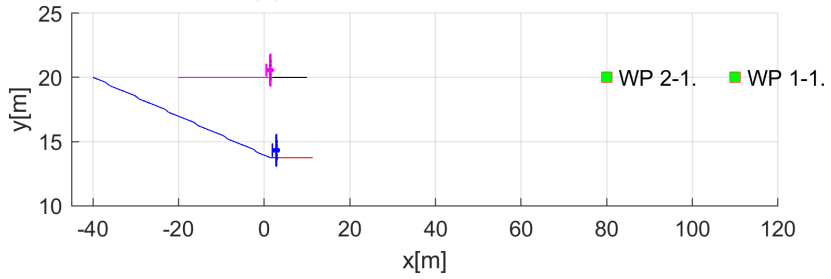
2. *Divergence waypoint reach* (fig. 7.41b) - *Faster UAS* (blue) successfully reached *Divergence Waypoint*, setting *Convergence Waypoint* as new *Goal waypoint*.

3. *Convergence waypoint reach* (fig. 7.41c) - *Faster UAS* (blue) successfully reached *Convergence Waypoint*, setting *Original Goal Waypoint* as new *Goal waypoint*. The *UTM* service is notified from *Faster UAS* (blue) that *Overtaken Maneuver* have been completed. UTM *acknowledges* maneuver competition and It sends notification to *Slower UAS* (magenta) that *Overtake Maneuver* is finished. *Slower UAS* (magenta) was successfully overtaken.

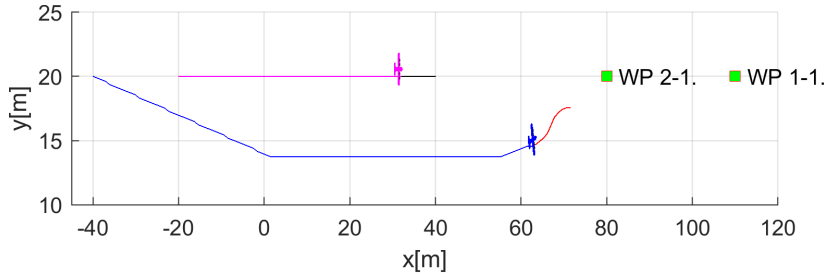
4. *Original waypoint reach* (fig. 7.41d) - *Faster UAS* (blue) successfully reached *Original Waypoint*, Starting landing Sequence.



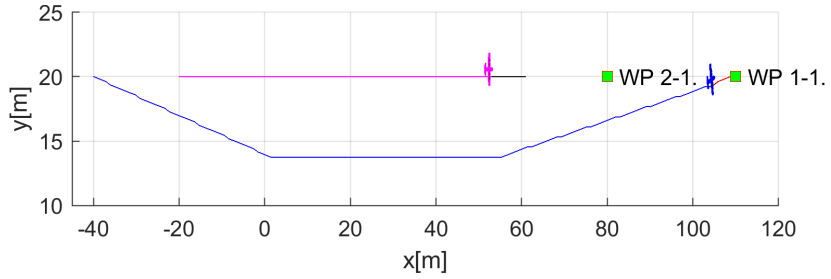

(a) Collision case creation.




(b) Divergence waypoint reach.




(c) Convergence waypoint reach.




(d) Original waypoint reach.

Figure 7.41: Test scenario for *Rule based Overtake* (double speed of overtaking aircraft).

**Collision Case Calculation:** The *Collision Case* (tab. 7.48) was calculated according to *Collision Calculation process* (sec. 6.8.8). *Faster UAS* (1) has *Overtaking* role and *Slower UAS* has *Right of the Way*. *Collision Point* is direct type at $[0.20.0]^T$. *Collision case type* was set based on *angle of approach* 0° as *Overtake*. The *Safety Margin* was set as *5 m*.

| Collision Case | | | | | | Margins | |
|---|---|---|---|---|---|---|---|
| id | UAS | role | collision point | angle of approach | type | safety | case |
| 1-2 | 1 | Overtaking | $[0, 20, 0]^T$ | 0° | Overtake | 5 | 5 |
| | 2 | Right o.W. | | | | 5 | |

Table 7.48: Collision case for *Rule-based Overtake* scenario 2x speed.

**Overtake Speed: Divergence/Convergence Waypoints** *Divergence waypoints* have been calculated according to (eq. 6.187), and, *Convergence Waypoints* have been calculated according to (eq. 6.188). Following *Speed Differences* were taken into account (Faster/Slower UAS speed ratio): *2x, 3x, 4x*. Following observations can be made:

1. *Distance between Divergence and Convergence waypoint* is decreasing with increasing *speed difference*.
2. *Divergence waypoint* is moving *back/right* in *UAS Local Coordinate Frame* with Increasing *speed difference*.
3. *Convergence waypoint* is moving like *Divergence waypoint* but little bit faster.

| Speed | Divergence | | Convergence | | Final |
|---|---|---|---|---|---|
| diff. | waypoint | difference | waypoint | difference | waypoint |
| 2x | $[0, 14, 0]^T$ | $[-10, -1, 0]^T$ | $[24, 14, 0]^T$ | $[-8, -1, 0]^T$ | $[110, 20, 0]^T$ |
| 3x | $[-10, 13, 0]^T$ | $[-3.4, -1, 0]^T$ | $[16, 13, 0]^T$ | $[-1.3, -1, 0]^T$ | $[110, 20, 0]^T$ |
| 4x | $[-13.4, 12, 0]^T$ | | $[14.7, 12, 0]^T$ | | $[110, 20, 0]^T$ |

Table 7.49: Convergence and divergence waypoints for various speed differences.

**Overtake Speed: Impact on Trajectory** Overtake *speed difference* is visible in (fig. 7.42). The *Slower vehicle trajectory*(cyan) is following *standard mission waypoints*. The *Faster vehicle trajectory* for 2x (blue), 3x (green), 4x (black) are following *Divergence/Convergence* waypoints from (tab. 7.49).

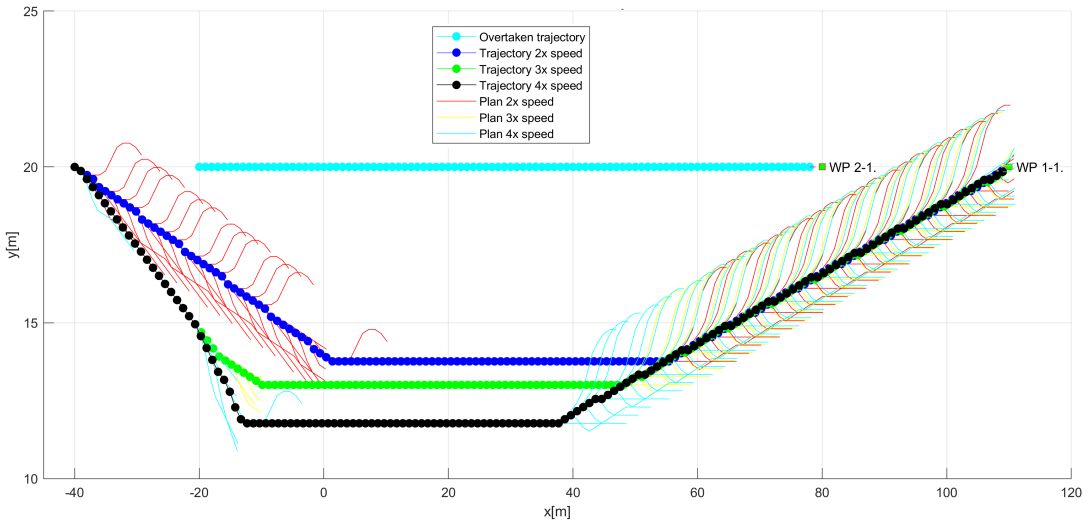



Figure 7.42: *Rule based overtake* trajectories for different speed.

**Overtake Speed: Impact on Distance to Safety Margin Evolution** *Safety margin* (red line) is set to *5 m*. It is obvious that *Faster UAS* will take down *Slower UAS* if there was not for an *Overtake maneuver*. The distance of *Faster UAS* to *Slower UAS* evolution is depending on *Speed difference*. *Inflection point* (closest point of two UAS) is reached sooner with *Higher speed*. *Safety margin performance* was measured for the *UTM performance time* in interval $[0, 35]$ *s* and *Speed difference* of 2x (blue), 3x (green), 4x (black).

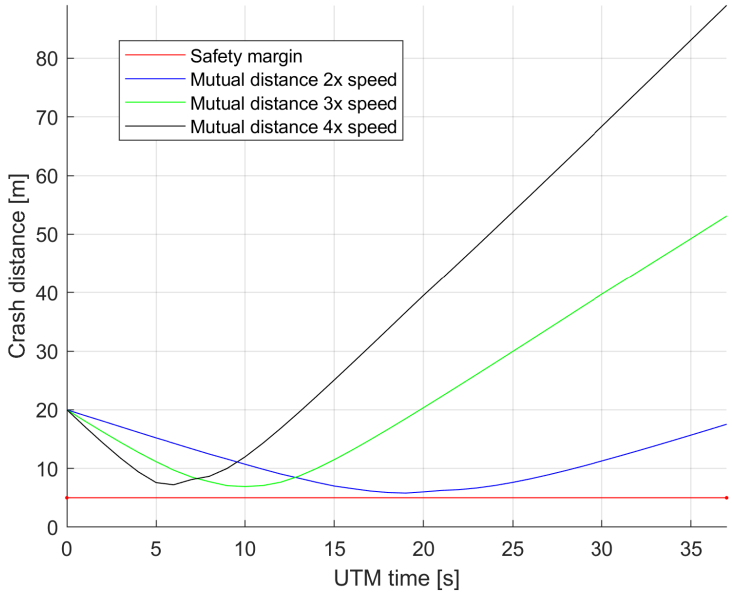



Figure 7.43: Overtake speed dependent distance to safety margin evolution for *rule based overtake scenario*.

**Overtake Speed: Impact on Distance to Safety Margin Peaks** There is summary table (tab. 7.50) for measurement of minimal and maximal values for *Distance to Safety Margin* over *UTM time* (fig.7.43). The minimal *Overtake Distance to Safety Margin* in 0.7991 *m* for 2x *Speed Difference*. The minimal *Overtake closest point reach time* is 7 *s* for 4x *Speed Difference*.

For each *Speed difference* (2x, 3x, 4x), the *Well Clear Margin* (Safety Margin) was not reached by the *Faster UAS Body boundary*.

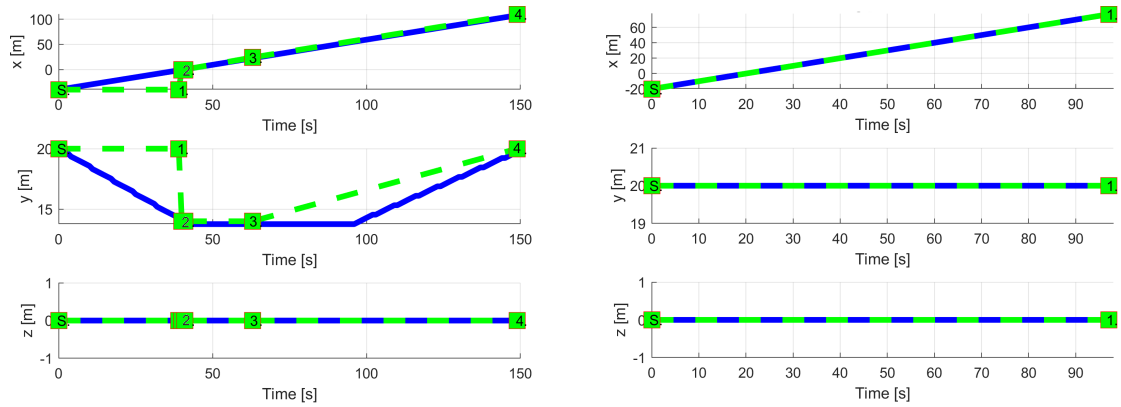
| Speed diff. | Minimal | | Maximal | | Breach |
|---|---|---|---|---|---|
| | distance | time | distance | time | |
| 2x | 0.7991 | 20 | 48.8508 | 76 | false |
| 3x | 1.9180 | 11 | 73.5336 | 51 | false |
| 4x | 2.2154 | 7 | 84.0721 | 38 | false |

Table 7.50: Distance to safety margin peaks for various overtake speed in *Rule based overtake scenario*.

**Path Tracking Performance: 2x Speed** Performance was only evaluated for case when *Faster/Slower UAS speed ratio* is $2x$. All waypoints are marked as green numbered *squares* with number. Initial waypoint is marked as green square with *S*. Reference trajectory is annotated as *green dashed line*. *Executed trajectory is annotated* as *blue solid line*.

Following observations can be made from path tracking (fig. 7.44):

1. *UAS 2 has the Right of the Way* (fig. 7.44b) - *reference trajectory* and *executed trajectory* are identical.
2. *UAS 1 is Overtaking* (fig. 7.44a) - the following waypoins are marked on reference trajectory:
   a. *Collision Point* ($\mathcal{WP}$ 1.) - this is not used for navigation, its marking of *Collision Point*.
   b. *Divergence waypoint* ($\mathcal{WP}$ 2.) - there will *Faster UAS* navigate to avoid *Collision*.
   c. *Convergence waypoint* ($\mathcal{WP}$ 3.) - there will *Faster UAS* navigate to gain *Safe Return Distance*.
   d. *Original Goal Waypoint* ($\mathcal{WP}$ 4.) - there will *Faster UAS* continue until *original goal* is reached.




(a) UAS 1. (b) UAS 2.

Figure 7.44: Trajectory tracking for *Rule based overtake double speed* situation test case.

**Path Tracking Deviations: 2x Speed** Path tracking deviations (tab. 7.51) are interesting for an *Overtake Maneuver* performance.

*Maximal deviation distance* is for important waypoints: Divergence ($\mathcal{WP}$ 2.), Convergence ($\mathcal{WP}$ 3.) and Original Goal Waypoint ($\mathcal{WP}$ 4.), equal to 0 $m$. This is *desired effect* for *Overtake maneuver*.

*Collision point* ($\mathcal{WP}$ 1.) is avoided at minimal distance 5.7991 $m$ (tab. 7.50) and maximal distance 24.5 $m$ (tab. 7.51).

Other *Speed Difference Ratios* yields similar results.

| Param. | UAS 1 | | | | UAS 2 |
|---|---|---|---|---|---|
| | $\mathcal{WP}_1$ | $\mathcal{WP}_2$ | $\mathcal{WP}_3$ | $\mathcal{WP}_4$ | $\mathcal{WP}_1$ |
| | col. | div. | conv. | orig. | nav. |
| max $\|x\|$ | 20 | 0 | 0 | 0 | 0 |
| max $\|y\|$ | 6 | 0 | 4 | 5 | 0 |
| max $\|z\|$ | 0 | 0 | 0 | 0 | 0 |
| max $dist.$ | 24.5 | 0 | 4 | 5 | 0 |

Table 7.51: Path tracking properties for *Rule overtake 2x speed* scenario.

**Computation Load:** The *computation load* for *scenario* (fig.7.45) shows used time (y-axis) over decision frame (x-axis).

The load is minimal on both UAS, because the rule calculates only divergence (eq. 6.189) and convergence (eq. 6.190) waypoints.

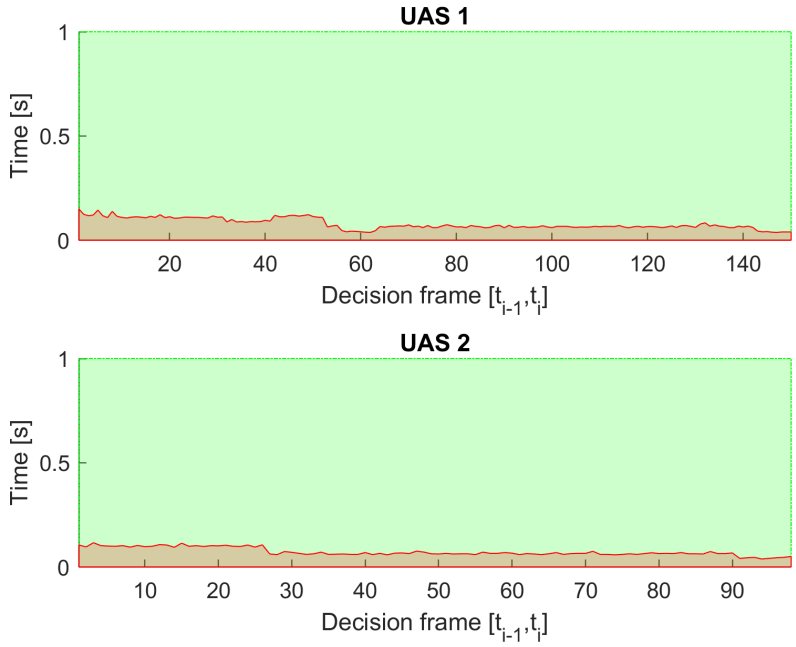



Figure 7.45: Computation time for *Rule based overtake* scenario.

## 7.5 (R) Test Cases Conclusion

This section contains summary of performance evaluation (sec. 7.1.3), adversary behaviour impact on our approach (sec. 7.5.2), *calculation load* in (sec. 7.5.3).

### 7.5.1 (R) Performance Evaluation

Performance of test cases were evaluated according to criteria given by (sec. 7.1.3). The performance for *test cases* from test plan (tab. 7.1) have been summarized in (tab. 7.52).

| Scenario name | Safety Margin | | | Trajectory tracking | | | Pass / Fail |
|---|---|---|---|---|---|---|---|
| | Distance | | Breach | Waypoint Reach | Reference Deviation | Acceptable Deviation | |
| | min | max | | | | | |
| Building avoidance (sim. 7.1) | 0.69 m UAS 1 | 24.98 m UAS 1 | No (7.2) | Yes/UAS 1/(7.3) | $\mathcal{WP}_1$ : 107.05$m$<br>$\mathcal{WP}_2$ : 86.20$m$<br>$\mathcal{WP}_3$ : 28.70$m$<br>$\mathcal{WP}_4$ : 32.84$m$ | Yes (7.10) | Pass |
| Slalom (sim. 7.5) | 0.09 m UAS 1 | 3.74 m UAS 1 | No (7.6) | Yes/UAS 1/(7.7) | $\mathcal{WP}_1$: 20.06$m$ | Yes (7.14) | Pass |
| Maze (sim 7.9) | 0.01 m UAS 1 | 2.95 m UAS 1 | No (7.10) | Yes/UAS 1/(7.11) | $\mathcal{WP}_1$ : 28.06$m$ | Yes (7.18) | Pass |
| Storm (sim. 7.13) | 0.04 m UAS 1 | 34.99 m UAS 1 | No (7.14) | Yes/UAS 1/(7.15) | $\mathcal{WP}_1$: 15.76$m$ | Yes (7.22) | Pass |
| Emergency Converging (sim. 7.17) | 1.67 m UAS 1-2 | 27.08 m UAS 1-1 | No (7.18) | Yes/UAS 1/(7.19a)<br>Yes/UAS 2/(7.19b) | $\mathcal{WP}_1$ : 3.25$m$<br>$\mathcal{WP}_1$ : 0.00$m$ | Yes (7.26) | Pass |
| Emergency Head On (sim. 7.21) | 0.38 m UAS 1-2 | 38.00 m UAS 1-2 | No (7.22) | Yes/UAS 1/(7.23a)<br>Yes/UAS 2/(7.23b) | $\mathcal{WP}_1$ : 3.25$m$<br>$\mathcal{WP}_1$ : 0.00$m$ | Yes (7.30) | Pass |
| Emergency Multiple (sim. 7.25) | 0.20 m UAS 2-4 | 45.46 m UAS 3-4 | No (7.26) | Yes/UAS 1/(7.27a)<br>Yes/UAS 2/(7.27b)<br>Yes/UAS 3/(7.27c)<br>Yes/UAS 4/(7.27d) | $\mathcal{WP}_1$ : 4.84$m$<br>$\mathcal{WP}_1$ : 1.83$m$<br>$\mathcal{WP}_1$ : 3.45$m$<br>$\mathcal{WP}_1$ : 2.05$m$ | Yes (7.34) | Pass |
| Rule-based Converging (sim. 7.29) | 1.22 m UAS 1-2 | 20.28 m UAS 1-2 | No (7.37) | Yes/UAS 1/(7.31a)<br>Yes/UAS 2/(7.31b) | $\mathcal{WP}_1$ : 10.22$m$<br>$\mathcal{WP}_1$ : 0.00$m$ | Yes (7.38) | Pass |
| Rule-based Head On (sim. 7.33) | 0.21 m UAS 1-2 | 36.33 m UAS 1-2 | No (7.34) | Yes/UAS 1/(7.35a)<br>Yes/UAS 2/(7.35b) | $\mathcal{WP}_1$ : 5.40$m$<br>$\mathcal{WP}_1$ : 5.40$m$ | Yes (7.42) | Pass |
| Rule-based Multiple (sim. 7.37) | 0.54 m UAS 2-3 | 32.24 m UAS 1-2 | No (7.38) | Yes/UAS 1/(7.39a)<br>Yes/UAS 2/(7.39b)<br>Yes/UAS 3/(7.39c)<br>Yes/UAS 4/(7.39d) | $\mathcal{WP}_1$ : 11.40$m$<br>$\mathcal{WP}_1$ : 11.40$m$<br>$\mathcal{WP}_1$ : 11.40$m$<br>$\mathcal{WP}_1$ : 11.40$m$ | Yes (7.46) | Pass |
| Rule-based Overtake (sim. 7.41) | 0.80 m UAS 1-2 | 48.85 m UAS 1-2 | No (7.43) | Yes/UAS 1/(7.44a)<br>Yes/UAS 2/(7.44b) | $\mathcal{WP}_1$ : 24.00$m$<br>$\mathcal{WP}_2$ : 0.00$m$<br>$\mathcal{WP}_3$ : 4.00$m$<br>$\mathcal{WP}_4$ : 5.00$m$<br>$\mathcal{WP}_1$ : 0.00$m$ | Yes (7.51) | Pass |

Table 7.52: Test cases *performance evaluation*.

**Highlights:** Each *scenario* contains reference to notable simulation moments and results. The scenarios were grouped according to *Operational Space* category and each category is separated by strike line.
*Non cooperative test cases for Rural/Urban environment:*

1. *Static obstacle avoidance* (Building/Slalom/Maze) - the buildings were correctly avoided without security breach, navigation algorithm was sufficient for given scenarios and obstacle density.

2. *Weather avoidance* (Storm) - the moving *storm* have been avoided in both *soft constraint* and *hard constraint* state. The assumption of *early detection/notification* is key in successful weather avoidance.

*Non cooperative test cases for Intruder Avoidance* - the key assumptions are early intruder detection in *Avoidance Grid* and *non-adversarial* behaviour. Each UAS was running own instance of *Navigation loop* (fig. 6.25). The summary of test cases is going like follow:

1. *Emergency converging* - both UAS identified correct roles according rules of the air. The UAS 2 kept *right of the way*.

2. *Emergency head on* - both UAS identified correct roles according rules of the air, both of them uses full separation with *Combined Reach Set Approximation* (sec. 6.4.5).

3. *Emergency mixed* - all four UAS enters into emergency avoidance mode intermediately after intruders detection. The *non-cooperative* consensus of separation is reached (fig. 6.28)

*Cooperative test cases with UTM supervision* are working according to *UTM architecture* (fig. 6.26), where the *UTM* is considered as main authority. The key assumptions are UTM Resolution fulfillment and *non-adversary behaviour*. Each UAS was running own instance of *Navigation loop* (fig. 6.25) with enabled *Rule Engine* (sec. 6.9). The summary of test cases is going like follow:

1. *Rule-based converging* - correct handling of *converging maneuver* (fig. 6.30), proper rule invocation (rule 6.13) on UAS side.

2. *Rule-based head on* - correct handling of *head on maneuver* (fig. 6.29), proper rule invocation (rule 6.12) on UAS side.

3. *Rule-based multiple* - proper *Collision case Merge* (tab. 7.44) with new collision point (eq. 6.171) and *safety margin calculation* (eq. 6.172).

4. *Rule-based overtake* - correct handling of *overtake maneuver* (fig. 6.31), proper rule invocation (rule 6.14). Divergence/Convergence (eq. 6.187,6.188) for multiple waypoints calculation works for various speed difference (fig. 7.43).

### 7.5.2 (R)Adversary Behaviour Impact

The *abuse* of UAS for *ill intentions* realization is expected. The *UAS* is cheap, disposable and does not have an ethic boundaries.

One of the *assumptions* was that there are only intruders whom does not actively look to harm our *UAS*. Breaking this assumption can be lethal for our system and also for other systems.

Let us take *Rule-based Head on* test case (sec. 7.4.2), changing only following aspects:

1. *UAS 2 position spoofing* - the adversarial vehicle is *faking its position* according to expected behaviour.

2. *UAS 2 Navigation goal* - set as *UAS 1 position* from intercepted *position notifications* (tab. 6.6).

**Simulation:** The *simulation* (fig. 7.46) have been run with defined condition. UAS 2 (magenta) has been chosen as the *adversary*. UTM sees expected trajectory of UAS 2 (grey plane/trajectory) based on spoofed *position notifications*. The *navigation/avoidance grid* range (black dashed line boundary) is shown. The notable moment of simulation are:

1. *Deviation detection (*$UAS2 \leftrightarrow UTM$*)* (fig. 7.46a) - the *collision case* (tab. 7.40) is active and *enforced* by UTM. The *adversary* UAS 2 (magenta) starts deviating from expected trajectory (grey). UAS 1 (blue) does not register any foreign object in *avoidance grid range* (black dashed line).

2. *Adversary attacking (*$UAS2 \rightarrow UAS1$*)* (fig. 7.46b) - the adversary UAS 2 (magenta) starts actively pursuing UAS 1 (blue) by changing original heading. This can be considered as beginning of *active pursuit*. UAS 1 (blue) does not detect any foreign object in *avoidance grid* (black dashed line boundary). *UTM* is receiving expected UAS position (grey plane/line).

3. *Emergency avoidance* ($UAS1 \rightarrow UAS2$) (fig. 7.46c) following happens:

   a. *Adversary UAS 2* (magenta) is spotted by *UAS 1* (blue), it entered into UAS 1 avoidance grid (black dashed line boundary).

   b. *UAS 1* (blue) enters into *Emergency Avoidance Mode*, because there is an *foreign object* in *avoidance grid*.

   c. *UTM* notices a warning to *UAS 1* (blue), because it entered into *Emergency Avoidance Mode*. UTM is not aware of any breach, because of expected UAS 2 position (grey plane/line)

   d. *Adversary UAS 2* (magenta) has UAS 1 (blue) locked in *navigation grid* as goal (which guarantees optimal path).

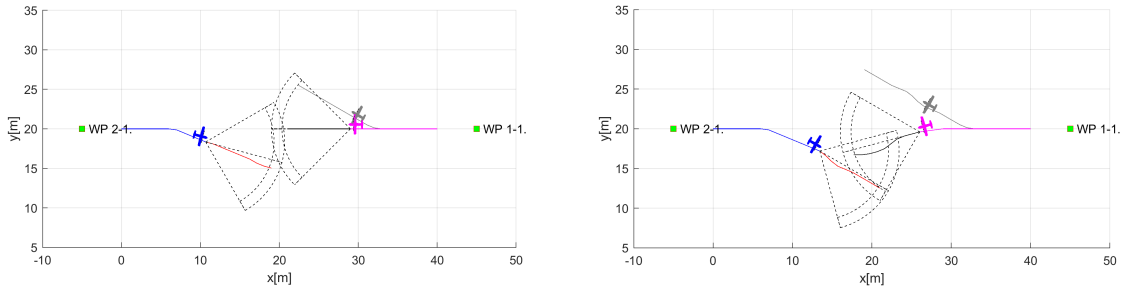
4. *Blind spot (*$\circlearrowleft UAS1$*)* (fig. 7.46d) following happens:

   a. *UAS 1* (blue) returns to *Navigation Mode*, because there is no *foreign object* in *avoidance grid* (black dashed line boundary).

   b. *UTM* receives the mode change and it starts enforcing *resolutions* for *collision case*, Adversary *UAS 2* is considered clear due *expected position* (grey plane/line) compliance with resolution.

   c. *Adversary UAS 2* (magenta) is on UAS 1 blind spot. The target UAS 1 (blue) is locked in UAS 2 navigation grid (black dashed line boundary).

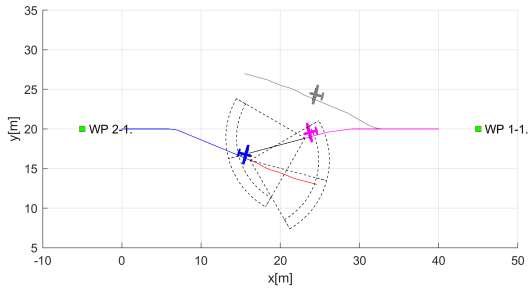
5. *Collision detail (*$UAS1 \leftrightarrow UAS2$*)*(fig. 7.46e) - Target *UAS 1* (blue) is hit by *Adversary UAS 2* (magenta) on left wing tip. Both UAS are going down. UTM will detect sudden loss of both UAS systems.



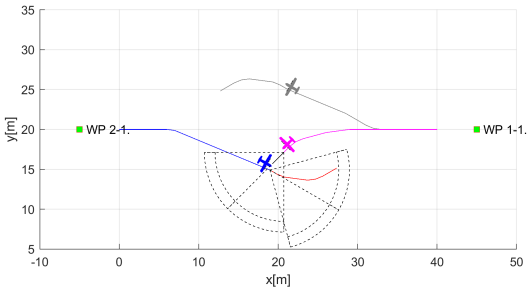

(a) Deviation detection (UAS 2).



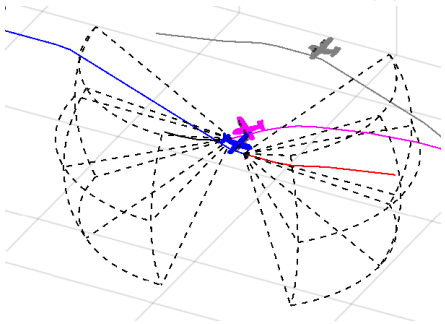

(b) Adversary attacking (UAS 2).




(c) Emergency avoidance (UAS 1).




(d) Blind spot (UAS 1).



(e) Collision detail.

Figure 7.46: Adversarial behaviour of *UAS 2* (magenta) to compliant *UAS 1* (blue)

**Performance Parameters Evaluation:** Performance parameters (y-axis) are tracked over *UTM time* (x-axis). The evolution of *performance* (fig. 7.47) is tracking following parameters:

1. *Expected crash distance* (gray line) - defined as (eq . 7.1) between UAS 1 (blue)and expected UAS 2 position (grey plane/line) over mission time $t \in [0, 22]$.

2. *Crash distance* (blue line) - defined as (eq . 7.1) between UAS 1 (blue) and real UAS 2 position (magenta plane/line) over mission time $t \in [0, 22]$.

3. *Safety margin* (yellow line) - constant value according to *collision case* (tab. 7.40) as value of *10 m*. The safety margin is considered as *soft constraint*.

4. *Body margin* (red line) - constant value according to (tab. 7.28) as value of *1.2 m*. The body margin is considered as *hard constraint*. The breaking of *body* margin means an effective *collision* UAS 1 and UAS 2.

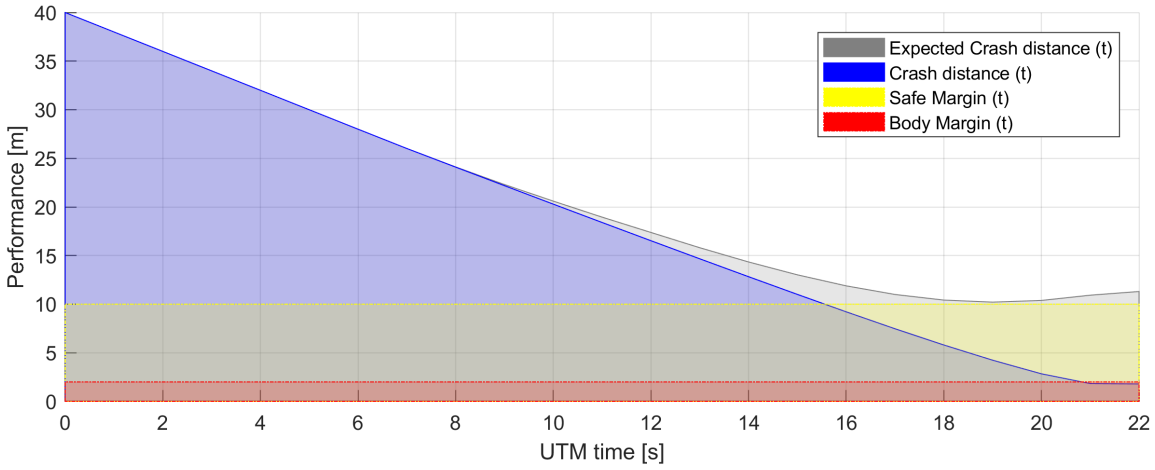



Figure 7.47: Expected/Real Distance to body/safety margin evolution for *adversarial behaviour* of UAS 2.

*Safety criteria* for both *body* and *safety margins* in case of *expected behaviour* are satisfied (eq. 7.10). This means that *UAS 1* fulfilled the *UTM directive* despite the fact that it entered *Emergency Avoidance Mode* (fig. 7.46c).

$$\begin{aligned} expectedDistanceToSafetyMargin(t) \geq 0, \qquad & \forall t \in [0, 22] \\ expectedDistanceToBodyMargin(t) \geq 0 \qquad & \forall t \in [0, 22] \end{aligned} \tag{7.10}$$

*Safety Margin* is broken at UTM time *15 s*, *body margin* is broken at UTM time *21 s*, the collision happens at UTM time *22 s*. This is summarized in *Distance Condition Breach* (eq. 7.11).

$$\begin{aligned} distanceToSafetyMargin(t) < 0, \qquad & \forall t \in [21, 22] \\ distanceToBodyMargin(t) < 0 \qquad & \forall t \in [15, 22] \end{aligned} \tag{7.11}$$

*Note.* An *adversary behaviour* needs to be addressed on:

1. *UAS Traffic Management Level* - our UTM implementation failed to detect *deviation* (fig. 7.46a) and *start of attack* (fig. 7.46b). UAS 2 (magenta) had clean intention from beginning and did not change pursuit even when *safety margin* was breached.

2. *Emergency Avoidance Level* - our *navigation loop implementation* does not consider the *ill-intentions*. The UAS 1 (blue) properly switched to *Emergency avoidance mode* (fig. 7.46c) after detection of UAS2 (magenta). UAS 2 (magenta) then used the blind spot to exploit UAS 1 vulnerability.

### 7.5.3 (R) Computation Footprint

The *computation footprint* is summarized in computation load (tab. 7.53). The *computation load* (eq. 7.5) was calculated for each *time-frame* in scenarios. There is summary of *minimal, maximal, average* and *median* values.

The *computational load* never exceed more than 55.95% in case of *emergency Head On* (eq. 7.6), which means that *every path* was calculated on time.

| Scenario | Computation load | | | |
|---|---|---|---|---|
| | min. | max. | avg. | med. |
| Building avoidance (fig. 7.4) | 2.20% | 27.40% | 12.11% | 13.20% |
| Slalom (fig. 7.8) | 12.20% | 30.50% | 21.42% | 21.50% |
| Maze (fig. 7.12) | 24.90% | 46.10% | 31.51% | 30.80% |
| Storm (fig. 7.16) | 2.60% | 26.90% | 11.57% | 13.90% |
| Emergency Converging (fig. 7.20) | 2.75% | 16.50% | 5.84% | 4.95% |
| Emergency Head On (fig. 7.24) | 3.90% | 55.95% | 13.19% | 6.90% |
| Emergency Multiple (fig. 7.28) | 5.90% | 52.35% | 12.77% | 8.56% |
| Rule-based Converging (fig. 7.32) | 3.60% | 13.50% | 7.32% | 5.97% |
| Rule-based Head on (fig. 7.36) | 4.65% | 41.60% | 13.64% | 9.30% |
| Rule-based Multiple (fig. 7.40) | 4.37% | 23.30% | 11.96% | 10.93% |
| Rule-based Overtake (fig. 7.45) | 3.85% | 13.40% | 7.62% | 6.70% |

Table 7.53: *Computation load statistics* for all test cases.

*Following observations can be made:*

1. *Building avoidance*, *Slalom*, and *Maze* scenarios - the computation load is increasing with the *amount of static obstacles*. The *average load* for *Emergency avoidance mode* in *clustered environment* is 31.51% (Maze).

2. *Storm scenario* - the overall *computation load* is very low due the *moving constraint implementation* (sec. 6.6.5).

3. *Emergency Converging/Head On/Multiple* scenarios - the *overall computation load* is quite high due the ineffective *body volume intersection* (sec. 6.6.3) implementation.

4. *Rule-based Converging/Head On/Multiple* scenarios - the *median computational* load is low, because of the linear *rule implementation* (sec. 6.9.2)

5. *Rule-based Overtake* - the *average computation load* is very low, because only *divergence/convergence* (rule. 6.14) waypoints are calculated and UAS stays in *navigation mode*.

## 7.6 Reduced Reach Sets Performance

*Constrained Expansion Method* (alg. 6.1) is creating *Reach Sets* from *Root Node* as a tree expansion using *Expansion Constraint function* (depending on type).

The *Reach set creation procedure* is creating following artifacts:

1. *Nodes* - tree Node containing necessary data for discrete Trajectory portion, notably *System State Evolution*, *buffer*, and, *Reachibility Rating*.
2. *Trajectories* - leaf *Node* containing *unique buffer* which is not *prefix* in others Node buffer.

The *Reach Set Computation Time* is depending strongly on *Movement Automaton* prediction complexity and Node count. The *Constrained Expansion Method* (alg. 6.1) is separating all nodes entering into $cell_{i,j,k}$ into two distinctive groups: *Candidates for expansion* and *Leftover Nodes*.

The *Leftover Nodes* are thrown away every expansion. The *Leftover Nodes* are not expanded in next *Wave-front* iteration, but they leave a notable *computation* and *memory* footprint.

*Note. Average Trajectory Smoothness Rate* (def. 31) is important only in *Navigation Mode*, this aspect has been covered over (sec. 6.4.4, 6.4.5, 6.4.6).

**Approach:** For exactly same conditions (*Testing Avoidance Grid*, *UAS initial state*, *Movement Automaton*) compare performance of *Reach Set Approximations* created by various methods for following parameters:

1. *Coverage Ratio* - defined in (def. 30) shows how versatile *Reach Set Approximation* is (up to 100% of complete reach set coverage).
2. *Node count* - count of Nodes in *Reach Set Approximation* counted like:
   a. full - all active nodes existing over computation time,
   b. pruned - active nodes for real time use.
3. *Count of Trajectories* - count of Trajectories (leaf Nodes) counted like:
   a. full - all active trajectories existing over computation time,
   b. pruned - active trajectories leading to coating cells of *Avoidance Grid*.

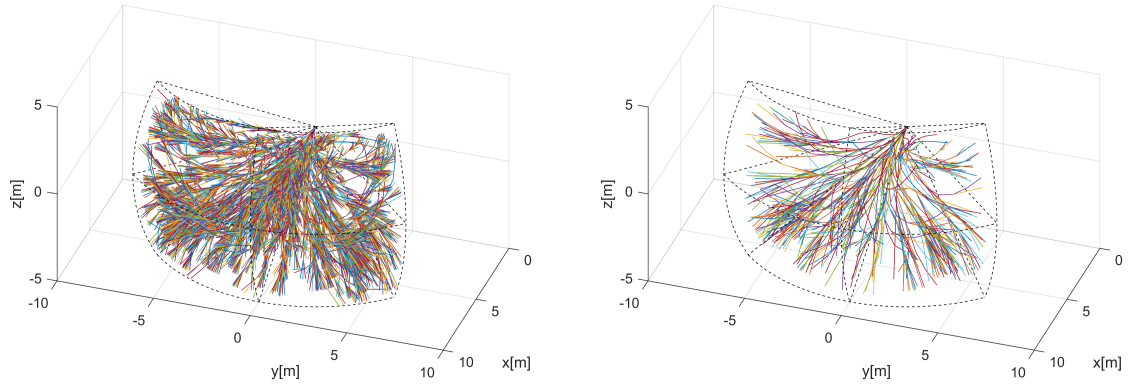
**Testing Avoidance Grid** with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45°, +45°]$, *Horizontal Cell Count 7*, *Vertical range* $[-30°, +30°]$, and *Vertical Cell Count 5*.

*Note.* The sizing of *Avoidance Grid* was chosen small scale, because property of *Coverage Ratio* can be calculated exactly up to some scale, after that it can be only assumed. Various sizes of *Avoidance Grid* was tested in [138].

The UAS is at *Back-side* of *Figure* (initial state is at all *Trajectory Origins*). The *black dashed line* marks *Avoidance Grid* space boundary. Each trajectory has its own color and ends at *Front-side* of *Avoidance Grid Boundary*.

**Chaotic Reach Set** (sec. 6.4.3) is used in *Emergency Avoidance Mode* for *Non-Controlled Airspace*. The *full* set of trajectories is given in (fig. 7.48a). The *Pruned* set of trajectories is given in (fig. 7.48b).

*Tuning parameters* were selected like follow: *Spread Ratio* is 15 (unique footprint trajectories in cell) and *trajectory footprint length* is 3 (last three unique passing cells).

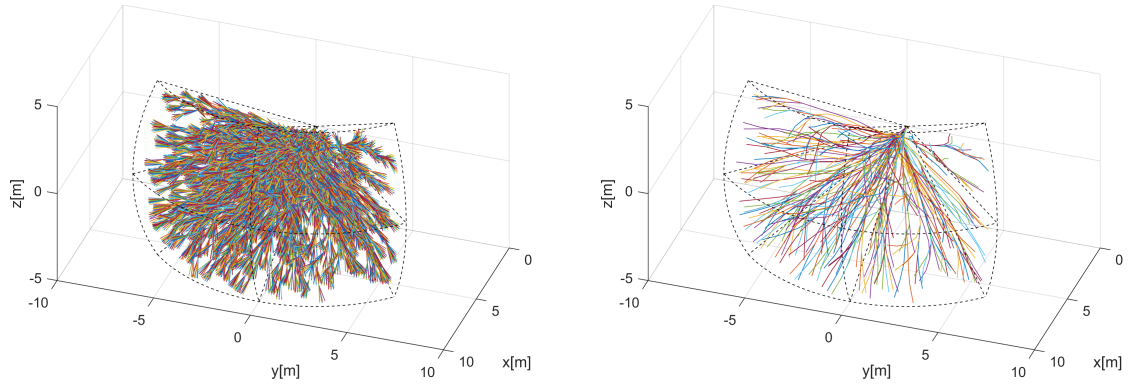



(a) Full. (b) Pruned.

Figure 7.48: Chaotic reach set computation example.

**Harmonic Reach Set** (sec. 6.4.4) is used in *Navigation Mode* for *Non Controlled Airspace*. The *full set* of trajectories is given in (fig. 7.49a). The *Pruned* set of trajectories is given in (fig. 7.49b).

*Tuning parameter* for *harmonic spread ratio* was set to 9 (which implies high coverage).

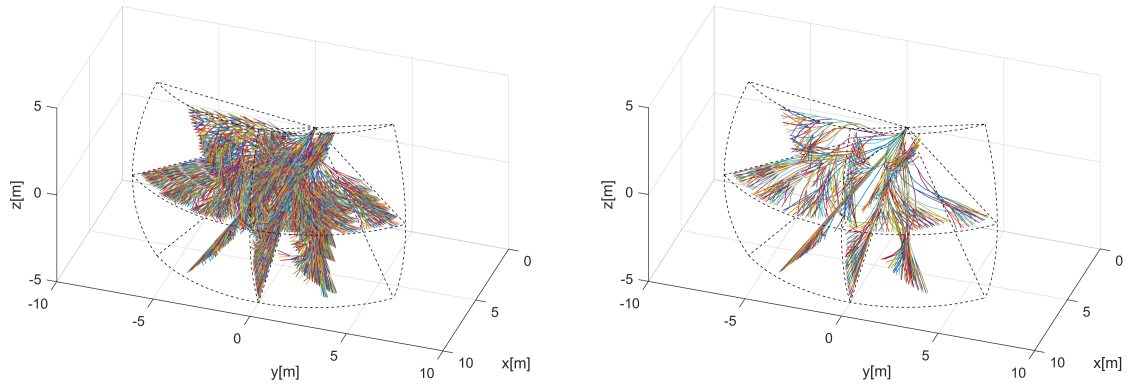



(a) Full. (b) Pruned.

Figure 7.49: Harmonic reach set computation example.

**Combined Reach Set** (sec. 6.4.5) is combination of *Chaotic Reach Set* (fig. 7.48) and *Harmonic Reach Set* (fig. 7.49). The *tuning parameters* are same for respective methods. It is used for both *Emergency Avoidance* and *Navigation*.

**ACAS-like Reach Set** (sec. 6.4.6) is used in *Navigation Mode* for *Controlled Airspace*. The separations used are: *Horizontal*, *Vertical*, *Slash*, and, *Backslash*, to give worst possible nodes and trajectories count. The *full* set of trajectories is given in (fig. 7.50a). The *Pruned* set of trajectories is given in (fig. 7.50b).




(a) Full. (b) Pruned.

Figure 7.50: ACAS-like reach set computation example.

**Computation Methods Performance Comparison** (tab. 7.54) gives overview of memory consumption and *Coverage Ratio*.

**Node count:** *Full Node Count* shows how much memory it takes to compute *Reach set* . *Pruned Node Count* shows how much *memory* is needed for storage.

*Note.* Total size of *full/pruned Reach Set* depends on Node implementation. The Object oriented prototype implementation in Matlab for example avoidance grid taken up to 1 megabyte of system memory. The effective implementation would take up to 100 kilobytes.

*Constrained expansion* (alg. 6.1) have different selection rate, depending on method. The survival rate directly reflects strictness of selection criteria. The rate of *node pruned* is summarized in (eq.7.12)

$$\begin{aligned} &\texttt{Nodes pruned} \\ &\quad chaotic \quad : 78.93\% \\ &\quad harmonic \quad : 18.50\% \\ &\quad ACAS-like \quad : 79.05\% \end{aligned} \tag{7.12}$$

The *interpretation or results* for each reach set estimation method is like follow:

1. *Chaotic* - the main exploration drive is *Coverage Rate*, the *Trajectory* segments are not usually smooth. For our *Movement Automaton* there is only one *Smooth Movement* : Straight. Other 8 are considered *Chaotic Movements*. Impact of this fact is significant, because 4/5 of nodes were pruned.

2. *Harmonic* - the main exploration drive is *Smoothness* of contained *Trajectories*. The *Trajectory segments* which are getting further away from *cell center* are not feasible. If *Smooth Movements* set size is considered, the Smooth/Chaotic ratio is 1/8 for our *Movement Automaton* implementation. The low node count was expected in this approach. Other Contributing factor is *Trajectory Footprint Length* for uniqueness selection, which is not a tuning parameter in this method, and it is set to most strict selection.

3. *ACAS-like* - the main drive is to create set consisting from *multiple 2D separation planes*. The expansion method applies full movement set on *candidate node*. The *Separation plane movement subset* is determining, which node will be selected for

further expansion. The size of separation plane subset to size of movement set rate is 1 : 3. There are four separation planes: horizontal, vertical, slash and backslash each containing full 2D plane reach set approximation which caused high node prune rate. Nodes used rate should get lower with increasing grid size.

**Trajectories count:** *Full trajectories count* shows how many *leaf nodes* were existing during calculation process without pruning. The difference between *full node count* and *full trajectories count* is count of inner tree nodes. *Pruned trajectories count* shows how many *leaf nodes* are used in run-time of *avoidance algorithm*. The difference between *pruned node count* and *pruned trajectories count* shows the count of inner nodes in active reach set.

The most of *waste leaf nodes* are removed during *layer pruning*: function *reachSet.purge- SameFootprint()* (alg. 6.1). The *Waste trajectories* or *unused leaf nodes count* have significant impact. Because *leaf nodes* are side product of *Node Expansion procedure* the amount of *pruned trajectories* is around 90 % regardless of the used method. The results are summarized in (eq. 7.13)

$$\begin{aligned} &\texttt{Trajectories pruned} \\ &\quad chaotic && : 91.24\% \\ &\quad harmonic && : 88.21\% \\ &\quad ACAS-like && : 89.43\% \end{aligned} \tag{7.13}$$

| Calculation method | Node count | | Trajectories | | Coverage ratio | Parameters |
|---|---|---|---|---|---|---|
| | full | pruned | full | pruned | | |
| chaotic | 6727 | 1417 | 4557 | 399 | 90% | spread:15 |
| harmonic | 1724 | 1405 | 1528 | 180 | 30% | spread:9 |
| combined | - | 2405 | - | 435 | 95% | CH spread:15<br>H spread:9<br>tree comb. |
| ACAS-like | 11294 | 2366 | 7437 | 786 | 74.95% | Separations:<br>H/V/S/BS<br>Coverage pruning:<br>disabled |

Table 7.54: *Reduced reach set* computation methods performance

**Coverage ratio:** (def. 30) is showing how much maneuvering versatility of *Reach Set*. *Full Reach Set Approximation* have coverage ratio of 100 %. It is possible to construct *Reference Reach Set* without constrained expansion method which contains all possible *trajectory footprints*. Following observations for *coverage ratio* can be made:

1. *Chaotic* reach set estimation method by design select *Nodes* which have probability of *trajectory footprint* diversification. The high coverage ratio was achieved at values around 90 %.

2. *Harmonic* reach set estimation method by design selects most smooth trajectories which cause low *trajectory footprint* diversity. Fairly high coverage ratio of 30 % has been achieved.

3. *Combined* reach set estimation method takes two reach set and combine their trajectory trees into single trajectory tree. It is given that *Coverage ratio* will achieve at least maximal coverage ratio of original reach sets. Harmonic reach set supplemented narrow smooth trajectories which were throw away previously. This increased overall *coverage ratio* to 95 %.

4. *ACAS-like* reach set estimation method contained 4 separation planes, which caused that it was similar to *Chaotic Reach Set Approximation* for given *Avoidance Grid*, in terms of performance. The coverage ratio For 2D plane was 100 %.

# Chapter 8

# (W) Conclusion and Future Work

*To be done here:*

- Conclusion of the work (This chapter can not be predicted for now)

## 8.1 (W) Framework Summary

## 8.2 (W) Other Methods Comparison

1. Vector field avoidance [153]
2. Potential field [174]

## 8.3 (R) Approach Reusability

**UTM Services:** The constrained *UTM functionality* is outlined in (sec. 6.8) including:

1. *Future UTM Communication Architecture* (fig. 6.26) as the authority over *airspace segment* (fig. 2.2)[3].
2. *Cooperative Conflict Resolution Under UTM Supervision* (fig. 6.27) designed as mild/feasible directives (commands) with *constant supervision*.
3. *Rules of the Air Enforcement* (sec. 6.8.4, 6.8.5, 6.8.6) including designs of *Position Notification* (sec. 6.8.7) and *Collision Case Structure/Calculation* (sec. 6.8.8).
4. *Divergence/Convergence Waypoints* concept is showcased in *Overtake Rule* (rule 6.14).
5. *Weather Avoidance* (sec. 6.8.9) is using similar concept to *Collision Case*: *Weather Case*. The information are provided by *Local Airspace Authority*.

**Emergency Avoidance Functionality:** The standard framework implementation (fig. 6.25) can handle the situations given in non-cooperative test cases (sec. 7.3). The list of threats is given by (tab. 4.2).

**Event Based Avoidance Functionality:** The standard framework implementation (fig. 6.25) with active $C2$ link and rules setup (fig. 6.33) can handle the situations given in cooperative test cases (7.4). The list of threats is given by (tab. 4.1). The *Avoidance Mode Concept* enables to switch between *Event Based Avoidance* (Navigation) and *Emergency Avoidance.*

*Note.* The emergency Avoidance Functionality is included in *Event Based Avoidance* (Navigation) mode. The prioritization of *threats* may differ (tab. 4.1).

**Reusability for More Complex Systems:** The framework (fig. 5.1) with implemented rule engine (fig. 6.32) can be used on *any system*, with appropriate *Movement automaton* (sec. 6.2.3) enabling *wave-front* propagation (alg. 6.1) for reach set estimation. Following artifacts needs to be delivered for concept reuse:

1. The *Movement Automaton* is used to generate *thick series of waypoints* which guarantees desired degree of safety.
2. The *complex UAS system* is following the *reference trajectory* (sec. 6.2.4).
3. The *Sensor Fusion* (sec. 4.1.8) implementation including classification to *Free*, *Occupied*, *Restricted* space type.
4. The *sensor field* supporting detection of threats. There should be at least one sensor with capability of feeding *Avoidance Grid*. Our implementation was based on LiDAR/ADS-B feeds.
5. The *Information Sources* supporting the online/offline threat processing. This one is completely optional.

*Note. On UTM integration:* The future UTM system will not giving the extreme commands, the directives are more like constraints, therefore our system can provide the guidance and constraint evaluation

*Note. On Safety Margin:* The disparity between real flown trajectory (nonlinear dynamics) and planned trajectory (Movement Automaton) needs to be accounted into *Safety Margin.*

**Reach Set Approximations:** The *wave-front* approach (alg. 6.1) can be used with *Constrained expansion function* (sec. 6.4.2) to create own *Reach set Approximation Method*. Existing reach set approximation methods are always following a different goal, they can be reused for other tasks (perf. 7.6):

1. *Chaotic* (def. 6.4.3) - high space coverage, ideal for unpredictable and complex avoidance maneuvers.
2. *Harmonic* (def. 6.4.4) - smooth trajectories, medium space coverage, ideal for navigation maneuvers.
3. *Combined* (def. 6.4.5) - combination of the *harmonic* and

   *chaotic* approximations, the cost function defines preferred trajectories. The procedure is reusable for any reach set approximation types ($2^+$) combination.
4. *ACAS-X Like* (def. 6.4.6) - following *TCAS/ACAS separation modes*, can be used as alternative for *controlled avoidance* and *navigation.*

## 8.4 (W) Lessons learned

What can be done differently

- The discretization - euclidian grid vs polar grid
- Intruder modeling ideas, the linear intersection without body volume
- The probabilistic/vs rating approach

## 8.5 (W) Future Work

1. Adversarial avoidance
2. Real system implementation

# Bibliography


[1] Kimon P Valavanis and George J Vachtsevanos. Uav sense, detect and avoid: Introduction. In *Handbook of Unmanned Aerial Vehicles*, pages 1813–1816. Springer, 2015.

[2] United States. Federal Aviation Administration. *Pilots' Role in Collision Avoidance*. Number 48 in 1. US Department of Transportation, Federal Aviation Administration, 1983.

[3] Ingrid Gerdes, Annette Temme, and Michael Schultz. Dynamic airspace sectorization using controller task load. *Sixth SESAR Innovation Days*, 2016.

[4] Edward Balaban, Indranil Roychoudhury, Lilly Spirkovska, Shankar Sankararaman, Chetan S Kulkarni, and Matthew Daigle. Dynamic routing of aircraft in presence of adverse weather using a pomdp framework. In *17$^{th}$ AIAA Aviation Technology, Integration, and Operations Conference*, page 3429, 2017.

[5] Glenn K Rutledge, Jordan Alpert, and Wesley Ebisuzaki. Nomads: A climate and weather model archive at the national oceanic and atmospheric administration. *Bulletin of the American Meteorological Society*, 87(3):327–341, 2006.

[6] Theodore C Yapo, Charles V Stewart, and Richard J Radke. A probabilistic representation of lidar range data for efficient 3d object detection. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.

[7] Alojz Gomola, João Borges de Sousa, Fernando Lobo Pereira, and Pavel Klang. Obstacle avoidance framework based on reach sets. In *Iberian Robotics conference*, pages 768–779. Springer, 2017.

[8] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001.

[9] Oskar Ljungqvist, Niclas Evestedt, Marcello Cirillo, Daniel Axehill, and Olov Holmer. Lattice-based motion planning for a general 2-trailer system. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 819–824. IEEE, 2017.

[10] José Queirós Pinto, Paulo Sousa Dias, Rui Gonçalves, Gil Manuel Gonçalves, João Tasso de Figueiredo Borges Sousa, Fernando Lobo Pereira, et al. Neptus a framework to support a mission life cycle. In *Proceedings of the 7th Conference on Manoeuvring and Control of Marine Craft*, 2006.

[11] Maria Cerna. Usage of maps obtained by lidar in uav navigation. Master thesis, Institute of Automotive Mechatronics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovicova 3, Bratislava. Slovak Republic, jun 2018.

[12] Alessandro Gardi, Roberto Sabatini, Subramanian Ramasamy, Matthew Marino, et al. Automated atm system for 4-dimensional trajectory based operations. In *AIAC16: 16th Australian International Aerospace Congress*, page 190. Engineers Australia, 2015.

[13] Michael Huerta. Integration of civil unmanned aircraft systems (uas) in the national airspace system (nas) roadmap. *Federal Aviation Administration, Retrieved Dec*, 19:2013, 2013.

[14] Artur Zolich, David Palma, Kimmo Kansanen, Kay Fjørtoft, João Sousa, Karl H. Johansson, Yuming Jiang, Hefeng Dong, and Tor Johansen. Survey on communication and networks for autonomous marine systems. *Journal of Intelligent & Robotic Systems*, page tba, 04 2018.

[15] Francesco Nex and Fabio Remondino. Uav for 3d mapping applications: a review. *Applied geomatics*, 6(1):1–15, 2014.

[16] William Kress Bodin, Jesse Redman, and Derral Charles Thorson. Navigating a uav with obstacle avoidance algorithms, June 5 2007. US Patent 7,228,232.

[17] Jonathan P How, BRETT BEHIHKE, Adrian Frank, Daniel Dale, and John Vian. Real-time indoor autonomous vehicle test environment. *IEEE control systems*, 28(2):51–64, 2008.

[18] Anouck R Girard, Adam S Howell, and J Karl Hedrick. Border patrol and surveillance missions using multiple unmanned air vehicles. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 620–625. IEEE, 2004.

[19] Fabio AA Andrade, Rune Storvold, and Tor Arne Johansen. Autonomous uav surveillance of a ship's path with mpc for maritime situational awareness. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 633–639. IEEE, 2017.

[20] Kristian Klausen, Thor I Fossen, and Tor Arne Johansen. Nonlinear control with swing damping of a multirotor uav with suspended load. *Journal of Intelligent & Robotic Systems*, 88(2-4):379–394, 2017.

[21] Thomas W Kennedy Jr and Donald F Fenstermaker. Resolution advisory display instrument for tcas guidance, January 17 1995. US Patent 5,382,954.

[22] Mike Marston and Gabe Baca. Acas-xu initial self-separation flight tests, 2015.

[23] Jon A Blaskovich and Stephen G McCauley. Declutter of graphical tcas targets to improve situational awareness, December 11 2007. US Patent 7,307,578.

[24] Roberto Sabatini, Alessandro Gardi, and Mark A Richardson. Lidar obstacle warning and avoidance system for unmanned aircraft. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(4):702–713, 2014.

[25] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Jing Liu. Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55:344–358, 2016.

[26] Subramanian Ramasamy, Roberto Sabatini, and Alessandro Gardi. Avionics sensor fusion for small size unmanned aircraft sense-and-avoid. In *Metrology for Aerospace (MetroAeroSpace), 2014 IEEE*, pages 271–276. IEEE, 2014.

[27] John D Lee, Daniel V McGehee, Timothy L Brown, and Michelle L Reyes. Collision warning timing, driver distraction, and driver response to imminent rear-end collisions in a high-fidelity driving simulator. *Human factors*, 44(2):314–334, 2002.

[28] Ronald Miller and Qingfeng Huang. An adaptive peer-to-peer collision warning system. In *Vehicular technology conference, 2002. VTC Spring 2002. IEEE 55th*, volume 1, pages 317–321. IEEE, 2002.

[29] Lisa C Thomas, Christopher D Wickens, and IL Savoy. Effects of cdti display dimensionality and conflict geometry on conflict resolution performance. In *Proceedings of the 13th International Symposium on Aviation Psychology*. Citeseer, 2005.

[30] David H Williams. Self-separation in terminal areas using cdti. In *Proceedings of the Human Factors Society Annual Meeting*, volume 27, pages 772–776. Sage Publications Sage CA: Los Angeles, CA, 1983.

[31] Alessandro Gardi, Roberto Sabatini, Subramanian Ramasamy, and Trevor Kistan. Real-time trajectory optimisation models for next generation air traffic management systems. In *Applied Mechanics and Materials*, volume 629, pages 327–332. Trans Tech Publ, 2014.

[32] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[33] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*, pages 359–370. Springer, 1991.

[34] Mark J Koetse and Piet Rietveld. The impact of climate change and weather on transport: An overview of empirical findings. *Transportation Research Part D: Transport and Environment*, 14(3):205–221, 2009.

[35] Hiroshi Yamashita, Volker Grewe, Patrick Jöckel, Florian Linke, M Schaefer, and D Sasaki. Climate impact assessment of routing strategies: Interactive air traffic in a climate model. *Climate Proceedings*, 2015.

[36] Travis M Smith, Valliappa Lakshmanan, Gregory J Stumpf, Kiel L Ortega, Kurt Hondl, Karen Cooper, Kristin M Calhoun, Darrel M Kingfield, Kevin L Manross, Robert Toomey, et al. Multi-radar multi-sensor (mrms) severe weather and aviation products: Initial operating capabilities. *Bulletin of the American Meteorological Society*, 97(9):1617–1630, 2016.

[37] Gregory Thompson, Marcia K Politovich, and Roy M Rasmussen. A numerical weather model's ability to predict characteristics of aircraft icing environments. *Weather and Forecasting*, 32(1):207–221, 2017.

[38] Thomas P Spriesterbach, Kelly A Bruns, Lauren I Baron, and Jason E Sohlke. Unmanned aircraft system airspace integration in the national airspace using a ground-based sense and avoid system. *Johns Hopkins APL Technical Digest*, 32(3):572–583, 2013.

[39] Adrian Muraru. A critical analysis of sense and avoid technologies for modern uavs. In *Mechanical, Industrial, and Manufacturing Engineering–Proceedings of 2011 International Conference on Mechanical, Industrial, and Manufacturing Engineering (MIME 2011)*, 2011.

[40] John Lai, Jason J Ford, Luis Mejias, Peter O'Shea, and Rod Walker. See and avoid using onboard computer vision. *Sense and Avoid in UAS Research and Applications, Plamen Angelov (ed.), John Wiley and Sons, West Sussex, UK*, 2012.

[41] Mykel J Kochenderfer, Leo P Espindle, J Daniel Griffith, and James K Kuchar. Encounter modeling for sense and avoid development. In *2008 Integrated Communications, Navigation and Surveillance Conference*, pages 1–10. IEEE, 2008.

[42] JARUS regulations. `http://jarus-rpas.org/regulations`. Accessed: 2018-10-28.

[43] Edward A Lee. *Structure and interpretation of signals and systems*. Lee & Seshia, 2011.

[44] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, 1987.

[45] Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

[46] Nikolai Nikolaevich Krasovskij, Andrei Izmailovich Subbotin, and Samuel Kotz. *Game-theoretical control problems*. Springer-Verlag New York, Inc., 1987.

[47] NN Krasovskii and AI Subbotin. Game-theoretical control problems. translated from the russian by samuel kotz, 1988.

[48] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool kronos. In *Hybrid Systems III*, pages 208–219. Springer, 1996.

[49] Thomas A Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and computation*, 111(2):193–244, 1994.

[50] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: the next generation. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 56–65. IEEE, 1995.

[51] G Leitmann. Optimality and reachability via feedback controls. *Dynamic systems and mycrophysics*, pages 119–141, 1982.

[52] LS Pontryagin, VG Boltyanskii, and RV Gamkrelidze. Ef mischenko the mathematical theory of optimal processes (english translation by k. n. trirogoff). *Interscience, New York*, 1962.

[53] Igor Vladimirovich Girsanov. *Lectures on mathematical theory of extremum problems*, volume 67. Springer Science & Business Media, 2012.

[54] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[55] Ondřej Šantin and Vladimir Havlena. Combined partial conjugate gradient and gradient projection solver for mpc. In *Control Applications (CCA), 2011 IEEE International Conference on*, pages 1270–1275. IEEE, 2011.

[56] Frangois G Pin and Hubert A Vasseur. Autonomous trajectory generation for mobile robots with non-holonomic and steering angle constraints. Technical report, Oak Ridge National Lab., 1990.

[57] Ralph G Andrzejak, G Widman, K Lehnertz, C Rieke, P David, and CE Elger. The epileptic process as nonlinear deterministic dynamics in a stochastic environment: an evaluation on mesial temporal lobe epilepsy. *Epilepsy research*, 44(2-3):129–140, 2001.

[58] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

[59] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Trajectory tracking control design for autonomous helicopters using a backstepping algorithm. In *American Control Conference, 2000. Proceedings of the 2000*, volume 6, pages 4102–4107. IEEE, 2000.

[60] Mircea Lazar. Model predictive control of hybrid systems: Stability and robustness, 2006.

[61] Francesco Borrelli, Alberto Bemporad, Michael Fodor, and Davor Hrovat. An mpc/hybrid system approach to traction control. *IEEE Transactions on Control Systems Technology*, 14(3):541–552, 2006.

[62] Thor I Fossen. Mathematical models for control of aircraft and satellites. *Department of Engineering Cybernetics Norwegian University of Science and Technology*, 2011.

[63] Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.

[64] S Martin, J Bange, and F Beyrich. Meteorological profiling of the lower troposphere using the research uav" m 2 av carolo". *Atmospheric Measurement Techniques*, 4(4):705–716, 2011.

[65] Qi Chen. Airborne lidar data processing and information extraction. *Photogrammetric engineering and remote sensing*, 73(2):109, 2007.

[66] Leonhard Euler. Formulae generales pro translatione quacunque corporum rigidorum. *Novi Acad. Sci. Petrop*, 20:189–207, 1775.

[67] Hanspeter Schaub and John L Junkins. *Analytical mechanics of space systems*. Aiaa, 2003.

[68] Manuel Kramer and Douglas J Dapprich. Gyro stabilized inertial reference system with gimbal lock prevention means, October 4 1977. US Patent 4,052,654.

[69] Alexander B. Kurzhanski and Pravin Varaiya. Dynamic optimization for reachability problems. *Journal of Optimization Theory and Applications*, 108(2):227–251, 2001.

[70] Pravin Varaiya. Reach set computation using optimal control. In *Verification of Digital and Hybrid Systems*, pages 323–331. Springer, 2000.

[71] José Pinto, Paulo S Dias, Ricardo Martins, Joao Fortuna, Eduardo Marques, and Joao Sousa. The lsts toolchain for networked vehicle systems. In *OCEANS-Bergen, 2013 MTS/IEEE*, pages 1–9. IEEE, 2013.

[72] José Pinto, Pedro Calado, José Braga, Paulo Dias, Ricardo Martins, Eduardo Marques, and JB Sousa. Implementation of a control architecture for networked vehicle systems. *IFAC Proceedings Volumes*, 45(5):100–105, 2012.

[73] George J Pappas, Gerardo Lafferriere, and Shankar Sastry. Hierarchically consistent control systems. *IEEE transactions on automatic control*, 45(6):1144–1160, 2000.

[74] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[75] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990.

[76] Philippe Souères and J-D Boissonnat. Optimal trajectories for nonholonomic mobile robots. In *Robot motion planning and control*, pages 93–170. Springer, 1998.

[77] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[78] Héctor J Sussmann. Lie brackets, real analyticity and geometric control. *Differential geometric control theory*, 27:1–116, 1983.

[79] Fredrik Gustafsson. *Statistical sensor fusion*. Studentlitteratur,, 2010.

[80] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Trevor Kistan. Next generation flight management system for real-time trajectory based operations. *Applied Mechanics and Materials*, 629:344–349, 2014.

[81] Yuriy Chynchenko, Tatyana Shmelova, and Oksana Chynchenko. Remotely piloted aircraft systems operations under uncertainty conditions. *Proceedings of the National Aviation University*, 1(1):18–22, 2016.

[82] T Shmelova, D Bondarev, and Y Znakovska. Modeling of the decision making by uav's operator in emergency situations. In *Methods and Systems of Navigation and Motion Control (MSNMC), 2016 4th International Conference on*, volume 1, pages 31–34. IEEE, 2016.

[83] Volodymyr Kharchenko, Tatyana Shmelova, Yevgeniya Znakovska, Dmitriy Bondarev, and Andriy Stratiy. Modelling of decision making of unmanned aerial vehicle's operator in emergency situations. *Proceedings of the National aviation university*, 1(1):20–28, 2017.

[84] Kwang-Yeon Kim, Jung-Woo Park, and Min-Jea Tahk. Uav collision avoidance using probabilistic method in 3-d. In *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, pages 826–829. IEEE, 2007.

[85] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE transactions on robotics and automation*, 18(5):662–669, 2002.

[86] Barbara Pfeiffer, Rajan Batta, Kathrin Klamroth, and Rakesh Nagi. Path planning for uavs in the presence of threat zones using probabilistic modeling. *IEEE Trans. Autom. Control*, 43:278–283, 2005.

[87] Mangal Kothari and Ian Postlethwaite. A probabilistically robust path planning algorithm for uavs using rapidly-exploring random trees. *Journal of Intelligent & Robotic Systems*, pages 1–23, 2013.

[88] Lars Blackmore, Hui Li, and Brian Williams. A probabilistic approach to optimal robust path planning with obstacles. In *American Control Conference, 2006*, pages 7–pp. IEEE, 2006.

[89] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[90] Ira M Gessel. A probabilistic method for lattice path enumeration. *Journal of statistical planning and inference*, 14(1):49–58, 1986.

[91] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J Kay. A comparative analysis of methods for pruning decision trees. *IEEE transactions on pattern analysis and machine intelligence*, 19(5):476–491, 1997.

[92] Kaoru Hirota. Concepts of probabilistic sets. *Fuzzy sets and systems*, 5(1):31–46, 1981.

[93] Jacco M Hoekstra, Ronald NHW van Gent, and Rob CJ Ruigrok. Designing for safety: the 'free flight'air traffic management concept. *Reliability Engineering & System Safety*, 75(2):215–232, 2002.

[94] Alessandro Gardi, Roberto Sabatini, and Trevor Kistan. Multi-objective 4d trajectory optimization for integrated avionics and air traffic management systems. *IEEE Transactions on Aerospace and Electronic Systems*, 2018.

[95] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.

[96] Yixiang Lim, Alessandro Gardi, and Roberto Sabatini. Energy efficient 4d trajectories for terminal descent operations. In *International Symposium on Sustainable Aviation*, 2018.

[97] Swati Mishra and Pankaj Bande. Maze solving algorithms for micro mouse. In *Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on*, pages 86–93. IEEE, 2008.

[98] Ibrahim Elshamarka and Abu Bakar Sayuti Saman. Design and implementation of a robot for maze-solving using flood-fill algorithm. *International Journal of Computer Applications*, 56(5), 2012.

[99] Quentin Chatelais, Horatiu Vultur, and Emmanouil Kanellis. Maze solving by an autonomous robot. *Aalborg University*, 2014.

[100] Nathan Richards, Manu Sharma, and David Ward. A hybrid a-star automaton approach to on-line path planning with obstacle avoidance. In *AIAA 1st Intelligent Systems Technical Conference*, page 6229, 2004.

[101] Zhuoning Dong, Zongji Chen, Rui Zhou, and Rulin Zhang. A hybrid approach of virtual force and a* search algorithm for uav path re-planning. In *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*, pages 1140–1145. IEEE, 2011.

[102] Allison Ryan and J Karl Hedrick. A mode-switching path planner for uav-assisted search and rescue. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 1471–1476. IEEE, 2005.

[103] Oskar Ljungqvist, Daniel Axehill, and Anders Helmersson. Path following control for a reversing general 2-trailer system. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 2455–2461. IEEE, 2016.

[104] Niclas Evestedt, Oskar Ljungqvist, and Daniel Axehill. Path tracking and stabilization for a reversing general 2-trailer configuration using a cascaded control approach. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 1156–1161. IEEE, 2016.

[105] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[106] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. William "red" whittaker. *Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, Dave Ferguson, Autonomous driving in urban environments: Boss and the Urban Challenge, Journal of Field Robotics*, 25(8):425–466, 2008.

[107] Marcello Cirillo. From videogames to autonomous trucks: A new algorithm for lattice-based motion planning. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 148–153. IEEE, 2017.

[108] Claire J Tomlin, John Lygeros, and S Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, 2000.

[109] Hong Chen, Carsten W Scherer, and F Allgower. A game theoretic approach to nonlinear robust receding horizon control of constrained systems. In *American Control Conference, 1997. Proceedings of the 1997*, volume 5, pages 3073–3077. IEEE, 1997.

[110] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[111] Yasutake Takahashi, Minoru Asada, and Koh Hosoda. Reasonable performance in less learning time by real robot based on incremental state space segmentation. In *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 3, pages 1518–1524. IEEE, 1996.

[112] Isaac Kaminer, Antonio Pascoal, Eric Hallberg, and Carlos Silvestre. Trajectory tracking for autonomous vehicles: An integrated approach to guidance and control. *Journal of Guidance, Control, and Dynamics*, 21(1):29–38, 1998.

[113] Marina H Murillo, Alejandro C Limache, Pablo S Rojas Fredini, and Leonardo L Giovanini. Generalized nonlinear optimal predictive control using iterative state-space trajectories: Applications to autonomous flight of uavs. *International Journal of Control, Automation and Systems*, 13(2):361–370, 2015.

[114] Diego Merani, Alessandro Berni, John Potter, and Ricardo Martins. An underwater convergence layer for disruption tolerant networking. In *Internet Communications (BCFIC Riga), 2011 Baltic Congress on Future*, pages 103–108. IEEE, 2011.

[115] Kanna Rajan, Frédéric Py, and Javier Barreiro. Towards deliberative control in marine robotics. In *Marine Robot Autonomy*, pages 91–175. Springer, 2013.

[116] Paulo Sousa Dias, Rui MF Gomes, and José Pinto. Mission planning and specification in the neptus framework. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3220–3225. IEEE, 2006.

[117] Paulo Sousa Dias, Sergio Loureiro Fraga, Rui MF Gomes, Gil M Goncalves, Fernando Lobo Pereira, Jose Pinto, and Joao Borges Sousa. Neptus-a framework to support multiple vehicle operation. In *Oceans 2005-Europe*, volume 2, pages 963–968. IEEE, 2005.

[118] Ricardo Martins, Paulo Sousa Dias, Eduardo RB Marques, José Pinto, Joao B Sousa, and Fernando L Pereira. Imc: A communication protocol for networked vehicles and sensors. In *Oceans 2009-Europe*, pages 1–6. IEEE, 2009.

[119] Lorenzo Marconi, Roberto Naldi, and Luca Gentili. A control framework for robust practical tracking of hybrid automata. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 661–666. IEEE, 2009.

[120] Virtual arena mpc framework for advanced uav-uas simulations. `https://github.com/andreaalessandretti/VirtualArena`. Accessed: 2016-05-30.

[121] Andrea Alessandretti. Notes on trajectory-tracking and path-following controllers for constrained underactuated vehicles using model predictive control.

[122] Brian L Stevens, Frank L Lewis, and Eric N Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. John Wiley & Sons, 2015.

[123] Karthik Balakrishnan, Joe Polastre, Jessie Mooberry, Richard Golding, and Peter Sachs. The roadmap for the safe integration of autonomous aircraft. Blueprint for the sky - Airbus, www.utmblueprint.com, sep 2018.

[124] Andrew Hately. Concept of operations for u-space. Exploratory research call, EUROCONTROL, sep 2018.

[125] AAMI Standard. Recommended practices. *Operation of Aircraft, Annex*, 6, 1986.

[126] Maria Prandini and Jianghai Hu. Application of reachability analysis for stochastic hybrid systems to aircraft conflict prediction. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4036–4041. IEEE, 2008.

[127] Florian Homm, Nico Kaempchen, Jeff Ota, and Darius Burschka. Efficient occupancy grid computation on the gpu with lidar and radar for road boundary detection. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 1006–1013. IEEE, 2010.

[128] Sandeep Gupta, Holger Weinacker, and Barbara Koch. Comparative analysis of clustering-based approaches for 3-d single tree detection using airborne fullwave lidar data. *Remote Sensing*, 2(4):968–989, 2010.

[129] Osmar R Zaïane and Chi-Hoon Lee. Clustering spatial data when facing physical constraints. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 737–740. IEEE, 2002.

[130] Alex A Kurzhanskiy and Pravin Varaiya. Ellipsoidal toolbox (et). In *Decision and Control, 2006 45th IEEE Conference on*, pages 1498–1503. IEEE, 2006.

[131] John Birmingham and Peter Kent. Tree-searching and tree-pruning techniques. In *Computer chess compendium*, pages 123–128. Springer, 1988.

[132] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. In *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*, pages 65–100. Springer, 2009.

[133] Catherine Plaisant, Jesse Grosjean, and Benjamin B Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 57–64. IEEE, 2002.

[134] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[135] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

[136] Jay Shively. Uas integration in the nas: Detect and avoid. 2018.

[137] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. Policy compression for aircraft collision avoidance systems. In *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*, pages 1–10. IEEE, 2016.

[138] Alojz Gomola, Pavel Klang, and Jan Ludvik. Probabilistic approach in data fusion for obstacle avoidance framework based on reach sets. In *Internal publication collection*, pages 1–93. Honeywell, 2017.

[139] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. The MIT press, 2009.

[140] Jack Ritter. An efficient bounding sphere. *Graphics gems*, 1:301–303, 1990.

[141] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[142] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151–162. IEEE, 1975.

[143] Jon Louis Bentley, Bruce W Weide, and Andrew C Yao. Optimal expected-time algorithms for closest point problems. *ACM Transactions on Mathematical Software (TOMS)*, 6(4):563–580, 1980.

[144] Jon Louis Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, (9):643–647, 1979.

[145] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *17th annual symposium on foundations of computer science*, pages 208–215. IEEE, 1976.

[146] Duncan Mclaren Young Sommerville. *Analytical geometry of three dimensions*. Cambridge University Press, 2016.

[147] ICAO. 4444: Procedures for air navigation services. Technical report, ICAO, 2018.

[148] Roberto Sabatini, Subramanian Ramasamy, Alessandro Gardi, and Leopoldo Rodriguez Salazar. Low-cost sensors data fusion for small size unmanned aerial vehicles navigation and guidance. *International Journal of Unmanned Systems Engineering.*, 1(3):16, 2013.

[149] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998.

[150] Roberto Sabatini, Celia Bartel, Anish Kaharkar, Tesheen Shaid, and Subramanian Ramasamy. Navigation and guidance system architectures for small unmanned aircraft applications. *International Journal of Mechanical, Industrial Science and Engineering*, 8(4):733–752, 2014.

[151] Roberto Sabatini, Alessandro Gardi, and M Richardson. Lidar obstacle warning and avoidance system for unmanned aircraft. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(4):718–729, 2014.

[152] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. A microeconomic view of data mining. *Data mining and knowledge discovery*, 2(4):311–324, 1998.

[153] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[154] Nico Zimmer, Jens Schiefele, Keyvan Bayram, Theo Hankers, Sebastian Frank, and Thomas Feuerle. Rule-based notam & weather notification. In *Integrated Communications, Navigation and Surveilance Conference (ICNS), 2011*, pages O1–1. IEEE, 2011.

[155] Thomas Prevot, Joseph Rios, Parimal Kopardekar, John E Robinson III, Marcus Johnson, and Jaewoo Jung. Uas traffic management (utm) concept of operations to safely enable low altitude flight operations. In *16th AIAA Aviation Technology, Integration, and Operations Conference*, page 3292, 2016.

[156] Nico Zimmer and Keyvan Bayram. Selective weather notification, March 18 2014. US Patent 8,674,850.

[157] Subramanian Ramasamy, Roberto Sabatini, and Alessandro Gardi. Towards a unified approach to cooperative and non-cooperative rpas detect-and-avoid. In *Fourth Australasian Unmanned Systems Conference*, 2014.

[158] Subramanian Ramasamy, Roberto Sabatini, A Gardi, and Yifang Liu. Novel flight management system for real-time 4-dimensional trajectory based operations. In *proceedings of AIAA Guidance, Navigation, and Control Conference*, 2013.

[159] Branka Subotic, Arnab Majumdar, and Washington Y Ochieng. Recovery from equipment failures in air traffic control (atc): The findings from an international survey of controllers. *Air Traffic Control Quarterly*, 15(2):157–181, 2007.

[160] ICAO. Annex 2 (rules of the air). Technical report, ICAO, 2018.

[161] ICAO. Annex 11 (air traffic services). Technical report, ICAO, 2018.

[162] Alexandre Bayen, Pascal Grieder, George Meyer, and Claire J Tomlin. Langrangian delay predictive model for sector-based air traffic flow. *Journal of guidance, control, and dynamics*, 28(5):1015–1026, 2005.

[163] Parimal Kopardekar and Sherri Magyarits. Dynamic density: measuring and predicting sector complexity [atc]. In *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, volume 1, pages 2C4–2C4. IEEE, 2002.

[164] MP Helme, K Lindsay, SV Massimini, and G Booth. Optimization of traffic flow to minimize delay in the national airspace system. In *Control Applications, 1992., First IEEE Conference on*, pages 435–437. IEEE, 1992.

[165] Confesor Santiago and Eric R Mueller. Pilot evaluation of a uas detect-and-avoid system's effectiveness in remaining well clear. In *Eleventh UAS/Europe Air Traffic Management Research and Development Seminar (ATM2015)*, 2015.

[166] Karl Bilimoria and Hilda Lee. Analysis of aircraft clusters to measure sector-independent airspace congestion. In *AIAA 5th ATIO and16th Lighter-Than-Air Sys Tech. and Balloon Systems Conferences*, page 7455, 2005.

[167] CR Brinton and S Pledgie. Airspace partitioning using flight clustering and computational geometry. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 3–B. IEEE, 2008.

[168] M Ebrahim Fouladvand, Zeinab Sadjadi, and M Reza Shaebani. Characteristics of vehicular traffic flow at a roundabout. *Physical Review E*, 70(4):046132, 2004.

[169] Raffaele Mauro and Marco Cattani. Potential accident rate of turbo-roundabouts. In *4th International Symposium on Highway Geometric DesignPolytechnic University of ValenciaTransportation Research Board*, 2010.

[170] Michael R Benjamin, Joseph A Curcio, John J Leonard, and Paul M Newman. Navigation of unmanned marine vehicles in accordance with the rules of the road. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3581–3587. IEEE, 2006.

[171] Ernest Friedman Hill. *Jess in action: Java rule-based systems*. Manning Publications Co., 2003.

[172] Georg S Seyboth, Dimos V Dimarogonas, and Karl H Johansson. Event-based broadcasting for multi-agent average consensus. *Automatica*, 49(1):245–252, 2013.

[173] Hans Samelson, Robert M Thrall, and Oscar Wesler. A partition theorem for euclidean n-space. *Proceedings of the American Mathematical Society*, 9(5):805–807, 1958.

[174] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.