Chapter 6

Approach

The levels of *Avoidance* depending on *reaction time* are summarized in (fig. 6.1).

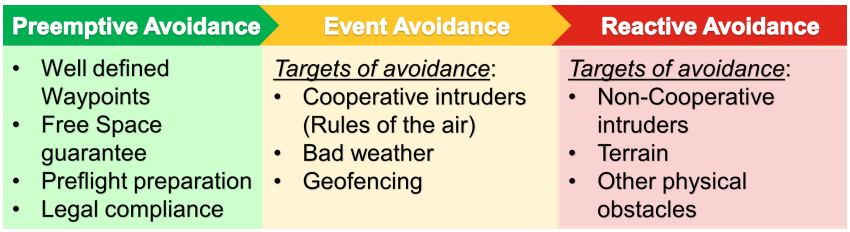


Figure 6.1: Avoidance levels based on reaction time.

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 before the flight. The risk mitigation is tedious and its done only when necessary. Even the best *preemptive* avoidance could fail.

Reactive Avoidance is solving most urgent situations with very short reaction opportunity. This work focus on physical obstacles and terrain. Non-cooperative intruders are partially considered. The adversary behavior was is not considered.

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

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

The *Discretization* is the key to ensure 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 [1].

6.1 Overview

The *Overview* is based on *Existing* Emergency avoidance framework [2] (fig. **??**). To achieve goals defined in *Problem Definition* (sec. **??**, **??**) following *Avoidance Framework Concept* (fig. 6.2) is proposed:

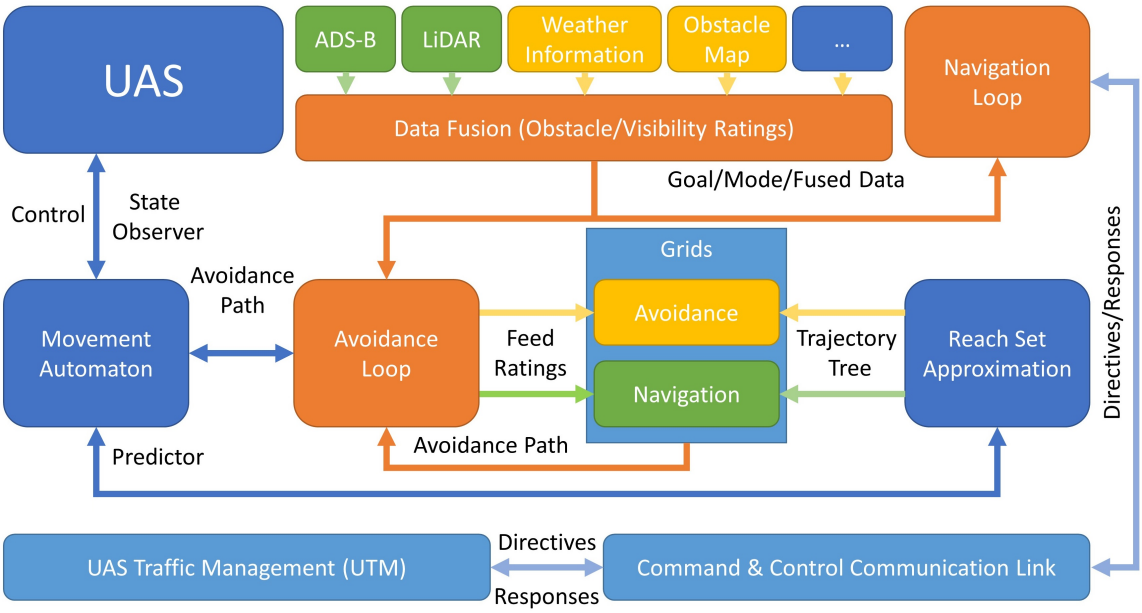


Figure 6.2: Avoidance Framework Concept.

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. 6.2.2).

2. *Movement Automaton* (Role: Control Interface/Predictor) - consumes *Discrete Command Chain* to generate discrete *reference trajectory*, it can also be used as a predictor of *future UAS states* (sec. **??**). The movement Automaton used in this work is given in (sec. 6.2.3).

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

   a. *LiDAR* (Static obstacle detection) - detection of physical obstacles (eq. 6.50)

   b. *ADS-B* (Intruder UAS/Plane detection) - detection of intruders who are broadcasting their position and sometimes heading with plans and additional parameters. The *intersection models* are given in (sec. 6.5.2, app. **??**, **??**, **??**).

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 by (def. 10).

   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 a portion of the space. The partial *ratings* are proposed in related sections. The data fusion procedure with *defuzzification* and final assessment into space sets are outlined in (sec. 6.5.4)

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. 6.4.2), the approximation methods with constrained expansion are outlined in (sec. 6.4.4, 6.4.5, 6.4.7, 6.4.6). The reach set estimation is the 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. 6.6.1).

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 are given in (sec. 6.6.2).

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 are given in (sec. 6.7).

Communication in Avoidance Framework:

1. *UAS* $\leftrightarrow$ *Movement Automaton* - sharing *actual system state*, commanding the UAS platform.

2. *Reach Set* $\leftrightarrow$ *Movement Automaton* - predicting a set of feasible trajectories for the given situation.

3. *Reach Set* $\leftrightarrow$ *Grids* - providing trajectory set depending on the 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 UAS Model and Control

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

1. *Movement Automaton Applications* (sec. 6.2.1) - the listing of related work and similar approaches to ours.
2. *UAS Model* (sec. 6.2.2) - a simple plane model used in this work as the *controlled plant*.
3. *UAS Movement Automaton* (sec. 6.2.3) - movement automaton for *UAS Nonlinear Model* constructed from scratch.

6.2.1 Movement Automaton Applications

Movement Automaton is a 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* has been discussed as an issue in following works:

1. Newton gradient method for evasive car maneuvers [3].
2. Non-holistic methods for trajectory generation [4].
3. Stochastic approach to elliptic trajectories generation [5].

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

1. Control of traffic flow [6].
2. Complex air traffic collision situation resolution system [7, 8].
3. SAA/DAA capable avoidance system [2].

6.2.2 UAS Model

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

State Vector (eq. 6.1) defined as a 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 the 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 vector function (eq. 6.3) is defined through the 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:*

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

Discretization for *fixed step* k we start with discretization of the model:
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. 1) 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:

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

6.2.3 UAS Movement Automaton

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

Movement Primitives by (def. **??**) are given as (eq. **??**). Each movement primitive will last for fixed duration $1s$.

Assumption 1. *Let assume that* transition time *of* roll, pitch, yaw, and the linear velocity *is* $0s$.

Under the assumption (as. 1) the *movement transitions* (def. **??**) have zero duration. Therefore movement primitives can be considered as movements.

Note. The assumption (as. 1) can be relaxed under the condition that *path tracking controller exists*.

Movements satisfying (def. **??**), for the nonlinear model (eq. 6.4) reduced to *discrete model* (eq. 6.10), are 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 the discrete model (eq. 6.10) is defined as a set of unitary movements on main axes (tab. 6.1) and diagonal axes (tab. 6.2).

The maneuvering capability of several commercial small fixed-wing UAS was abstracted together. The turning rate on horizontal/vertical is defined as 15°.

The deltas are posed in *UAS body-fixed coordinate frame* (ap. **??**) for discrete time k.

Parameter	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.

Parameter	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. The *movement set* in shortened form is given as:

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

The *implemented movement set example* (fig. 6.3) shows the movement used as basic building blocs of the trajectory for fixed-wing UAS:

1. *Initial position* (red plane) - the initial position, before any movement execution.

2. *Straight movement application* (blue plane) - the *neutral movement application* brings plane forward.

3. *Main axes movements* (cyan planes) - the application of movements from (tab. 6.1) {Up, $Down$, $Left$, $Right$}.

4. *Diagonal axes movements* (magenta planes) - the application of movements from (tab. 6.2) {$DownLeft$, $DownRight$, $UpLeft$, $UpRight$}.

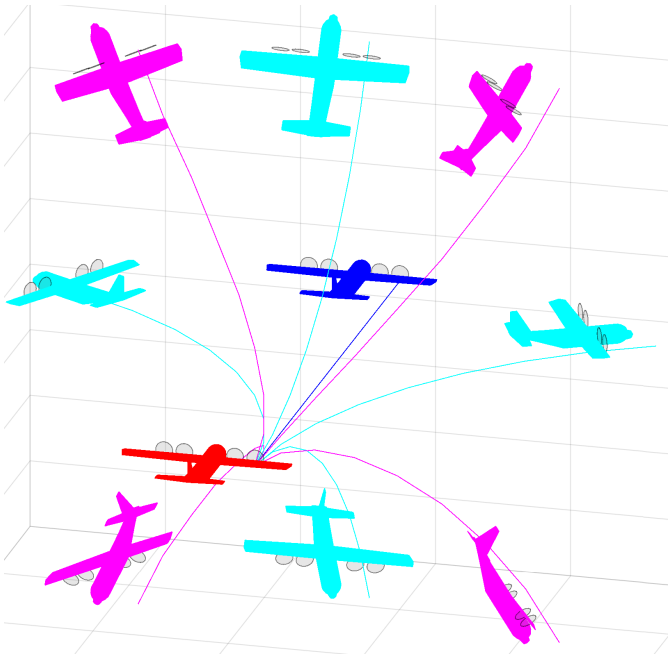


Figure 6.3: Implemented movement set example.

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

$$Buffer = \left\{ movement(j) : \begin{array}{l} movement(j) \in MovementSet(eq.6.11), \\ j \in 1 \dots n, n \in N^{+} \end{array} \right\} \tag{6.12}$$

Assumption 2. *The buffer is always non-empty, ordered, finite list of* movements.

Note. The buffer has finite count n of movements stored. The buffer is the planning instrument used by higher level navigation/avoidance algorithm to control UAS (Control/Command interface) (fig. 6.2).

The discrete trajectory (eq. 6.13) is ordered set of states bounded to discrete time $0 \dots n$, where n is movement count of *Buffer*. Trajectory set has $n + 1$ members defined like the following:

$$Trajectory(state(0), Buffer) = \left\{ \begin{array}{r} state(0) = state(0), \\ state(1) = applyMovement\left(state(0), movement(1)\right), \\ state(2) = applyMovement\left(state(1), movement(2)\right), \\ \vdots = \vdots \\ state(n-1) = applyMovement\left(state(n-2), movement(n-1)\right), \\ state(n) = applyMovement\left(state(n-1), movement(n)\right) \end{array} \right\} \tag{6.13}$$

The movement(k) vector is selected from movement tables (tab. 6.1, 6.2).

Note. Parameter movement$(\cdot)$ (eq. 6.13) is a movement order index in buffer (eq. 6.12).

6.3 Space Discretization - Avoidance Grid

Operation Space: The *Operation Space* is a space where UAS can effectively surveillance its surroundings, and it has the capability to act.

A Motivation for Discretization: The UAS surroundings needs to be represented in an *avoidance-friendly manner*, following principles matters:

1. *Discrete representation* - the space around UAS should be segmented into finite and exclusive portions which are considered as one point of the grid. This enables fast situation assessment.

2. *Threat proximity* - any form of threat gets more important with decreasing distance to UAS.

3. *LiDAR swipe density* - one LiDAR swipe scans many points; the grid needs to be customized to swipe characteristics.

The *Main Sensor* is *LiDAR* (problems **??**.-**??**). The *effective occupancy computation* needs to be done for all problems; the inspiration is taken from [9]. The *effective occupancy computation* is done in *LiDAR* scan portioned into *polar coordinates grid*. The *operation space* is abstracted as a *grid* where *space portions* are representing the points in the grid.

Note. Each member of the grid is a cell, represented as a point with properties, like threat level, visibility.

The *Discrete Situation Evaluation* is executed for a *UAS* local coordinate frame in fixed *time*. The goal is to enable *fast discrete situation assessment*.

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

$$point = [distance, horizontal^\circ, vertical^\circ]. \tag{6.14}$$

Note. For polar/Euclidean transformations and local/global coordinate frames refer to background theory (app. **??**).

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

LiDAR Swipe Portioning: The *polar coordinate space* can be portioned into distinctive cells. Each cell then represents one point in the grid.

The *reason* for this swipe portioning is *LiDAR* scanning density[1], which is extremely dense. The *threat* state in the cell can be assessed with linear complexity.

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

The *polar* $\rightarrow$ *euclidean* coordinate frame transformation is not amenable for LiDAR swipe. The *threat* assessment based on *LiDAR swipe* in *planar space portions* has minimal complexity, and it is cost effective. [10].

Cell: To discretize operational space into a grid of points there is a need to define cell space, which bounds the portion of the *local planar coordinate frame*. The point (eq. 6.14) is defined by distance, horizontal° offset angle, and vertical° offset angle. The cell is a closed compact set of such points. The boundary can be defined like follow:

Definition 1. *Cell*
The cell *bounds a portion of space in UAS local polar coordinate frame, defined by boundary ranges:*

1. Distance Range - *starts and ends:* $distance_{start} < distance_{end}$ *in* $\mathbb{R}^+$.
2. Horizontal Range - *starts and ends:* $horizontal^{\circ}_{start} < horizontal^{\circ}_{end} \in]-\pi, \pi]$.
3. Vertical Range - *starts and ends: by* $vertical^{\circ}_{start} < vertical^{\circ}_{end} \in]-\pi, \pi]$.

The space portion *belonging to the* cell *is given by function as:*

$$cell.spacePortion = \ldots$$
$$\left\{ point \in \mathbb{R}^3 \ where: \begin{pmatrix} cell.distance_{start} < & point.distance \leq & cell.distance_{end}, \\ cell.horizontal^{\circ}_{start} < & point.horizontal^{\circ} \leq & cell.horizontal^{\circ}_{end}, \\ cell.vertical^{\circ}_{start} < & point.vertical^{\circ} \leq & cell.vertical^{\circ}_{end} \end{pmatrix} \right\} \quad (6.15)$$

To evaluate a static obstacle threat*, it is necessary to know how many LiDAR hits landed in the cell space portion. For one* LiDAR Scan *the* hits set *is given a* set *of* all points *which lands into cell space portion:*

$$cell.LiDARHits = \{point \in LidarScan : point \in cell.spacePortion\} \quad (6.16)$$

Note. The *cell* space portion volume is increasing with the distance. This satisfies the requirement for threat-distance importance.

Effective Operation Space: The goal is to determine which of the operation space is going to be considered in our avoidance grid. The effective operation space determination according to [11] is influenced by the following factors:

1. *Sensors ranges* - there is no reason to assess the situation over effective *sensor range*.

2. *Information sources* impact - there is no real impact on *effective space boundary*, the information search and intersection algorithms are only of the importance.

3. *UAS maneuverability* - the space where UAS can maneuver, bounded by space-time (reach set boundary).

4. *Computation power* - the situation evaluation and threat assessment capabilities of the onboard computer.

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

Let show an example of an *effective operation space* for the UAS (fig. 6.4). The *full LiDAR Swipe* (cyan and red lines) of *UAS* (blue plane) has a *shape* of the conical cylinder.

Note. Under *ideal circumstances*, the *LiDAR swipe* would have a *ball shape*, but in real cases the *UAS body portion* where *LiDAR* is mounted is unused.

The *frontal portion* (red line) is a set of cells where *UAS* can make maneuvers. According to the *previous conditions*, there is no reason to consider a space portion out of this area.

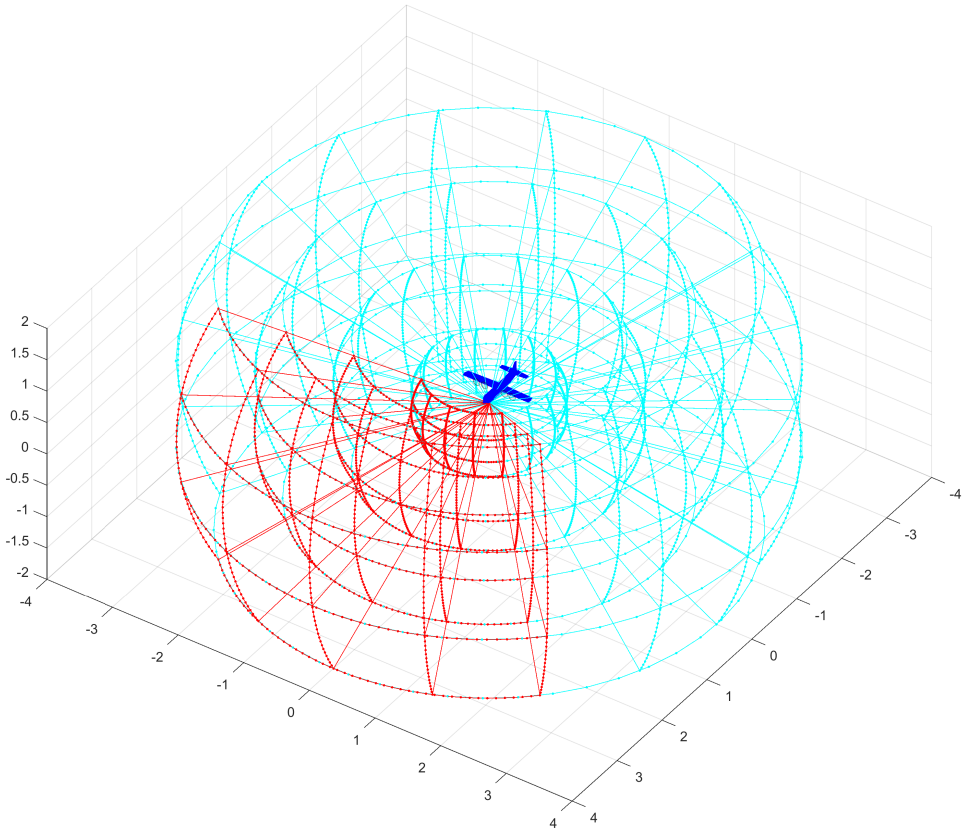


Figure 6.4: Example: The *LiDAR* reading portioning - cells.

Avoidance Grid Definition: The *effective operation space* is going to be portioned into cells.

Note. The avoidance grid is a set of cells from the full LiDAR swipe, that are reachable in limited number of movements by given control (sec. 6.2.3).

Definition 2. *Avoidance Grid*
The effective space portion *(fig. 6.4 red lines) given by a portion of space in UAS local polar coordinate frame, bounded by:*

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

The goal is to separate the effective operation space *into cells (def. 1). The idea is to split distance range into multiple distinctive distance ranges with count* $layerCount \in \mathbb{N}^+$. *The ranges for* distance layers *are given as follow:*

$$\begin{aligned} layer^i_{start} &= (i-1) \times \frac{distance_{end} - distance_{start}}{layerCount} \\ layer^i_{end} &= i \times \frac{distance_{end} - distance_{start}}{layerCount} \end{aligned} \quad ; \quad i \in 1 \dots layerCount \tag{6.17}$$

The same separation Layer horizontal/vertical separations defined by $horizontalCount \in \mathbb{N}^+ / verticalCount \in \mathbb{N}^+$:

$$\begin{aligned} horizontal^j_{start} &= (j-1) \times \frac{horizontal^{\circ}_{end} - horizontal^{\circ}_{start}}{horizontalCount} \\ horizontal^j_{end} &= j \times \frac{horizontal^{\circ}_{end} - horizontal^{\circ}_{start}}{horizontalCount} \end{aligned} \quad ; \quad j \in 1 \dots horizontalCount \tag{6.18}$$

$$\begin{aligned} vertical^k_{start} &= (k-1) \times \frac{vertical^{\circ}_{end} - vertical^{\circ}_{start}}{verticalCount} \\ vertical^k_{end} &= k \times \frac{vertical^{\circ}_{end} - vertical^{\circ}_{start}}{verticalCount} \end{aligned} \quad ; \quad k \in 1 \dots verticalCount \tag{6.19}$$

Then $cell_{i,j,k}$ *space portion by (def. 1) has the following ranges:*

1. Cell Distance Range *(eq. 6.17) depending on layer index* i.
2. Cell Horizontal Angle Range *(eq. 6.18) depending on horizontal angle index* j.
3. Cell Vertical Angle Range *(eq. 6.19) depending on vertical index* k.

Note. The example of *Avoidance Grid Cells* is given in (fig. 6.4 red boundary).

The Avoidance Grid *is the set of cells:*

$$AvoidanceGrid = \left\{ \begin{array}{c} i \in 1 \dots layerCount \\ cell_{i,j,k} : j \in 1 \dots horizontalCount \\ k \in 1 \dots verticalCount \end{array} \right\} \tag{6.20}$$

Note. For any distinctive cells $cell_{i,j,k}$, $cell_{m,n,o}$ their *space portion intersection* is empty set:

$$\forall cell_{i,j,k}, cell_{m,n,o} : cell_{i,j,k} \cap cell_{m,n,o} = \varnothing, i \neq o \vee j \neq n \vee k \neq o \tag{6.21}$$

Grid Sizing Approach: The sizing approach used in this work is outlined in (app. **??**).

Cell in Avoidance Grid Properties: For each cell in the Avoidance Grid 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 are given later in *data fusion procedure* outline (tab. 6.3).

6.4 Reach Set Approximation

Motivation: *Reach set* is strong tool for *Obstacle Avoidance* because it contains all possible *avoidance maneuvers*. The current implementations (sec. **??**) have following flaws:

1. *Realistic approximation* - *nonlinear systems* or *heavily constrained systems* cannot be approximated well by *linear continuous-time Reach Sets*.
2. *Finite count of possibilities* - continuous-time *Reach Set* contains infinite possibilities for *avoidance maneuvers*, the SAA system demands conflict resolution in finite time.
3. *Computationally feasible data structures* - 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. *Finite count of possibilities* - finite number of elements in *Reach set* will enable *scalable* calculation.
3. *Computationally feasible data structures* - 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. *Computationally feasible data-structures* - some specific behavior, like horizontal/vertical separation, or maneuver shape can be encoded into a different types of the reach set approximation algorithms.

6.4.1 Trajectory Set Approximation of 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 3 (Reach set Approximation by Movement Automaton)**.** A trajectory *(def.* **??***) for system* $\dot{state} = f(time, state, input)$ *under control of the movement automaton* $\mathcal{MA}$ *is given as execution of movement buffer (def.* **??***) with initial state of system* $state_0$*. Therefore notation* $Trajectory(state_0, buffer)$ *is used.*

The Complete Reach Set *(6.22) for system with initial state* $state_0$ *with existing* control strategy $control(time) \in Controls(time)$. *for time* $\tau > time_0$.

$$ReachSet(\tau, time_0, state_0) = \bigcup \{state(s) : control(s) \in Controls(s), s \in (time_0, \tau]\} \tag{6.22}$$

The Reach Set Approximation by Movement Automaton *(6.23) 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* $\tau > time_0$.

$$ReachSet(\tau, time_0, state_0) = \left\{ Trajectory(state_0, buffer) : \begin{matrix} duration(buffer) \\ \leq \\ (time_0 - \tau) \end{matrix} \right\} \tag{6.23}$$

Note. Reach Set Approximation (def. 3) is subset of *Full Reach Set* (def. **??**) in continuous space $\mathbb{R}^n$ it inherits all important properties, like *Invariance* [12].

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. 3) is constrained only by *future expansion time* τ. UAS makes space assessment in *Avoidance Grid*. There is no point to consider Trajectories outside of *Avoidance Grid*

Definition 4 (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:

$$ReachSet \begin{pmatrix} time_0, \\ state_0, \\ AvoidanceGrid_0 \end{pmatrix} = \left\{ \begin{matrix} Trajectory(\dots) \\ \in \\ ReachSet(6.23) \end{matrix} : \begin{matrix} \forall segment \in AvoidanceGrid_0, \\ segment \in Trajectory(\dots) \end{matrix} \right\} \tag{6.24}$$

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. 6.24) 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. **??**) and other different constraints.

Reach Set Trajectory Tree: (6.25) *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 $Trajectory(state_0, \varnothing)$.

$$ReachSet(state_0, \dots) = \left\{ Trajectory(state_0, buffer) : \begin{matrix} buffer \in Movements^i, \\ i \in \{1, \dots, k\} \end{matrix} \right\} \tag{6.25}$$

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:

$$intersection(segment, Set) : \begin{cases} \exists point \in segment, & \\ \ point \in Set & : true \\ Otherwise & : false \end{cases} \tag{6.26}$$

Definition 5 (Pruned Reach Set)**.** *For* Reach set *represented as* Trajectory Tree *(eq. 6.25) and some constraint set (Set) where exist* intersection function *(eq. 6.26). The* Pruned Reach set *is given as follows:*

$$Prune(ReachSet, Set) = \left\{ Trajectory(\dots) : \begin{matrix} \forall segment \in Trajectory, \\ \neg intersection(segment, Set) \end{matrix} \right\} \tag{6.27}$$

Note. Pruning(def. 5) [13] is applicable multiple times for various *Constraints Set*.

Example of *Approximated Reach set Calculation* (def. 3), *Reach Set Containment* (def. 4), and, *Pruning* is given in [2].

6.4.2 Distinctive Properties of the Trajectories

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. 6.25) for Avoidance Grid with *layerCount* and Movement automaton with *movementCount*, the *Node count* is given as:

$$1 + \left(\sum_{i \in \{1 \ldots layerCount\}} (movementCount)^i \right) \tag{6.28}$$

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.*

Performance Criteria: The scaling factor (eq. 6.28) 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 through 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 6 (Trajectory footprint)**.** *For* Trajectory *from* Reach set *(def. 4) defined for* Avoidance Grid *has membership function.* Membership Function *returns* ordered set of passing cells*:*

$$footprint \begin{pmatrix} Trajectory, \\ AvoidanceGrid \end{pmatrix} = \left\{ \begin{matrix} cell \in AvoidanceGrid : \\ isMember(trajectory, cell) \end{matrix} \right\} \tag{6.29}$$

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

$$isEqual \begin{pmatrix} Trajectory_1, \\ Trajectory_2, \\ AvoidanceGrid \end{pmatrix} : \begin{cases} \begin{pmatrix} footprint(Trajectory_1, \ldots) \\ = \\ footprint(Trajectory_2, \ldots) \end{pmatrix} & : true \\ Otherwise & : false \end{cases} \tag{6.30}$$

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. 6.29). Similarly there is a possibility to create *Reach set skeleton* containing unique trajectories, by using *equality function* (eq. 6.30). *Coverage set* is sufficient for now.

Definition 7 (Coverage Set)**.** Coverage set *(6.31) is defined for* Avoidance Grid *and* Reach Set *pair as set of unique* Trajectory footprints*:*

$$CoverageSet\begin{pmatrix} AvoidanceGrid, \\ ReachSet \end{pmatrix} = \left\{ footprint\begin{pmatrix} Trajectory, \\ AvoidanceGrid \end{pmatrix} : \begin{matrix} \forall Trajectory \\ \in ReachSet \end{matrix} \right\} \tag{6.31}$$

Coverage set properties: Trajectory footprint (eq. 6.29)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 8 (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*.*

$$\begin{aligned} referenceCoverage &= |CoverageSet(ReferenceReachSet, AvoidanceGrid)| \\ reducedCoverage &= |CoverageSet(ReducedReachSet, AvoidanceGrid)| \\ CoverageRatio &= \frac{reducedCoverage}{referenceCoverage} \in [0,1] \end{aligned} \tag{6.32}$$

Note. Reference Reach Set is usually *Full Reach Set* containing all possible trajectories in space contained by *Avoidance Grid*. In case *Full Reach Set* cannot 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:

$$\begin{gathered} MovementSet = SmoothMovements \cup ChaoticMovements \\ SmoothMovements \cap ChaoticMovements = \varnothing \\ |SmoothMovements| > 0, \quad |ChaoticMovements| > 0 \end{gathered} \tag{6.33}$$

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

$$\begin{aligned} isSmooth(movement) &= \begin{cases} movement \in SmoothMovements & : 1 \\ movement \in ChaoticMovements & : 0 \end{cases} \\ smoothCount(Trajectory(\ldots, buffer)) &= \sum isSmooth(movement), \\ &\qquad \forall movement \in Buffer \end{aligned} \tag{6.34}$$

Definition 9 (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. 6.33), with existing* classification *and* counter *functionals (eq. 6.34) is given as follows:*

$$Smoothness(Trajectory(\ldots, buffer)) = \frac{isSmooth(Trajectory)}{movementCount(Trajectory)} \in [0, 1] \tag{6.35}$$

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

6.4.3 Heuristic Trajectory Tree Building

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 [14]. The ideal candidate for propagation algorithm is *Wave-front* algorithm propagating *Trajectory tree* through Layers. Due the *Avoidance Grid* onion like layers, there is possibility to implement turn maneuver through layers iterative and effectively .

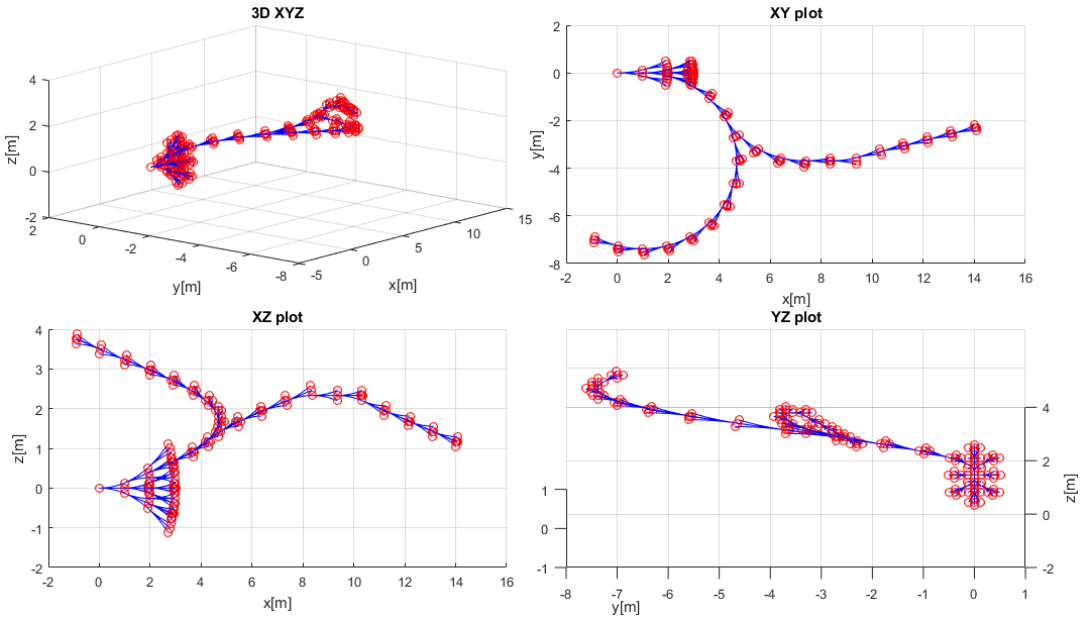


Figure 6.5: *Rapid Exploration tree as result of* Constrained trajectory expansion.

Rapid Exploration Tree (fig. 6.5) was selected, because it enables *Movement Automaton Utilization* and *Property Binding*. Similar approach was used for space exploration [15].

The example (fig. 6.5) 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 through *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 *Reachability* 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. 4).
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. 9).
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. 6.4.4, 6.4.5). *Pruning methods* over *Large Decision Trees* are *fast* and *viable* [16].

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*.

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 through, 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
```

6.4.4 Coverage-Maximizing Reach Set Aproximation

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

Background: There is *Coverage Ratio* property of *Reach Set* (def. 8). It has been shown that creating *Reach Set* via *greedy approach* is not feasible due the *Scaling Factor*. *Contracted Expansion* (sec. 6.4.3) 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 ∞. If you want to generate non redundant trajectories use ∞, 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 ∞. 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 *Coverage-Maximizing Reach Set Approximation*

Input : Node[] stack, Cell $\text{cell}_{i,j,k}$
Tuning Parameters: int^+ footprintLength, 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. 6.6). 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*.

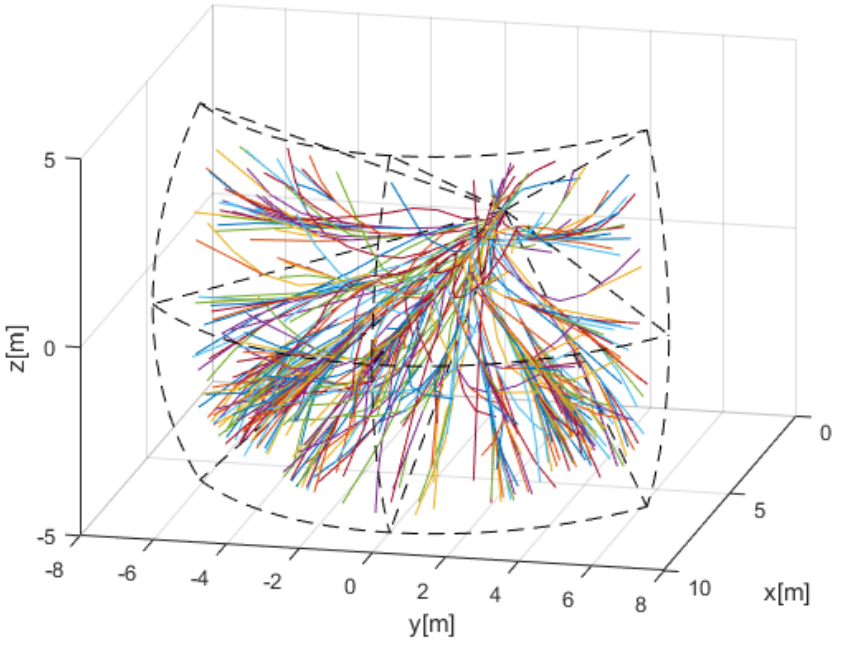


Figure 6.6: *Coverage-Maximizing* reach set *approximation.*

Pros and Cons: It can be seen from example (fig. 6.6) that *Coverage-Maximizing 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:

$$countTrajectories(ReachSet) \leq layerCellCount \times spreadLimit \times size(Movements) \quad (6.36)$$

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

$$countNodes(ReachSet) \leq layerCount \times layerCellCount \times size(Movements) \times spreadLimit \quad (6.37)$$

Absence of *Smooth Trajectories* disqualifies *Coverage Maximizing -RSA* to be used for *Navigation.* This type of reach set is feasible for *Avoidance*, because it contains variety of maneuvers.

6.4.5 Turn-Minimizing Reach Set Approximation

Motivation: Imagine having an *Avoidance Grid* like (fig. 6.4). There is a need of *Reach Set Approximation* which will have *Smooth Trajectories* (def. 9) going nearby *cell*

centers.

Background: The *Smoothness Rating for Trajectory* (def. 9) uses two distinct sets *Smooth Movements* and *Chaotic Movements* (eq. 6.33) which are defined for our *Movement Automaton* (sec. 6.2) like follow:

$$\begin{aligned} SmoothMovements &= \{Straight\} \\ ChaoticMovements &= Movements - SmoothMovements \end{aligned} \tag{6.38}$$

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. 6.4) 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 ∞. If maximal value ∞ 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 $cell_{i,j,k}$.

Algorithm 6.3: Expansion Constraint function for *Turn-Minimizing Reach Set Approximation*

Input : Node[] stack, Cell $cell_{i,j,k}$
Tuning Parameters: 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]
```

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.

$$getPerformance(Node, Cell) = \frac{distance(Node.Trajectory, Cell.Center)}{SmoothnessRate(Node.Trajectory)} \tag{6.39}$$

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 *coverage-maximizing 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*.

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 *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*.

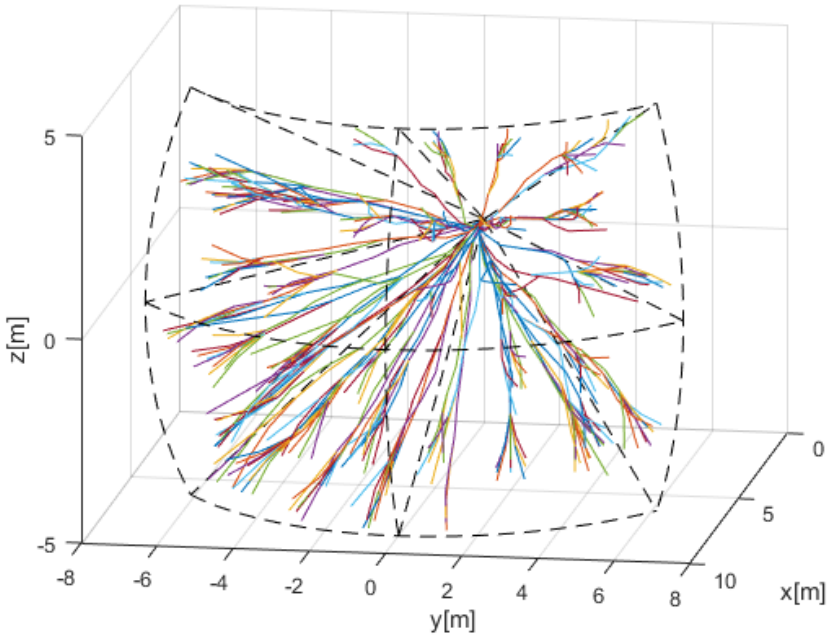


Figure 6.7: *Turn-minimizing* reach set *approximation*.

Pros and Cons: It can bee seen from example (fig. 6.7) that *Turn-Minimizing Reach Set Approximation Method* (alg. 6.3) generates *smooth evenly spread trajectories*.

High smoothness ratio (≥ 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:

$$countTrajectories(ReachSet) \leq layerCellCount \times spreadLimit \times size(Movements) \quad (6.40)$$

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

$$countNodes(ReachSet) \leq layerCount \times layerCellCount \times spreadLimit \quad (6.41)$$

Absence of *High Coverage Ratio* disqualifies *Turn-Minimizing 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.

6.4.6 ACAS-X like Reach Set Approximation

Motivation: The implementation of *ACAS-Xu* behavior in DAA system will be mandatory for *National Airspace System Integration* in United spaces [17].

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 [18]. The *Policy for Collision Avoidance* proposal has been given in [19].

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* (app. **??**). 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:

$$\begin{aligned} Horizontal &= \{Straight, Left, Right\} \\ Vertical &= \{Straight, Up, Down\} \\ Slash &= \{Straight, UpLeft, DownRight\} \\ Backslash &= \{Straight, UpRight, DownLeft\} \end{aligned} \tag{6.42}$$

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:

$$isSeparation(buffer, separation) = \begin{cases} \forall movement \in buffer, & \\ \quad movement \in separation & : true \\ otherwise & : false \end{cases} \tag{6.43}$$

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. 6.4) 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. 6.43) 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. 6.42) are not *exclusive sets* in *Movement Automaton* domain. One *Trajectory* contained by Node can belong to multiple *Separations*.

Algorithm 6.4: Expansion Constraint function for *ACAS-like Reach Set Approximation*

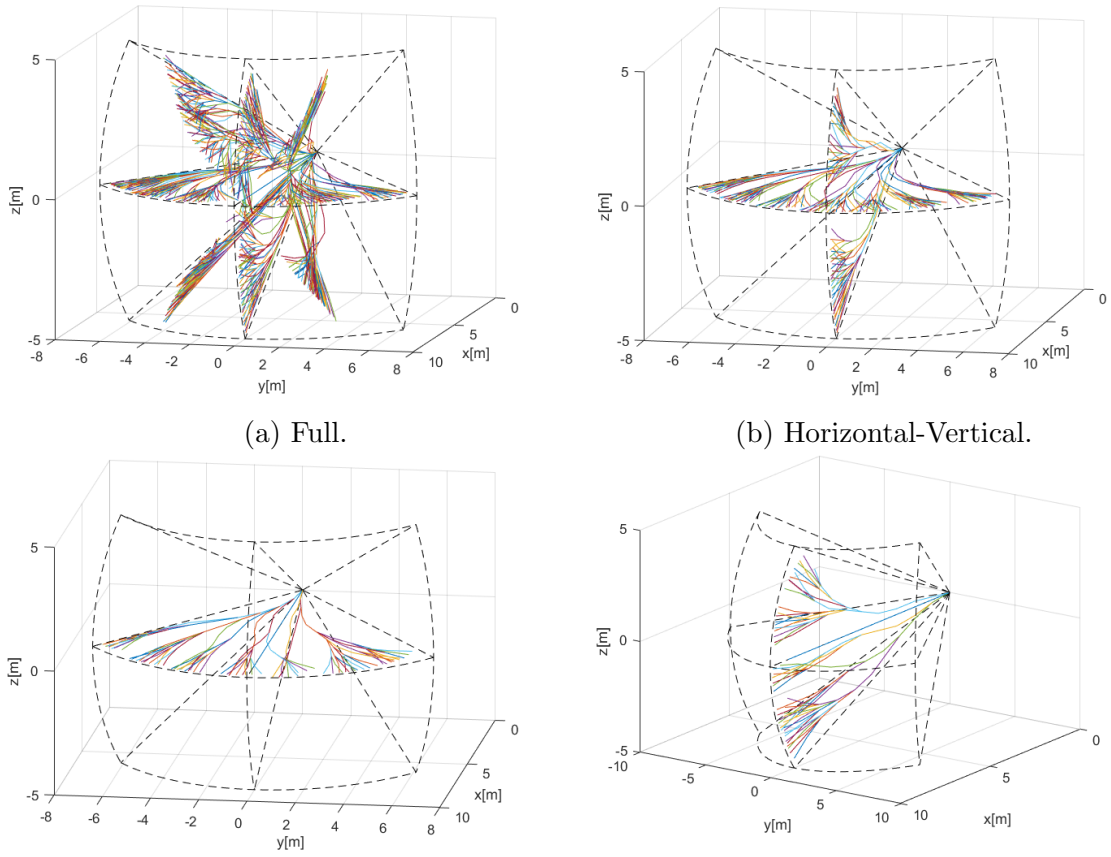
Input : Node[] stack, Cell $\text{cell}_{i,j,k}$, Separation[] separations
Tuning Parameters: $None : \varnothing$
Output : Node[] candidates, Node[] leftovers

```
# 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)(6.43) 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 [18], 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. 6.4) 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. 6.4.4, 6.4.5 6.4.7) is poor.

The *node* and *trajectory* count boundary was not implemented. It is common knowledge that *2D* avoidance sets does not require scaling [18]. Otherwise trajectory footprint mechanism like in *Turn-Minimizing 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. 6.4.7).

6.4.7 Combined Reach Set Approximation - Tree Merge

Motivation: Turn-Minimizing Reach Set Approximation (sec. 6.4.5) is *efficient* for *Navigation in* Controlled Airspace. Coverage-Maximizing Reach Set Approximation (sec. 6.4.4) 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 *Turning-Minimizing* and *Coverage-Maximizing* reach set aproximations 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.

$$\left[\begin{array}{l}\texttt{First Reach Set}\\ \quad \varnothing \rightarrow \left\langle \begin{array}{l} left \rightarrow \left\langle \begin{array}{l} left \\ right \end{array}\right. \\ \varnothing \end{array}\right.\\ \texttt{Second Reach Set}\\ \quad \varnothing \rightarrow \left\langle \begin{array}{l} \varnothing \\ right \rightarrow \left\langle \begin{array}{l} left \\ right \end{array}\right. \end{array}\right] \rightarrow \left[\begin{array}{l}\texttt{Combined Reach Set}\\ \quad \varnothing \rightarrow \left\langle \begin{array}{ll} left & \rightarrow \left\langle \begin{array}{l} left \\ right \end{array}\right. \\ right & \rightarrow \left\langle \begin{array}{l} left \\ right \end{array}\right. \end{array}\right. \end{array}\right] \tag{6.44}$$

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 [20] 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. 6.5) is implemented as function $NodecombinedReachSet(\dots)$ which takes root Node with *initial State*, *Avoidance Grid* and respective parameters for each calculation method. *turn-minimizing spread* for *Turn-Minimizing Reach set calculation* and *coverage spread*, *Footprint Length* for *Coverage-Maximizing Reach Set Approximation*.

Separate Reach Sets are calculated using *Wave-front propagation* (alg. 6.1) using respective *Constrained Expansion* functions for *Turn-Minimizing* (alg. 6.3) and *Coverage-Maximizing* (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* additional manuevering, calculate *Smoothness Rating for Trajectory* (def. 9) 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:

$$Cost(Node) = Cost(Node.Trajectory) \times \ldots \\ \cdots \times (1 + (1 - SmoothnessRate(Node.Trajectory))) \quad (6.45)$$

Tree Merge Function $mergeTree(\ldots)$ implements *Outer Join* operation on two trees. Example was given in (eq. 6.44). Function is applied on *root Node* iterating over

Movements in Movement Set, because *Movement is pivot*.

Algorithm 6.5: 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+ coverageSpread,
 int+ turnSpread, int+ footprintLength)
    Node cmrsa = chaoticReachSet(root,grid, footprintLength,coverageSpread);
    Node tmrsa = harmonicReachSet(root,grid, turnSpread);
    Node combined = mergeTree(cmrsa,tmrsa);
    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. 6.9). 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 *Coverage-Maximizing Spread* was set to 8, *Footprint Length* to 3 and *Turn-Minimizing Spread* to 1.

Note. Notice there are typical trajectories from both *Turn-Minimizing* (fig. 6.7) and *Coverage-Maximizing* (fig. 6.6) *Reach Set Approximations*.

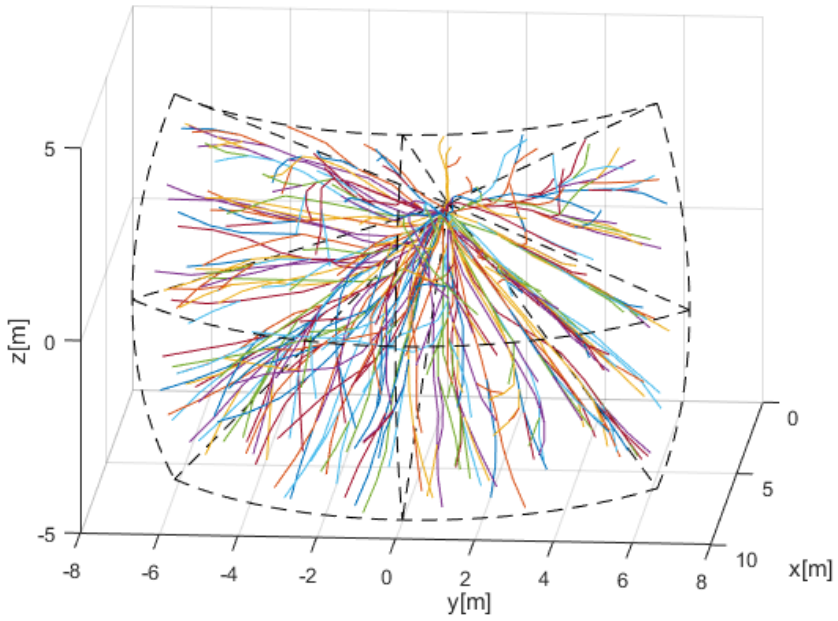


Figure 6.9: *Combined* reach set *approximation.*

Pros and Cons: It can bee seen from example (fig. 6.9) that *Combined Reach Set Approximation* (alg. 6.5) contains both types of maneuvers. *Cheaper turn-minimizing* for navigation and *More Expensive Coverage-Maximizing* for *Emergency Avoidance.* The upper limit of trajectories is given as follow:

$$countTrajectories(ReachSet) \leq countTrajectories(CM - RSA) + countTrajectories(TM - RSA) \quad (6.46)$$

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

$$countNodes(ReachSet) \leq countNodes(CM - RSA) + countNodes(TM - RSA) \quad (6.47)$$

Turn-Minimizing Reach Set is ideal for *Non-controlled Airspace* missions, because it contains *Automatic Mode Switch* between *Navigation* and *Emergency Avoidance.*

6.5 Situation Representation in the Avoidance Grid

This section gives overview how different types of threat are projected into *avoidance grid*:

1. *Obstacles* (sec. 6.5.1) - how static obstacles are represented, how map data are processed and represented, how concept of visibility impact certainty of obstacle in space.

2. *Intruders* (sec. 6.5.2) - how intruders are projected into avoidance grid, how great is probability to encounter specific intruder in given space and time.

3. *Constraints* (sec. 6.5.3) - how are constraints like geo-fencing or weather represented, how is their impact on space calculated.

4. *Data Fusion* (sec. 6.5.4) - how is final threat in cell calculated, how this threat impact the safety of passing trajectories, how we know which cells in grid are safe.

6.5.1 Obstacles

Introduction: The *static obstacles* were used in original concept [21], 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 [22].

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 $\mathrm{d}horizontal^\circ, \mathrm{d}vertical^\circ$ is unitary angle offset in which one LiDAR send and return is executed. That means the *LiDAR* ray is sent every $\mathrm{d}horizontal^\circ, \mathrm{d}vertical^\circ$ offset movement. The *LiDAR* ray density is decreasing with *distance offset.* The same amount of *LiDAR* rays passes through $cell_{i,j,k}$ in Avoidance Grid.

The surface of area given by some distance d, and unitary offsets$\partial horizontal°$, d$vertical°$ 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. 6.10) 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).

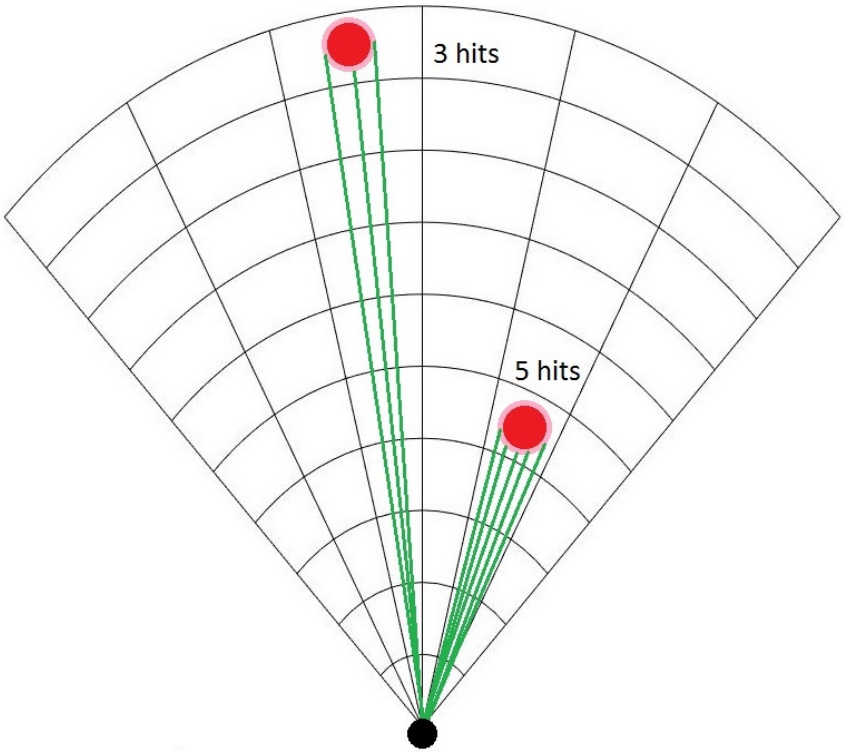


Figure 6.10: Different count of LiDAR hits with different distance from UAS.

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 [21].

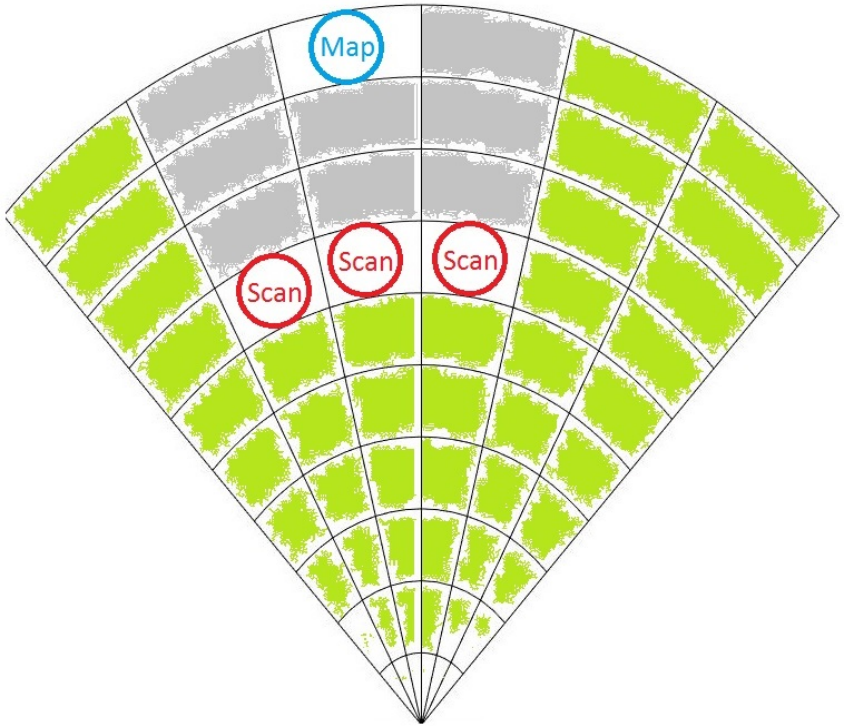


Figure 6.11: Overshadowed map obstacle by detected obstacles.

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. 6.11, 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.

Detected Obstacles 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. 6.86). 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:

$$lidarRays(cell_{i,j,k}) = \left\{ \begin{bmatrix} horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \tag{6.48}$$

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. 6.48) is finite countable and nonempty for any $c_{i,j,k}$, otherwise it will contradict the definition of avoidance grid (def. 2).
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. 6.49) in cell $cell_{i,j,k}$ is defined like follow:

$$lidarHits(cell_{i,j,k}) = \left\{ \begin{bmatrix} distance = lidarScan(), \\ horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} distance \in cell_{i,j,k}.distanceRange, \\ horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \tag{6.49}$$

The *naive* obstacle rate in case of LiDAR sensor defined as ratio between landed hits and possible hits:

$$obstacle^{LiDAR}_{cell_{i,j,k}} = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \tag{6.50}$$

Note. The *naive obstacle rate* (eq. 6.50) 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. 6.51) rate is naturally defined as supplement to naive obstacle rate. This definition is sufficient, because its reflecting the *remaining sensing capability* of LiDAR.

$$hindrance^{LiDAR}_{cell_{i,j,k}} = 1 - \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \tag{6.51}$$

Cell Density Function: Let's start with differential form of cell surface (eq. **??**). 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. **??**) and using horizontal range parameter and inverted vertical range parameter following surface integral is obtained (eq. 6.52).

$$Area(cell_{i,j,k}) = \int_{horizontal^{\circ}_{start}}^{horizontal^{\circ}_{end}} \int_{vertical^{\circ}_{end}}^{vertical^{\circ}_{start}} radius^2 \cos(vertical^{\circ}) \quad \mathrm{d}vertical^{\circ}\mathrm{d}horizontal^{\circ} \tag{6.52}$$

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:

$$Area(radius, horizontalRange, vertical^{\circ}_{start}, vertical^{\circ}_{end}) = \ldots$$
$$= \begin{cases} vertical^{\circ}_{start} < 0, vertical^{\circ}_{end} \leq 0 : \\ \qquad radius^2(\sin|vertical^{\circ}_{start}| - \sin|vertical^{\circ}_{end}|) \times horizontalRange) \\ vertical^{\circ}_{start} < 0, vertical^{\circ}_{end} > 0 : \\ \qquad r^2(\sin|vertical^{\circ}_{start}| + \sin|vertical^{\circ}_{end}|) \times horizontalRange) \\ vertical^{\circ}_{start} \geq 0 vertical^{\circ}_{end} < 0 : \\ \qquad r^2(\sin vertical^{\circ}_{end} - \sin vertical^{\circ}_{start}) \times horizontalRange) \end{cases} \tag{6.53}$$

An intersection surface for cell is defined in (eq. 6.53). Area covered by LiDAR hits (eq. 6.54) is defined as LiDAR hit rate (hits to passing rays ratio) multiplied by *Average* cell intersection surface (eq. 6.53).

$$lidarHitArea(cell_{i,j,k}) = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \times Area \begin{pmatrix} radius, horizontalRange, \\ vertical^{\circ}_{start}, vertical^{\circ}_{end} \end{pmatrix} \tag{6.54}$$

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. 6.55) and it removes bias of naive approach (eq.6.50).

$$obstacle(LiDAR, cell_{i,j,k}) = \min\left\{\frac{lidarHitArea(cell_{i,j,k})}{UAS.lidarThresholdArea}, 1\right\} \tag{6.55}$$

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. 6.51). 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. 6.56).

$$cellRow(j_{fix}, k_{fix}) = \left\{cell_{i,j,k} \in AvoidanceGrid : \begin{array}{l} i \in \{1, .., layersCount\}, \\ j = j_{fix}, k = k_{fix} \end{array}\right\} \tag{6.56}$$

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. 6.57).

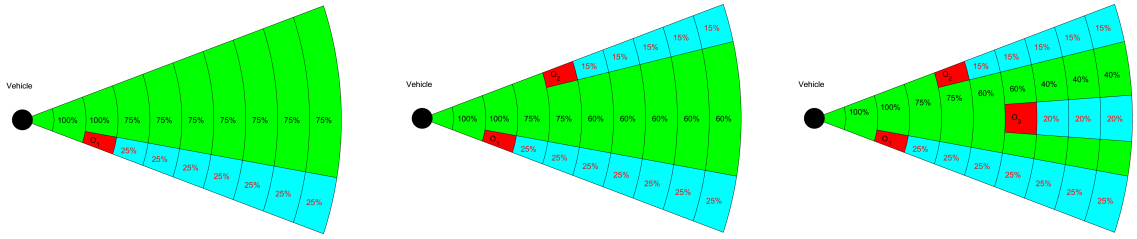
$$\begin{aligned} visibility(cell_{i_c,j_c,k_c}) &= \ldots \\ \cdots &= 1 - \sum_{index \in \mathbb{N}^+}^{index<i_c} hindrance(cell_{a,j_c,k_c} : cell_{a,j_c,k_c} \in cellRow(j_c, k_c) \end{aligned} \tag{6.57}$$

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:

$$0 \leq \sum_{cell \in cellRow(j_{fix},k_{fix})} visibility(cell) \leq 1 \tag{6.58}$$

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.



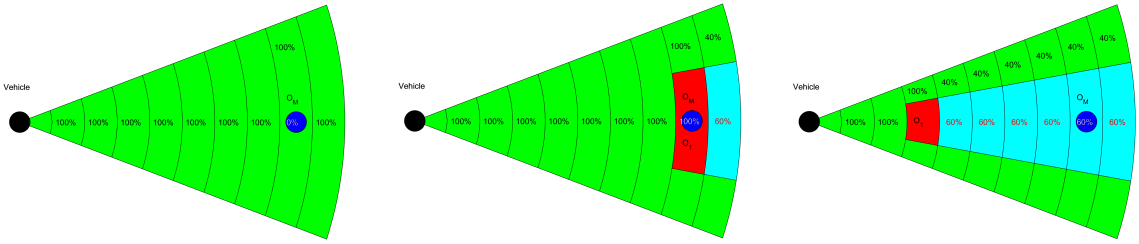
(a) 1st hindrance. (b) 2nd hindrance. (c) 3rd hindrance.

Figure 6.12: 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. 6.12a) - $\mathcal{O}_1(t_1) = \{o_1\}$ 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. 6.12b) - $\mathcal{O}_2(t_2) = \{o_1, o_2\}$ 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. 6.12c) - $\mathcal{O}_3(t_3) = \{o_1, o_2, o_3\}$ 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.

Map Obstacles: Use *stored LiDAR readings* from previous mission to build an compact obstacle map [23]. Then use *this map* as a additional information source.



(a) Undetected. (b) Detected. (c) Hindered.

Figure 6.13: Map obstacle states after *Data fusion*.

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. 6.13a), 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. 6.13b). 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. 6.13c). The detected obstacle O_1 is in $cell_{i,j,k}$ and is reducing visibility in follow up $cellRow_{i_f>i,j,k}$ by 60 percent.

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. 6.5.4).

First we start with obstacle map definition. The obstacle map (eq. 6.59) 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.

$$obstacleMap = \left\{ \begin{bmatrix} position, \\ orientation, \\ safetyMargin, \\ parameters \end{bmatrix} : \begin{array}{l} position \in \mathbb{R}^3(GCF), \\ orientation \in \mathbb{R}^3(GCF), \\ safetyMargin \in \mathbb{R}^+(m), \\ parameters \in \{\dots\} \end{array} \right\} \tag{6.59}$$

The *Map Obstacle* concept is taken from my *master student work* [23], implementing *compact representation* of point-cloud obstacle map. Te example of *cuboid obstacles* with *safe zone* is given in (fig. 6.14).

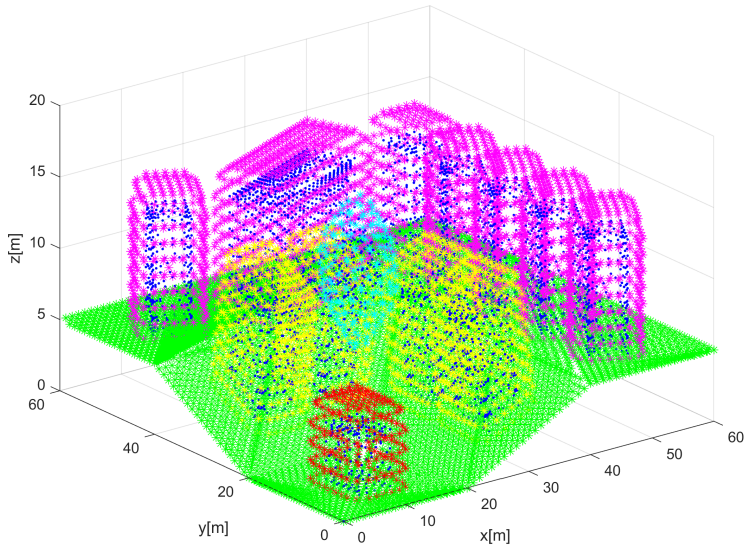


Figure 6.14: Example of Extracted Map Obstacle [23].

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. 6.60).

$$avoidanceGrid.radius \geq distance(UAS.position, mapObstacle) - totalMargin \quad (6.60)$$

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. 6.61). This point cloud is considered as hard constraints.

$$bodyPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapObstacleBody\} \quad (6.61)$$

2. *Safety Margin Point Cloud* - representing safety coating around mapped obstacle body approximation (eq. 6.62). This point cloud is considered as soft constraint.

$$marginPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapSafetyMargin\} \quad (6.62)$$

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. 6.61, 6.62) are homogeneous [23]. That means *each point* in point clouds covers similar portion of object volume. There is *threshold volume* (eq. 6.63) which represents minimal object volume to be considered as an *obstacle*.

$$0 < thresholdVolume \leq \frac{volume(pointCloud)}{|pointCloud|} \tag{6.63}$$

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. 6.64).

$$point.rate = \frac{pointCloudVolume}{thresholdVolume} \times \frac{1}{|pointCloud|} \tag{6.64}$$

The *intersection set* between *point cloud* and $cell_{i,j,k}$ is defined in (eq. 6.64). The *cell* intersection with points is defined in (eq. 6.15).

$$intersection(map, cell_{i,j,k}) = \ldots$$
$$\ldots \{points \in \mathbb{R}^3 : (point \rightarrow AvoidanceGridFrame) \in cell_{i,j,k}\} \tag{6.65}$$

The *map obstacle rating* for $cell_{i,j,k}$ and obstacle for our *information source* is defined in (eq. 6.66).

$$map(cell_{i,j,k}, obstacle) = \max \left\{ \sum_{\forall point \in intersection(map, cell_{i,j,k})} point.rate, 1 \right\} \tag{6.66}$$

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. 6.67).

$$map(cell_{i,j,k} = \max \{map(cell_{i,j,k}, obstacle) : \forall obstacle \in ActiveMapObstacles\} \tag{6.67}$$

Note. The *body point clouds* (eq. 6.61) never intersects, because they are created for inclusive obstacles. The *safety margin point clouds* (eq. 6.62) can intersects, because they represents protection zones around physical obstacles. Therefore the *maximum obstacle rating* (eq. 6.67) needs to be selected.

6.5.2 Intruders

Intruder behaviour: *Adversarial behaviour* of moving obstacle is trying to destroy avoiding our UAS. The *Intruder* UAS [24] 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 through 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.*

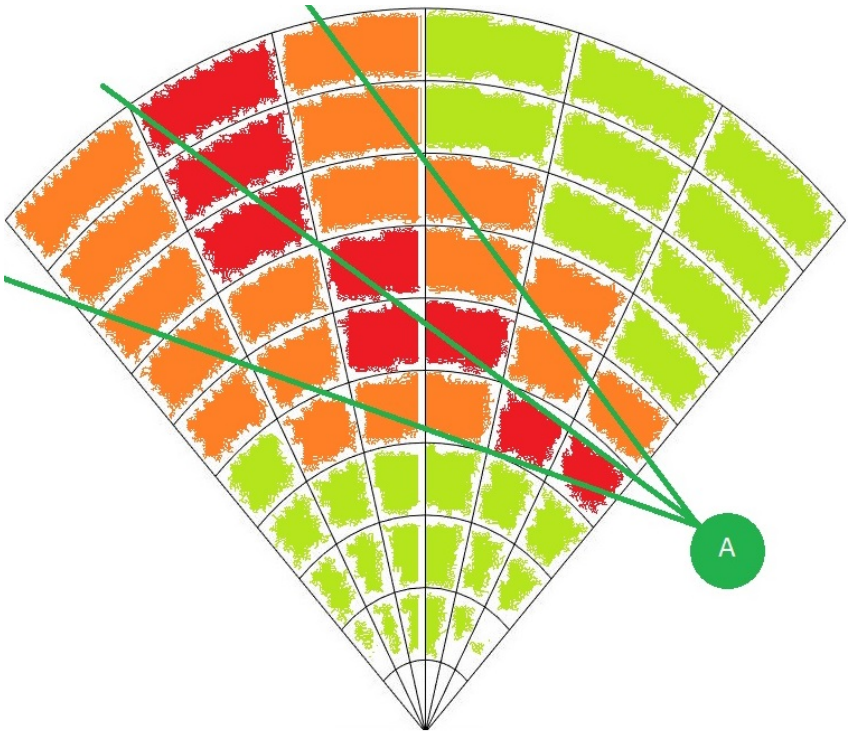


Figure 6.15: Intruder UAS intersection rate along expected trajectory.

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. 6.15). 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 through *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. 2) 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 [25] was solved by application of method [26].

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:[27] the selected algorithm *Shamos-Hoey* [28].

Note. The *Intruder Intersection* models are based on *analytically geometry* for *cones and ellipsoids* taken from [29].

Intruder Behaviour Prediction: *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:

$$\mathrm{d}position/\mathrm{d}time = velocity \quad | \quad \begin{aligned} position_x(t) &= position_x(0) + velocity_x \times t \\ position_y(t) &= position_y(0) + velocity_y \times t \\ position_z(t) &= position_z(0) + velocity_z \times t \end{aligned} \tag{6.68}$$

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. 6.15). There is *Trajectory Intersection* function from (def. 4) 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. 6.69), if there is no *passing trajectories* the UAS *entry time* is set to 0.

$$UAS_{entry}(cell_{i,j,k}) = \min \left\{ \begin{matrix} 0, entry(Trajectory, cell_{i,j,k}) : \\ Trajectory \in PassingTrajectories \end{matrix} \right\} \tag{6.69}$$

The UAS *Cell Leave* time is given as maximum of all *passing trajectory segments* entry times (eq. 6.70), if there is no *passing trajectories* the UAS *leave time* is set to 0.

$$UAS_{leave}(cell_{i,j,k}) = \max \left\{ \begin{matrix} 0, leave(Trajectory, cell_{i,j,k}) : \\ Trajectory \in PassingTrajectories \end{matrix} \right\} \tag{6.70}$$

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. 6.69) and leave (eq. 6.70).

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. 6.71).

$$time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} = \circ \end{pmatrix} = \frac{\left| \begin{matrix} [intruder_{entry}(\circ), intruder_{leave}(\circ)] \\ \cap \\ [UAS_{entry}(\circ), UAS_{leave}(\circ)] \end{matrix} \right|}{|[UAS_{entry}(\circ), UAS_{leave}(cell_{\circ})]|} \tag{6.71}$$

Intruder Intersection Rate: The *Intruder Intersection Rate* (eq. 6.72) is calculated as *multiplication* of *space intersection rate* (defined later) and *time intersection rate* (eq. 6.71).

$$intruder \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} = time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \times space \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \tag{6.72}$$

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. 6.73) is bounded to discrete point in intersection model [25, 26]. The intruder *entry/leave time* is calculated similar to *UAS cell entry (eq. 6.69)/leave (eq. 6.70) time.*

$$pointReachTime(Intruder, point) = \frac{distance(Intruder.initialPosition, point)}{|Intruder.velocity} \tag{6.73}$$

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. 6.68), 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.6.68).

   b. *Volume line* - intersection is going along the trajectory line defined by intruder model (eq. 6.68) 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. 6.68) $time \in [0, \infty]$. The cone width is set by horizontal and vertical spread.

6.5.3 Constraints

Static Constraints: 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 [30, 31].

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. **??**, **??**) and geo-fencing 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. 6.74) 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.

$$constraint = \{position, boundary, altitude_{start}, altitude_{end}, safetyMargin\} \tag{6.74}$$

Active constrain selection: The *active constraints* are constraints which are impacting *UAS active avoidance range*.

The *active constraints set* (eq. 6.75) 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*.

$$\begin{aligned} &ActiveConstraints = \dots \\ &\dots = \left\{ \begin{aligned} &constraint \in InformationSource : \\ &\qquad distance(constraint, UAS) \leq AvoidanceGrid.distance, \\ &\qquad constraint.altitudeRange \cap UAS.altitudeRange \neq \varnothing \end{aligned} \right\} \end{aligned} \tag{6.75}$$

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* [31] is used.

First the *set of contraint edges* (eq. 6.76) is a enclosed set of 2D edges between neighboring points defined as follow:

$$edges(constraint) = \left\{ \begin{aligned} &point \in boundary, \\ \left[point_i, point_j\right] &: i \in \{1, \dots, |boundary|\}, \\ &j \in \{2, \dots, |boundary|, 1\} \end{aligned} \right\} \tag{6.76}$$

The *constraint circle boundary* with calculated center on 2D horizontal plane and radius (representing body margin) is defined in (eq. 6.77).

$$circle(constraint) = \begin{bmatrix} center = \dfrac{\sum boundary.point}{|boundary.point|} + correction \\ radius = smallestCircle(edges(constraints)) \end{bmatrix} \tag{6.77}$$

The ($cell_{i,j,k}$ and *constraint* intersection (eq. 6.78) 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 $]radius, radius + safetyMargin]$.

$$\begin{aligned} &intersection, constraint) = \dots \\ &\dots = \begin{cases} hard & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) \leq \dots \\ \quad \dots \leq circle(constraint).radius, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\ soft & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) > \dots \\ \quad \dots > circle(constraint).radius, \\ distance(cell_{i,j,k}, circle(constraint)) \leq \dots \\ \quad \dots \leq circle(constraint).radius + safetyMargin, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\ none & : otherwise \end{cases} \end{aligned} \tag{6.78}$$

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:

$$cell_{i,j,k}.softConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = soft \end{matrix} \right\} \tag{6.79}$$

The constraints which have an *hard intersection with cell* are added to cells impacting constraints set:

$$cell_{i,j,k}.hardConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ intersection(cell_{i,j,k}, constraint) = hard \end{matrix} \right\} \quad (6.80)$$

Note. The final *constraint rate value* (eq. 6.89) is determined based on *mission control run* feed to *avoidance grid* (fig. 6.22) defined in 7^{th} to 10^{th} step.

Moving Constraints: 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. 6.70), which means concept from static obstacles can be fully reused.

Definition: The *moving constraint definition* (eq. 6.81) covers minimal data scope for moving constraint, assuming linear constraint movement.

Definition 10. *Moving Constraints The original definition (eq. 6.74) 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.*

$$\begin{aligned} constraint = \{&position, boundary, \dots \\ &\dots, velocity, detectionTime, \dots \\ &\dots altitude_{start}, altitude_{end}, safetyMargin\} \end{aligned} \quad (6.81)$$

Cell Intersection: The *intersection algorithm* follows (eq. 6.78), only shift of the *center and boundary points* is required.

First let us introduce $\Delta time$ (eq. 6.82), which represents difference between the constraint detection time and expected cell leave time (eq. 6.70).

$$\Delta time = UAS_{leave}(cell_{i,j,k}) - detectionTime \quad (6.82)$$

The constraint boundary is shifted to:

$$shiftedBoundary(constraint) = \{newPoint = point + velocity \times \Delta time : \ldots \\ \ldots \forall point \in constraint.boundary\} \quad (6.83)$$

The constraint center is shifted to:

$$shiftedCenter(constraint) = constraint.center + velocity \quad (6.84)$$

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 [27], the selected algorithm is *Shamos-Hoey* [28].

The implementation was tested on *Storm scenario* (sec. **??**) and it yelds same results.

6.5.4 Data fusion

The data fusion interfaces *Sensor Field* and *Information Sources* from *cell/trajectory properties*. The *Data Fusion Function* is outlined in (**??**).

First, there will be an outline of *Partial Rating* 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 the 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 the *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 the cell is given in (eq. 6.85) as minimal visibility calculated from all capable sensors in *Sensor Field*.

$$visibility(cell_{i,j,k}) = \min \left\{ \begin{matrix} visibility(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \quad (6.85)$$

The example of *visibility* calculation for *LiDAR* sensor is given in (fig. 6.13).

Note. Sensor reliability for *visibility* is already accounted for 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 the range $[0, 1]$ reflecting the probability of obstacle occurrence in a given cell.

The *maximal value* of *detected obstacle* rating is selected from readings multiplied by *visibility rating* to enforce *visibility bias*.

$$obstacle(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \times \dots \\ \dots \times visibility(cell_{i,j,k}) \quad (6.86)$$

The example of *detected obstacle rating* calculation for *LiDAR* sensor is given in (eq. 6.50).

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 a given cell. This property is bound to *Information Source*, and it has the *range* in $[0, 1]$.

The *Map Obstacle Rating* for a cell (eq. 6.87) is calculated as the product of maximal *Map Obstacle Rating* and *inverse visibility*. This gives *visibility biased* certainty of *Map Obstacle*.

$$map(cell_{i,j,k}) = \max \left\{ \begin{matrix} map(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \times \dots \\ \dots \times (1 - visibility(cell_{i,j,k})) \quad (6.87)$$

The example of *Map Obstacle Rating* calculation is given in (fig. 6.13).

Intruder: There is a set of *Active Intruders*, each intruder is using its *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 cannot be addressed by the simple *maximum*. The proposed formula (eq. 6.88) is simply bypassing the intruder rating if there is one intruder. If there are more intruders, the geometrical bias is applied.

$$intruder(cell_{i,j,k}) = 1 - \prod_{\forall intruder_i \in Intruders} \left(1 - intersection \begin{pmatrix} cell_{i,j,k}, \\ intruder_i \end{pmatrix}\right) \tag{6.88}$$

The *intruder intersection models* are outlined in (app. **??**).

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. 6.89) is in the *range* $[0, 1]$ reflecting certainty of constraint application in the cell (usually 1).

$$constraint(cell_{i,j,k}) = \max \left\{ \begin{matrix} constraint(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \tag{6.89}$$

The *Constraint Rating* calculation example for *static* constraints is given in (sec. 6.5.3), the example for *moving* constraints is given by (def. 10).

Note. Weather is already considered in constraints; the weather is handled as soft/hard static/moving constraints.

Threat: The concept of threat is a *rating of expected harm* to receive in a given portion 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. 6.22). All *sources of harm* are considered as equal. The threat is formalized in the *following definition*:

Definition 11. *The Threat is considered as any source of harm. The threat is a* maximal aggregation *of various harm ratings. Our* threat *for a specific cell is defined by (eq. 6.90).*

$$threat(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}), map(cell_{i,j,k}), \\ intruder(cell_{i,j,k}), constraint(cell_{i,j,k}) \end{matrix} \right\} \tag{6.90}$$

Reachability: The *Reachability* for trajectory reflects how safe is the *path along*. The *Threat* (def. 11) for each cell has been already assessed. The set of *Passing Cells* is defined in *Trajectory Footprint* (eq. 6.29).

The *Trajectory Reachability* is given as a product of *Threats* along the trajectory (eq. 6.91). The *Trajectory Reachability* can be calculated for each *trajectory segment* given as $\{movement_1, \ldots, movement_i\} \subset Buffer$ originating from $state_0$.

$$reachibility(Trajectory) = \prod_{PassingCells}^{\forall cell_{i,j,k} \in} (1 - threat(c_{i,j,k})) \tag{6.91}$$

Note. The *Reachability* of *trajectory* segment gives the property of *safety* of route from the beginning, until the last point of the segment. There can be a very unsafe trajectory which is very safe from the beginning.

The *Reachability* of the *cell* is given by the best trajectory segment passing through 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 the 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. 6.92).

$$reachability(cell_{i,j,k}) = \max\{Trajectory.Segment(cell_{i,j,k}).Reachability : \forall Trajectory \in PassingTrajectories(celli, j, k)\} \quad (6.92)$$

Note. Function Trajectory.Segment($cell_{i,j,k}$). Reachability gives same results for any segment in $cell_{i,j,k}$, because (eq. 6.91) 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* [32] demands the fail rate below 10^{-7} because there is no definition for *UAS* the *minimal fail rate* is expected to be at a similar level.

The *fuzzy values* $[0, 1]$ are projected to *Boolean* properties of *cell* and *Trajectory* in the following manner (tab. 6.3).

The high values of *Visibility* (eq. 6.85) and *Reachability* (eq. 6.92, 6.91) are expected. The low *threshold* for *threats* values is expected. The error margin is solved by *Sensor Fusion*, therefore, initial *false positive* cases have a low rate. The *Detected Obstacle Rate* (eq. 6.86), *Map Obstacle Rate* (eq. 6.87), *Intruder Rate* (eq. 6.88), and *Constraint Rate* (eq. 6.89) thresholds are considered low.

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	$reachability(trajectory)$	$\geq$	$(1 - threshold)$
Reachable Cell	$reachibility(cell_{i,j,k})$	$\geq$	$(1 - threshold)$

Table 6.3: Changing ratings from fuzzy to Boolean parameters.

Space Classification: The *Data Fusion Function* is outlined in (**??**). This classification is resulting in four distinct cell sets.

The *Uncertain* space for decision time t_i is a portion of *Avoidance Grid* which *UAS* cannot *read* with *Sensor Field*. The *cells* with a $\neg Visible$ property. The *Uncertain* space is given by (eq. 6.93).

$$Uncertain(t_i) = \{cell_{i,j,k} : cell_{i,j,k} \in AvoidanceGrid(t_i), cell_{i,j,k}.\neg Visible\} \tag{6.93}$$

The *Occupied* space for decision time t_i is the set of cells which are classified as *Detected Obstacles*. The *Visibility* is not an issue, due to the initial damping in (eq. 6.86). The formal definition is the space portion where it is possible to detect *obstacle bodies* or their portions (eq. 6.94).

$$Occupied(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.DetectedObstacle \end{array} \right\} \tag{6.94}$$

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. 6.95).

$$Constrained(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Visible, \\ cell_{i,j,k}.Constraint \vee cell_{i,j,k}.Intruder \end{array} \right\} \tag{6.95}$$

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. 6.96).

$$\begin{aligned} Free(t_i) = AvoidanceGrid(t_i) - \ldots \\ \cdots - (Uncertain(t_i) \cup Occupied(t_i) \cup Constrained(t_i)) \end{aligned} \tag{6.96}$$

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. 6.97).

$$Reachable(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Reachable \end{array} \right\} \tag{6.97}$$

Note. The Reachable Space at decision time t_i: The *Reachable space* is a non-empty set and its a subset of *Free(t_i)* space:

$$|Reachable(t_i)| > 0, \quad Reachable(t_i) \subset Free(t) \tag{6.98}$$

6.6 Avoidance Concept

This section introduces *Platform Independent Avoidance Concept* core functionality (fig. 6.2) modules responsible for *pathfinding* and *navigation*. The sections are organized like follow:

1. *Avoidance Grid Run* (sec.6.6.1) (inner avoidance run) - the *best pathfinding* in one *Avoidance Grid* with *situation assessment* done.

2. *Mission Control Run* (sec . 6.6.2) (outer navigation run) - main navigation and decision making an algorithm for *non-cooperative obstacle avoidance*.

3. *Computation Complexity* (sec. 6.6.3) - the *computational feasibility study* and *weak point identification* of our approach.

6.6.1 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. 6.5.4).

Note. The *rating calculation* is outlined in (sec. 6.5.4). Low-cost sensor fusion example usable to feed our data fusion procedure is given in [33]. Semi-optimal concatenation trajectory search like ours can be found in [34].

Note. The *Sensor Fusion Procedure* is solving all the following steps (sec. 6.5.4). The *main purpose* of *Avoidance Run* is finding the best path under certain conditions.

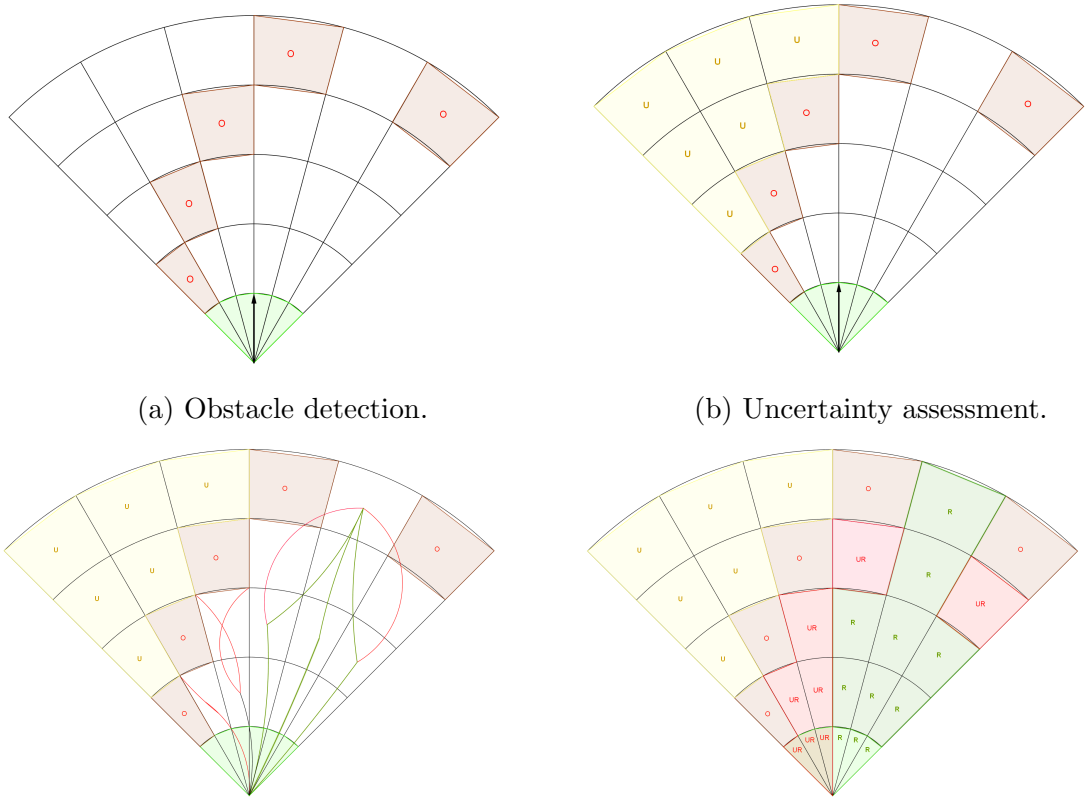
Space Assessment Principle: The *Avoidance Grid* is fed through *Data Fusion* (sec. 6.5.4). The process of *rating assessment* (tab. 6.3) is given in (fig. 6.16):

1. *Obstacle detection* (fig. 6.16a) - assessment of *detected obstacles* (eq. 6.86). The red (O) *cells* have Detected obstacle set as *true*. The other threats: *map obstacles* (eq. 6.87), *intruders* (eq. 6.88), *constraints* (eq. 6.89) are false. The red (0) *cells* are representing $Occupied(t_i)$ (eq. 6.94) space in *Avoidance Grid* at decision time t_i.

2. *Uncertainty assessment* (fig. 6.16b) - the uncertain cells are cells which status cannot be *assessed*. The *Visibility* (eq. 6.85) is low. The *Uncertain* cells (yellow (U) mark) are equal to $Uncertain(t_i)$ (eq. 6.93) in *Avoidance Grid* in *decision time* t_i. The $Constrained(t_i)$ (eq. 6.94) space is equal to $\varnothing$ in this example.

3. *Trajectory reachability evaluation* (fig. 6.16c) - the *Reach Set* given as *Trajectory Set* (eq. 6.25). is then projected through *Avoidance Grid* and pruned according to (def. 5). *Reachable Trajectories* (eq. 6.91) are only those contained in $Free(t_i)$ space (eq. 6.96). The *Reachable Trajectories* are denoted as *green lines*. The *Unreachable* trajectory segments are denoted as *red lines*.

4. *Cell reachability evaluation* (fig. 6.16d) - the evaluation of *cells* reachability is going according to (eq. 6.92). The *Reachable cells* are those which *contains* at least one *Reachable Trajectory Segment*.



(a) Obstacle detection.

(b) Uncertainty assessment.

(c) Trajectories reachability evaluation.

(d) Cell reachibality evaluation.

Figure 6.16: Significant steps of *Avoidance grid run* (inner loop).

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. 6.3). The following properties are known prior the *trajectory* selection:

1. *Reachability* for each $cell_{i,j,k}$ (eq. 6.92).

2. *Reachability* for each $Trajectory(\circ)$ (eq. 6.91).

3. *Free Space* as non-empty set of *cells* in *Avoidance Grid* (eq. 6.96), with *Reachable Space* (eq. 6.97).

4. *Goal Waypoint* $\mathcal{WP}_G$ from *Mission Control Run* (sec. 6.6.2).

[2] Avoidance Run Function Implementation: `RuleEngine/MissionControl/MissionControl.m::findBestPath(avoidanceGrid)`

Algorithm 6.6: Find best *Path* in *Avoidance Grid*

Input : Cell[] reachable (eq. 6.97), Waypoint goal, AvoidanceGrid(t_i) grid
Output: Trajectory avoidancePath, Error message

```
# Initialization & Reachability test;
avoidancePath = ∅;
if reachable == ∅ then
    return [avoidancePath,"No path available, empty Reach Set"]
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
        return [avoidancePath,"Waypoint not Reachable"]
    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]
```

The *Algorithm* (alg. 6.6) is based on *shortest path* search. Navigation is trying to reach *goal waypoint*; therefore it tries to shorten the distance between *final trajectory 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 the closest cell considered as an *outer cell* to *goal waypoint*.

The *Avoidance Path* selection is simple lowest cost selection of $Trajectory \in \text{cell}_{i,j,k}$.

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*.

Space Assessment Example: For better understanding, there is the following example of *space assessment* and *Best Path Selection*.

The *UAS* (blue plane) is following a *mission plan* in open space. Then there is a detection of a *collision situation* (fig. 6.17). The *Obstacle* is detected in the *top-right* Avoidance Grid corner.

The *LiDAR hits* are denoted as red filled circles. The *Avoidance Grid* space is constrained by the black dashed line. The *Avoidance Grid* is separated into five layers going from top to *bottom*. The *Reach Set* is projected as a set of *Trajectories* with colorization.

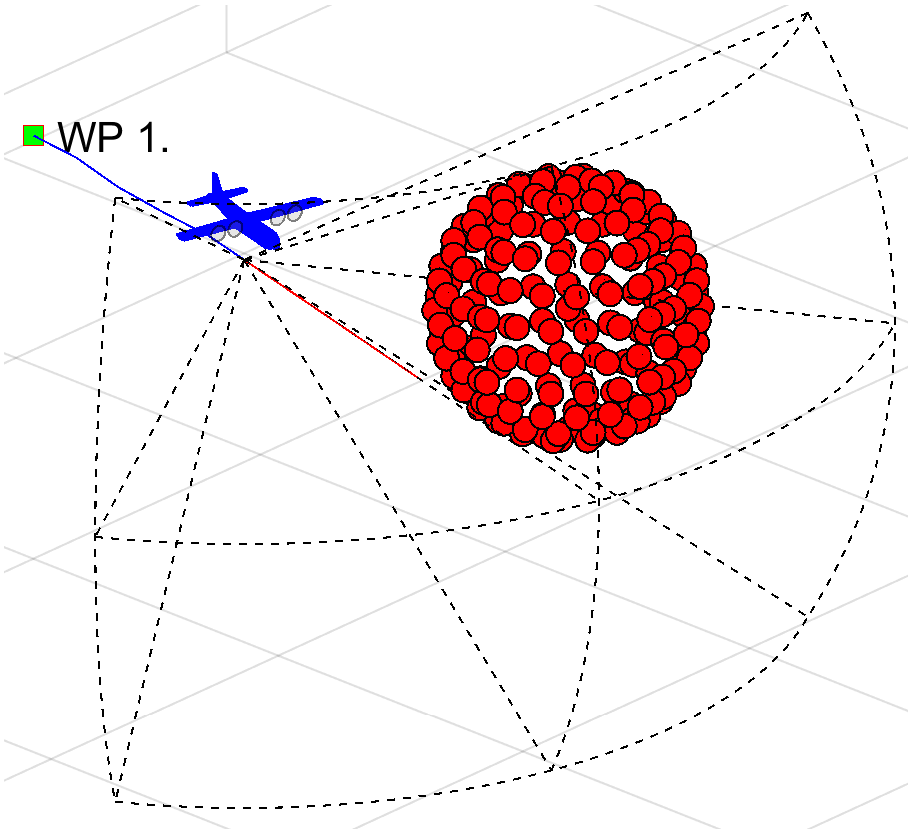


Figure 6.17: Example: The situation to be evaluated by *Avoidance Run*.

Visibility Assessment: The visibility assessment (fig. 6.18) divides the *Avoidance Grid* into two

1. *Visible space* (blue filled cells) is space *through* 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.

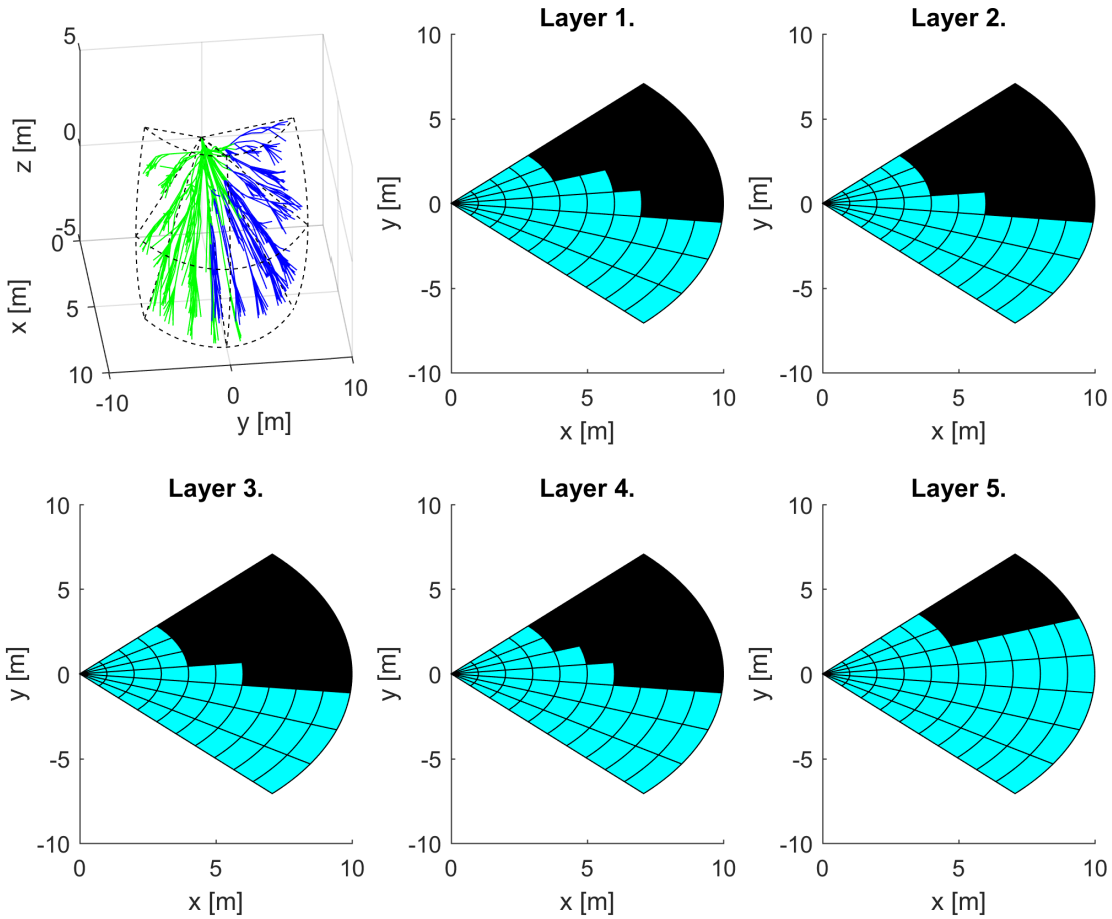


Figure 6.18: Example: The *Visibility* evaluation by *Avoidance Run*.

Reachability Assessment: For Each trajectory, the *Reachability* is assessed (fig. 6.19). The *Obstacle Space* and *Uncertain Space* are rendering *reachability*, effectively separating *trajectories* into two categories:

1. *Unreachable Trajectories* (red lines) - there is at least one trajectory segment leading through *Obstacle* or *Uncertain* space.

2. *Reachable Trajectories* (green lines) - all trajectory segments are lying in *Free* space.

Cells in Avoidance grid are divided in a similar matter, depending on the count of *reachable trajectories* passing through them:

1. *Unreachable Cells* (red fill) - there is no trajectory through *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*.

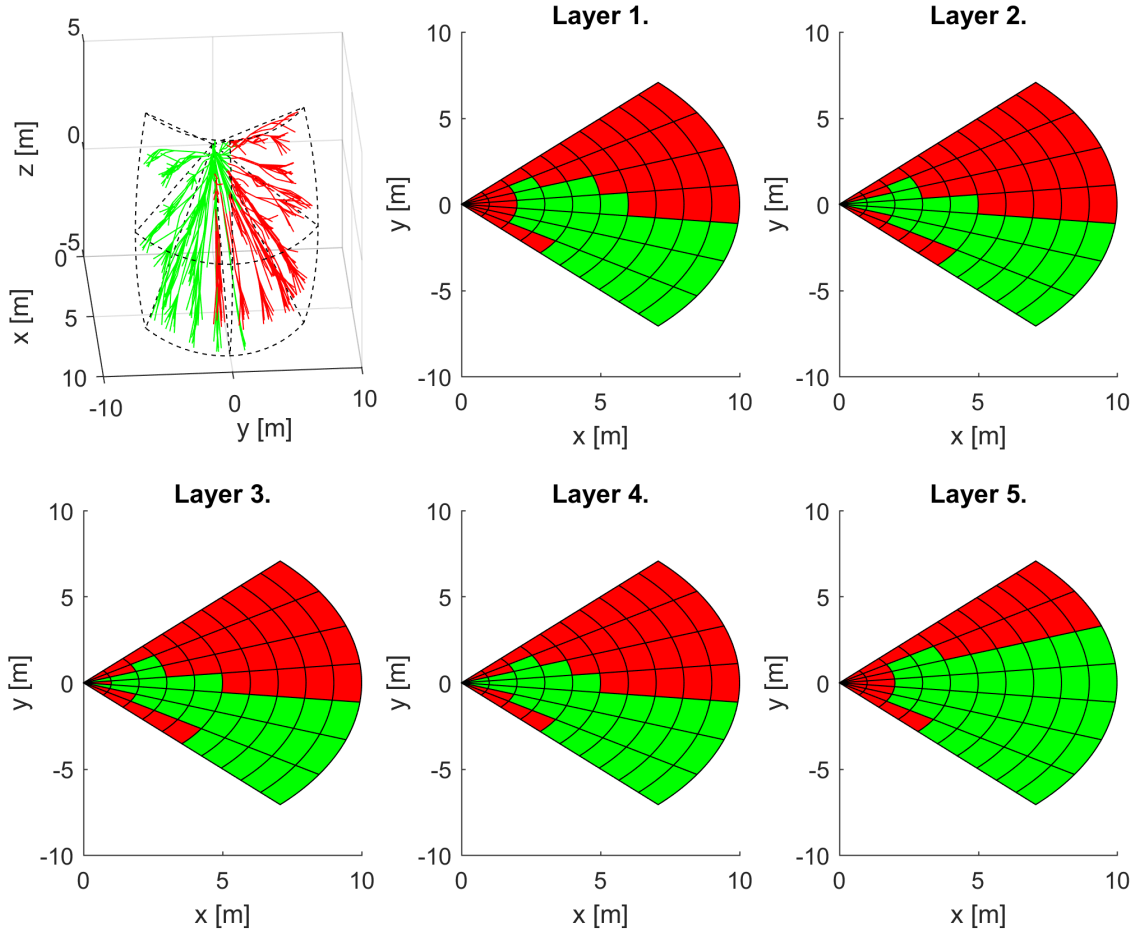


Figure 6.19: Example: The *Reachability* evaluation by *Avoidance Run*.

Note. The *best avoidance path* is selected form *reachable outer cells* (green fill in fig. 6.19), depending on *goal waypoint* according to (alg. 6.6).

6.6.2 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 a *sub-system* for long term *trajectory tracking*. The *Avoidance Grid Run* (sec. 6.6.1) 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 the *distance* of one *Avoidance Grid*. The *UAS* is controlled via *Movement Automaton*. The *Movements* which are in *Movement Buffer* can be replaced with other movements. This feature of *Movement Automaton* is called *Movement Chaining* (eq. **??**).

To join the multiple *Avoidance Grids* paths following terminology needs to be established (fig. 6.20a):

1. *Goal* (Selecting Goal of Navigation) - the point where UAS want to get in the global coordinate frame. The selection needs to be defined.

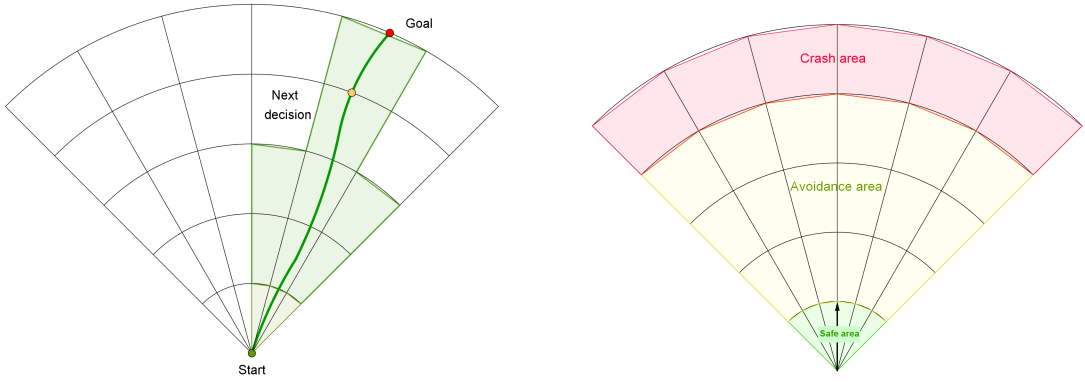
2. *Next Decision* - the point when the next *Avoidance Grid Run* is applied. The outline of events and triggers is required. The *decision* will be made in the *next decision time* t_{i+1}.

The *Avoidance Grid* from *UAS* viewpoint can be separated into following zones (fig. 6.20b):

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. 6.4.2) 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 6.20: Definitions for *Mission Control Run* (outer loop).

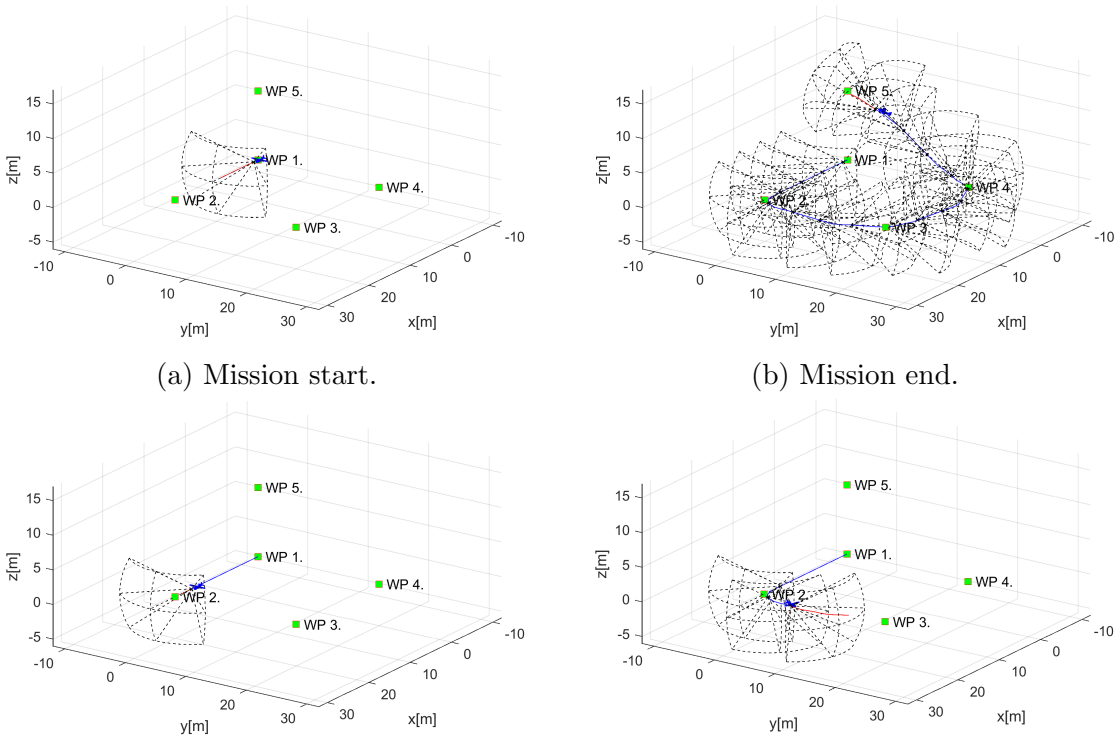
Joining *Avoidance Grid Runs* (fig. 6.21) example portrays *Avoidance Grid Runs* invoked on various *Decision Points* to achieve *Navigation* functionality. The UAS (blue plane) is flying Mission (green numbered waypoints). The *Avoidance Grid* boundary (black dashed line) for each *Decision Point* (UAS position at time t_i). Following the example of *Navigation* (fig. 6.22) run is shown:

1. *Mission Start* (fig. 6.21a) - 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. 6.21b) - UAS have reached the *last waypoint*. All *Avoidance Grid* boundaries (black dashed line) for all *runs* are drawn along flown trajectory.

3. *Waypoint Reach* (fig. 6.21c) - 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. 6.21d) - 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 6.21: Joining multiple *Avoidance Grid Runs* to achieve Navigation.

General Concept: [3] The *General Concept* is taken from [35, 36], consisting of following main modules:

1. *Navigation Loop* - module responsible for *Navigation* providing *Goal Waypoint*.
2. *Data Fusion* (background in sec. 6.5.4) - module responsible for *Surveillance Data Feed*.
3. *Situation Assessment* - module responsible for *UAS Safety Evaluation*.
4. *Avoidance Run* (background in sec. 6.6.1) responsible for *Avoidance Path* selection.

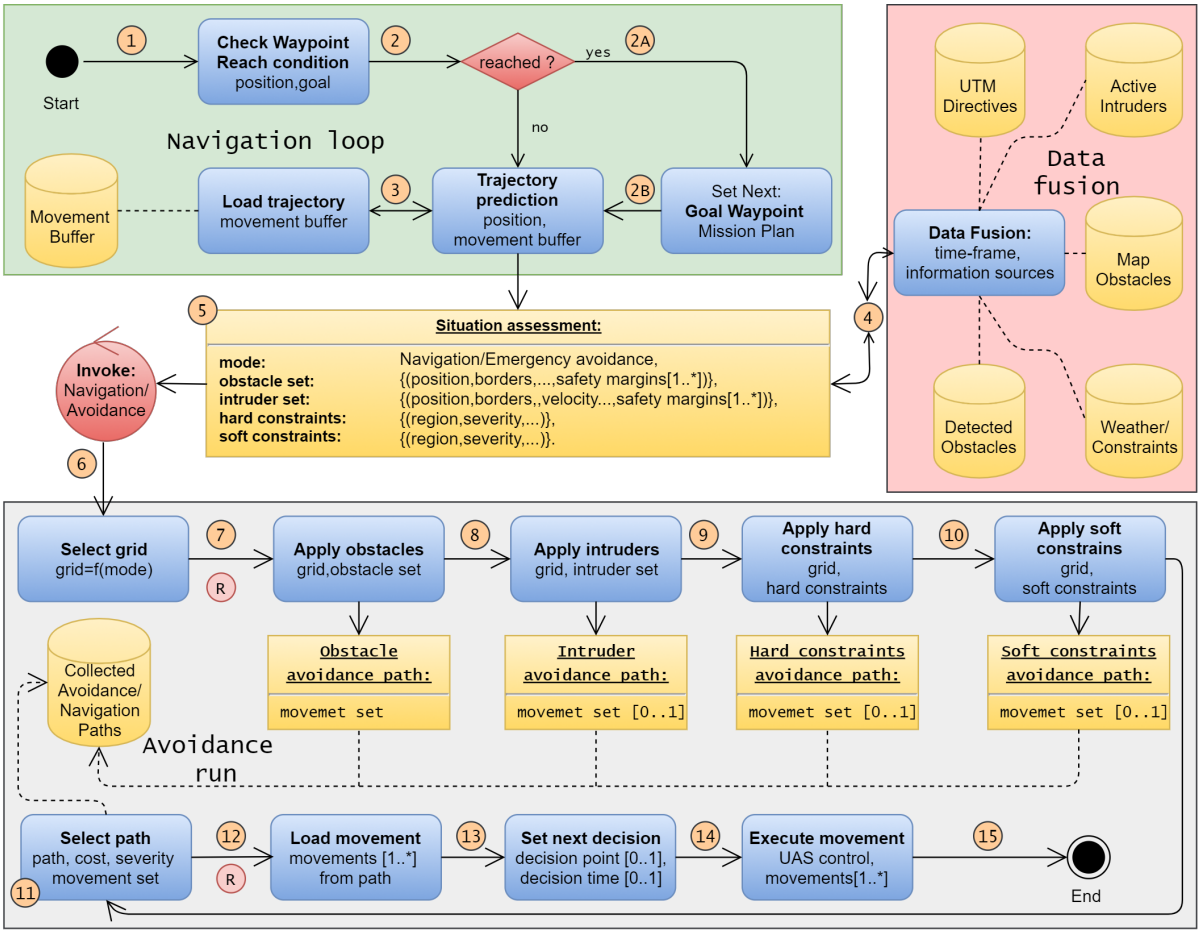


Figure 6.22: Mission control run activity diagram.

The main changes to *Navigation architecture* are given in *Mission Control Run* activity diagram (fig. 6.22):

1. *Situation Assessment* - added event-based mode switching control.
2. *Avoidance Run* - added hierarchical evaluation for *Avoidance Path* selection; This is responsible for 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 the following modes are selected in *Avoidance Run*:

1. *Navigation Mode* - the *UAS* is navigating through *Airspace* following *cost-effective patterns* and obeying *Airspace Authority* (UTM). The *Navigation Grid* is an instance of *Avoidance Grid* (sec. 6.3) with initialized *Navigation Reach Set* (ex. *Turn-Minimizing Reach Set Approximation* (sec. 6.4.5)).

2. *Emergency Avoidance Mode* - the *UAS* is *threatened* by obstacle, intruder, hard constraint or *soft constraint*, the *UAS* is navigating through *Airspace* following *safe avoidance patterns* and *minimizing the impact* of possible damages. The *Avoidance Grid* is a term used for *Emergency Avoidance Mode*. The *Avoidance Reach Set Approximation* is initialized in *Avoidance Grid* (ex. *Coverage-Maximizing Reach Set Approximation* (sec. 6.4.4))

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* share the space portioning pattern; therefore the *Data Fusion* (sec. 6.5.4) 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. 6.2.2) controlled by *Movement Automaton Implementation* (sec. 6.2.3) *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 the *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* are 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. **??**). Using control scheme (fig. 6.2) with given *Sensors*:

$$Sensors = \{LiDAR, ADS - B\} \tag{6.99}$$

The sensors obstacle assessment into avoidance grid is outlined for static obstacles in (sec. 6.5.1) and for moving obstacles in (sec. 6.5.2.)

The *Data Fusion Procedure* is given as follow:

$$DataFusion = \{RatingBasedDataFusion \quad (sec.6.5.4)\} \tag{6.100}$$

Then the *UAS system* (sec. 6.2.2) with *Movement Automaton Implementation* (sec. 6.2.3) with empty movement buffer:

$$MovementBuffer = \{\} \tag{6.101}$$

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:

$$\begin{aligned} AvoidanceGrid(0) &= \{UAS.position(0), AvoidanceReachSet(UAS.ReachSet)\} \\ NavigationGrid(0) &= \{UAS.position(0), NavigationReachSet(UAS.ReachSet)\} \\ OperationMode &= Navigation \end{aligned} \tag{6.102}$$

The *Mission* is set up as a set of *ordered waypoints*. The *initial goal waypoint* is *first waypoint*. The initialization is summarized like follow:

$$\begin{aligned} Mission &= \{Waypoint_1 \dots Waypoint_n\} \\ GoalWaypoint &= Mission.waypoint_1 \\ LastWaypoint &= Mission.waypoint_n \end{aligned} \tag{6.103}$$

The *actual threats* are set as empty sets for *decision time* $t_i = 0$:

$$obstacles = \{\}, intruders = \{\}, hardConstraints = \{\}, softConstraints = \{\} \tag{6.104}$$

Navigation Loop (1st-3rd step): The purpose of *Navigation Loop* is to select proper *Goal Waypoint* from *Mission* (sec. **??**). If *last waypoint* have been reached the *Landing Procedure* will be initiated and *Mission Control Run* Ends.

First, start with the definition of *waypoint reach condition* (def. 12) and *Unreachable waypoint* (def. 13).

Definition 12. *Waypoint Reach Condition for* current *decision time* t_i *for* UAS *position and current* Goal Waypoint *is satisfied only if:*

$$distance(UAS.position(t_i), GoalWaypoint(t_i)) \leq 2 \times \max\{length(movement) : \forall movement \in MovementSet\} \quad (6.105)$$

Note. The movements in our solution have a *uniform length* of *1 m* (tab. 6.1, 6.2), therefore the waypoint reach condition is satisfied when the *distance to goal waypoint* is lesser than 2 m. The maximal movement length has an impact on *navigation/avoidance* precision.

Definition 13. *Unreachable Waypoint. The* Goal Waypoint *evaluates as unreachable in decision time* t_i *when* Avoidance Grid Run *(alg. 6.6) cannot 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*:*

$$Grid(t_i).range \geq distance(UAS.position(t_i), GoalWaypoint(t_i)) \quad (6.106)$$

and following condition is satisfied:

$$\forall cell_{i,j,k} \in Grid(t_i) \; \nexists cell_{i,j,k}.Reachable == true \wedge \ldots \cdots \wedge distance(cell_{i,j,k}, GoalWaypoint(t_i)) \leq \ldots \cdots \leq 2 \times \max\{length(movement) : \forall movement \in MovementSet\} \quad (6.107)$$

The Goal Waypoint *is unreachable.*

Then the *Navigation Loop* is invoked every *decision time* t_i, *Mission Control Run* (fig. 6.22), it is described as a sequence of the following steps:

1st Check Waypoint Reach Condition - the *UAS position* for given a *time frame* t_i is checked under condition (eq. 6.105). If the condition is met continue with 2nd step otherwise continue with 3rd step.

2nd Set Next Waypoint - until the 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. **??**):

$$PredictedTrajectory = Trajectory(state = UAS.state(t_i), buffer = futureMovements)$$

The *Predicted Trajectory* is used in 5th step *Situation Assessment*.

Data Fusion (4th step) The *Data Fusion* (sec. 6.5.4) in this context is *Threat Sets* preparation for *Avoidance Run*. It depends on the values of *Boolean values* defined in (tab. 6.3) for *threat* classification.

Note. Avoidance Grid's Data fusion (sec. 6.5.4) is run in the 7th- 10th step (fig. 6.22).

The *static obstacles* source is from *LiDAR* scan received at least at the beginning of current *decision frame* t_i:

$$obstacles = LiDAR.scan(UAS.position(t_i))$$

The *intruder's* 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 to 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 the search. The sets of hard and soft constraints are obtained in the 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 the next step.

Invoke Navigation/Avoidance based on Situation Assessment (5th-6th step): The *deciding events* depending on *Trajectory Prediction* (3^{rd} step) and *Data Fusion* (4^{th} step) (fig. 6.22) are the 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 the new path.

b. *Waypoint Reached* (2nd step) - the *Navigation Loop* run is forced to set goal *Goal Waypoint*. If the *last waypoint* from *Mission* (sec. **??**) the *Landing Procedure* is enforced.

c. *Waypoint Unreachable* - this type of event is very situations based. The *Waypoint Reachability* (assumption. **??**) has not been relaxed; therefore this event is not properly handled in approach. The *implementation* considers *selecting next waypoint in the mission* as a goal waypoint of the *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. 6.7.6), 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* (eq. 6.50) 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 (app. **??**)) *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 (def. 10)) *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 the 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. 6.96) for all cells.

Avoidance Run (7th-15th step): The *Avoidance Run* goal is to obtain *Path* represented as *Trajectory*(state(t_{+1},MovementBuffer)) (eq. **??**) from *Navigation/Avoidance Grid* and associated *Navigation/Avoidance Reach Set Approximation.*

If the *Operation Mode* is set as *Navigation Mode*, the algorithm continues with the 11th step. Otherwise, the *Avoidance Grid Space Assessment* is run multiple times to obtain $Reachable(t_{i+1})$ (eq. 6.97). The *Threat Data* obtained from the 4th step are used.

7th Apply Obstacles - The *Space assessment* (tab. 6.3) 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. 6.3) 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. 6.3) 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. 6.3) 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 a standard *Find Best Path* (alg. 6.6) procedure. The *Navigation Reach Set Approximation* can be constrained by *Rule Engine* (fig. 6.29).

The *Avoidance Path* for *Emergency Avoidance Mode* is selected from *Collected Avoidance Paths* with the 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 the 12th step.

Note. The *Waypoint Reachability* (assumption **??**) is weakened to the point that it is necessary for the waypoint to be *Reachable* only in static obstacle environment. The *Constrained* and *Occupied* spaces are shrunk in the 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 movement is 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 the point before *UAS* enters into *Crash Zone* (fig. 6.20b) 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 the *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.

Decision Frame: The *mission control run* (fig. 6.22) describes the overall process in *sequence*. The *orchestration overview* is given in (fig. 6.23).

The key idea is to explain what happens in one *decision frame*. The *mission control run* is implemented as multi-thread application which sends the signals between threads. Each thread is the semi-independent process with forced synchronization on *decision frame switch*.
The notable threads and their roles & responsibilities are summarized like follow:

1. *Sensor Fusion* - responsible for processing real-time sensor array (sec. **??**). The output is a partial known world assessment (sec. **??**). *Obstacle detection* and *intruder detection* events can be risen by this thread.

2. *Data Fusion* - responsible for enhancing data from *sensor fusion* by mixing data originating from *information sources* (sec. **??**). The information sources used in this work contains constraints originating from *geo-fencing*, *weather*, *airspace restrictions*. This thread is delayed by *sensor fusion*. A *data fusion procedure* strongly depends on the *operational space context* (controlled/non-controlled airspace). The output of *data fusion* is full *known world assessment* (sec. **??**, 6.5.4). The *UTM-related* and *constraint related* events can arise from *data fusion*.

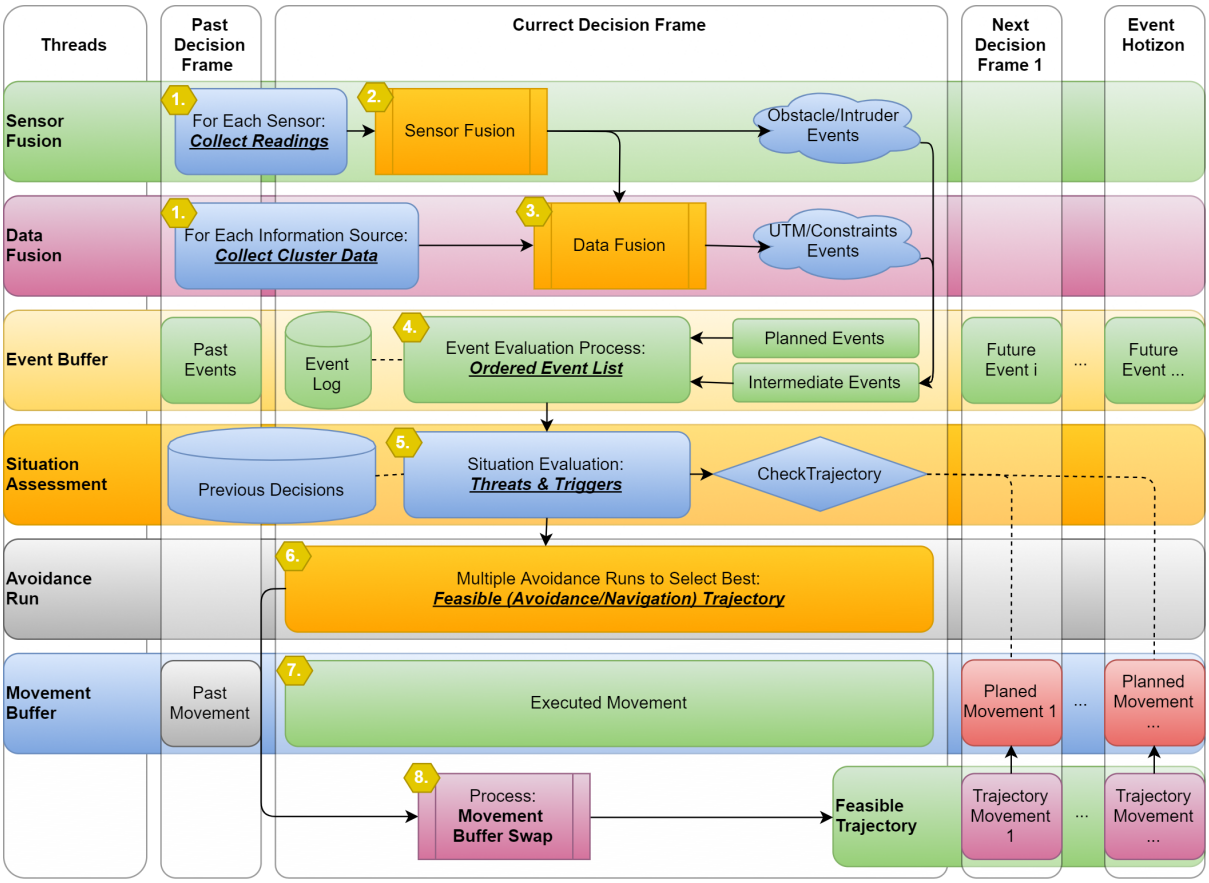


Figure 6.23: Mission control orchestration diagram.

3. *Event Buffer* - special data structure to store, raise, handle, prioritize events raised by other threads.

   The *implemented events* are listed in the 5^{th}-6^{th} step of *mission control run*. The events can be categorized like follow:

   a. *Planned events* - raised in previous decision frames to be executed in actual or future *decision frame*.

   b. *Intermediate events* - raised in *actual decision frame* by other threads to be solved intermediate.

   The event buffer thread executes following event-related activities:

   a. *Storing* - the *events* are stored in the *event log*. The trace is useful for process and rules fine-tuning.

   b. *Raising* - the combination of events (multiple avoidance events) (example sec. **??**) can trigger additional avoidance behavior in the form of combined-event.

   c. *Handling* - the events are handled by invoking the *situation assessment* or by rule engine invocation (sec. 6.8.1).

   d. *Prioritizing* - the multiple events can rise during one *decision frame*. Some events cannot be merged and need to have proper prioritization before handling, like the *obstacle detection* events before *intruder detection event*.

4. *Situation Assessment* - invoked by *event buffer* to assess the situation, responsible for proper *avoidance run* (sec. 6.6.1) dataset preparation and invocation. The main responsibility is to check *planned trajectory feasibility* stored in *movement buffer* as *planned movements*.

5. *Avoidance Run* - invoked by *necessity to plan trajectory* originating from *event buffer* or *situation assessment* threads. The avoidance run produces one or multiple *avoidance/navigation* feasible trajectories according to the 7^{th}-11^{th} step of *mission control run*.

6. *Movement Buffer* - represents *movement automaton implementation* (sec. 6.2.3). The movement automaton consumes *movement automaton buffer* each decision frame contains exactly one *movement*. The movements can be viewed as:

   a. *Past movements* - already executed movements in *past decision frames*.

   b. *Executed movement* - actually executed movement in the current decision frame, this movement cannot be changed.

   c. *Future movements* - future planned movements to be executed after *current decision frame* expires. These movements outline planned trajectory (predictor mode sec. **??**).

7. *Feasible Trajectory* - consists of *future planned movements* taking place directly after the *correct decision frame*. If its necessary, the planned trajectory in movement buffer is no longer feasible, the planned movements will throw away and replaced by *trajectory movements*.

The *roles & responsibilities* of each thread have been explained to outline their orchestration and roles in *mission control run* (fig. 6.22). The numbered steps in (fig. 6.23) shows the threads orchestration in the following manner:

1. *Sensor & Data fusion data set preparation/collection* - the sensor readings are collected through multiple past and over current *decision frame*. Each sensor reading is filtered and processed according to best practices.

   The raw information from various data sources is loaded for relevant space clusters. The relevant space clusters are determined based on *UAS expected position*.

2. *Sensor fusion* - the readings from sensors are preprocessed according to (sec. 6.5.1, 6.5.2).

3. *Data fusion* - the information sources are preprocessed according to (sec. 6.5.1, 6.5.2).

4. *Event evaluation process* - the events are evaluated, if there is any triggering event (5^{th}-6^{th} mission control run steps) the situation evaluation process is called.

5. *Situation evaluation process* - the situation is evaluated according to 5^{th}-6^{th} mission control run steps.

6. *Feasible trajectory selection process* - from collected *navigation/avoidance trajectories* (7^{th}-10^{th} mission control run steps). If there are more feasible trajectories (increasing threat) the one compliant with most of the threats is selected.

7. *Movement execution* - the movement for the *current decision frame* is being executed.

8. *Movement buffer swap* - if there is a new *feasible trajectory* the future movements for next decision frames are flushed away. The movement buffer is then filled with *feasible trajectory movements*.

   Note. This step impacts the duration of future *decision frames*.

6.6.3 Computation Complexity

Introduction: The *Computation Complexity* one mission control run assessment is necessary to identify the strong and weak points of approach. Let us get through modules to assess notable calculations/algorithms complexity on high abstraction level.

Navigation Loop: On the navigation loop, the *waypoint reach condition* (eq. 6.105) is checked, this is a unitary operation with worst complexity $\mathcal{O}(1)$. The selection process of the next *Goal Waypoint* can get through 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* depends on clustering. The *Airspace Clustering* is considered as static for our setup. Therefore the *count of active airspace clusters* has the 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 a *hard non-polynomial problem* [37]. Usually, all information sources and sensor have *polynomial complexity* of processing. The *controlled airspace clusters* are usually set for a very long period. 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 *eight* 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 the most critical part of *Mission Control Run* because of *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^{th} to the 10^{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* (app. **??**) is going through the maximum of *layers count* cells.

The *body volume intersection model* (app. **??**) can check the *simple intersection condition* overall *Avoidance Grid* in the worst case; therefore complexity for this check is bounded by a *count of cells*.

The *Maneuverability Uncertainty Intersection* (app. **??**) can hit all cells in *Avoidance Grid*. The calculation complexity boundary is exponential depending on the *horizontal/vertical* spread in $[rad]$. The *intersection* implementation was done *ad-hoc*. The impact of *intersection application* is visible only when there are more than *four* concurrence intruders (fig. **??**).
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 the 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 a study [27], the selected algorithm *Shamos-Hoey* [28]. The *calculation complexity* boundary is given like follow:

`Hard Constraints Intersection:`

$$\mathcal{O}(|cells| \times |hardConstraints| \times \max|constraintPoints|^2)$$

`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^{th} to the 10^{th} step. All ratings (tab. 6.3) expect $Reachability(cell_{ij,k})$ and $Reachability(Trajectory)$ are calculated. The *calculation complexity* boundary for one *reachability rating* is $\mathcal{O}(1)$. (eq. 6.91, 6.92). The *Recalculate Reachability* operation applied $4\times$ have maximal *complexity* boundary given as follow:

$$\texttt{Recalculate Reachability:} \quad \mathcal{O}(4 \times (|nodes| + |cells|))$$

Each time at the end of in 7^{th} to the 10^{th} step the *Avoidance Path is Selected*. The *Worst Case* (expected) scenario is to *select* four paths for each *threat* application. The algorithm for *best path selection* (alg. 6.6) iterates overall *cells* in avoidance grid and over all *trajectories* passing through 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. **??**).

Note. The calculation of *Reach Set* is eliminated by pre-calculation for *state range* [2].

6.7 UTM Prototype Implementation

The *Traffic Management* for UAS is based on existing Air Traffic Management System for manned aviation [32]. The controlled airspace segments are *static* and have one *authority for one zone* principle. The dynamic zones have been proposed in [38]. However, it will be omitted for *simplification purpose*. The necessity for *UAS integration* into *National Airspace* has been outlined in [39].

The latest *Airbus blueprint* [40] 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. 6.7.1) - centralized ATM-like authority over airspace cluster.
2. *Handling Standard Collision Situations* - head-on approach (sec. 6.7.2), converging situation (sec. 6.7.3), overtake (sec. 6.7.4).
3. *Position Notification* (sec. 6.7.5) - position notification design.
4. *Collision Case* (sec. 6.7.6) - calculation and handling of *collision situations*.

The additional material can be found in:

1. *Cooperative Conflict Resolution* (app. **??**) - the model used for conflict resolution in *controlled airspace*.
2. *Non-Cooperative Conflict Resolution* (app. **??**) - the model used for conflict resolution in *non-controlled* airspace and *emergency avoidance*.
3. *Weather Case* (app. **??**) - definition and handling of *weather hazards*.

6.7.1 UTM Architecture

UTM Concept is based on *asynchronous event-based control* [41]. *Event* in *controlled airspace* is handled in the form of *cases* [42]. There are following *event sources*:

1. *Weather Information Service* (from [43]) - used to create *weather case* (tab. **??**).
2. *Position Notification from UAS systems* (tab. 6.4) - used to create *collision cases* (new functionality) (tab. 6.6).

Decision Frame (eq. 6.108). The *UTM* is operating in discrete decision frames which are starting on current *decision time* and ending at next *decision time*:

$$decisionFrame_i = [decisionTime_i, decisionTime_{i+1}[, \quad i \in 1, \ldots, k, k \in \mathbb{N}^+ \qquad (6.108)$$

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, controlled UAS systems fulfill the state of affairs for directives and mandates.

3. *Acknowledgment* - the closing phase, when UTM assess and acknowledges the performance of controlled UAS systems.

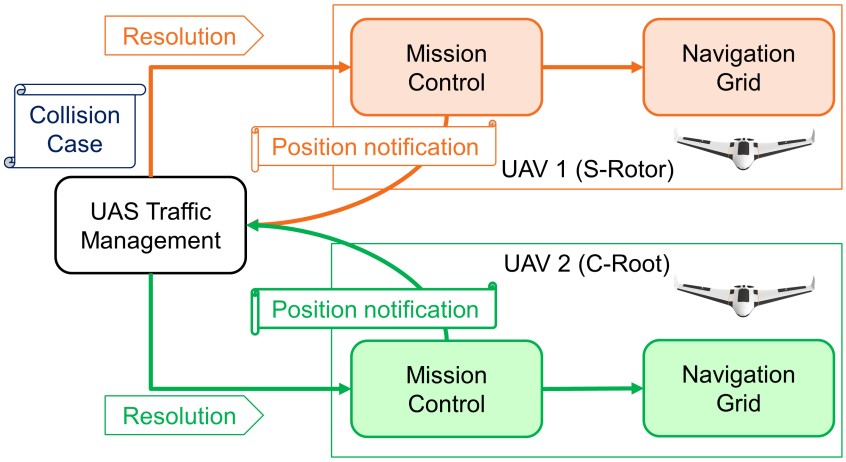


Figure 6.24: UAS Traffic Management (UTM) architecture overview.

Architecture (fig. 6.24). 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. 6.4).

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.

6.7.2 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 [44], 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."

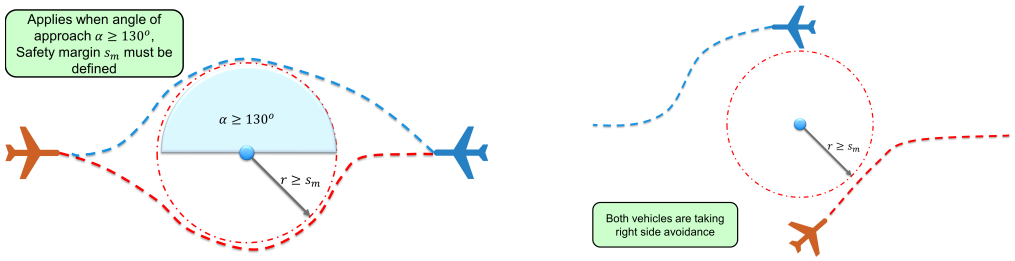
IFR: The *Instrument Flight Rules* in annex 2. [44] and 11. [45] are defining the boundaries and events for success full *Head-on resolution* in larger detail.

The parameter values are useless due to the UAS scaling factor; the following parameters can be used in UTM:

1. The *angle of approach* $\geq 130°$ - the minimal planar angle between aircraft positions and expected collision point is in the 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 the value of safety margin.



(a) Detection. (b) Resolution/Closing.

Figure 6.25: Head-on approach detection/resolution/Closing

Triggering Events: The *head-on approach* (fig. 6.25) *triggering events* are the following:

1. *Detection* (fig. 6.25a) - the *collision case* is open when *collision point* with the respective angle of approach is detected. This must happen until the *point of no return* is achieved.

2. *Resolution* (fig. 6.25b) - the *virtual* roundabout is enforced until the closing condition is met.

3. *Closing* (fig. 6.25b) - based on the condition that all vehicles are heading away from *collision point* and their mutual heading is neutral or opposite.

Virtual roundabout: The *flight levels* can be abstracted as the *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 phenomena.

3. *Enforced velocity* - all attendants at *virtual roundabout* keeps the same velocity. It helps to keep constant mutual distances.

6.7.3 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 [44]. The rule is different from *Head-on Approach* (sec. 6.7.2) because multiple roles are depending on the relative aircraft position:

1. *Avoiding Aircraft* - there is an aircraft on the relative right side (blue).

2. *Right Of the Way (ROA) Aircraft* - there is an aircraft on the relative left side (red).

The *avoiding aircraft* should take the *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 both *aircraft* belong to the same *maneuverability class* [44].

IFR: The *Instrument Flight Rules* in annex 2. [44] and 11. [45] are defining *converging maneuver* in detail.

The *parameters* from a *head-on approach* can be reused:

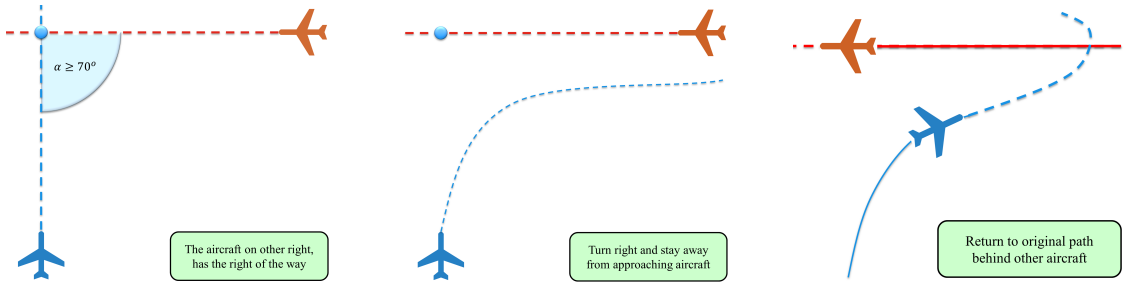
1. $70° \leq$ the *Angle of Approach* $< 130°$ - the minimal planar angle between aircraft position and expected collision point is in the interval $[70°, 130°[$.

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 the value of *Safety Margin*.

Note. The lesser *angle of approach* induces stronger wake turbulence impact on avoiding aircraft. This results in an increase of *safety margin*.

The *wake turbulence* is represented as a 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 6.26: Converging maneuver Detection/Resolution/Closing

Triggering Events: The *converging maneuver* (fig. 6.26) *triggering events* are the following:

1. *Detection* (fig. 6.26a) - 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 the *detection range*.

2. *Resolution* (fig. 6.26b) - the *Right Of the Way aircraft* (red) stays at the original course. The *avoiding aircraft* (blue) follows the *parallel* to another *plane*. The distance of *avoiding plane* to *other plane trajectory* is greater or equal to *safety margin*.

3. *Closing* (fig. 6.26c) - when both planes have an 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 *another plane* (red) at greater or equal to *safety margin*.

6.7.4 Handling Overtake Maneuver

Goal: Identify *required parameters* sufficient for automatic solution of *Overtake Maneuver*

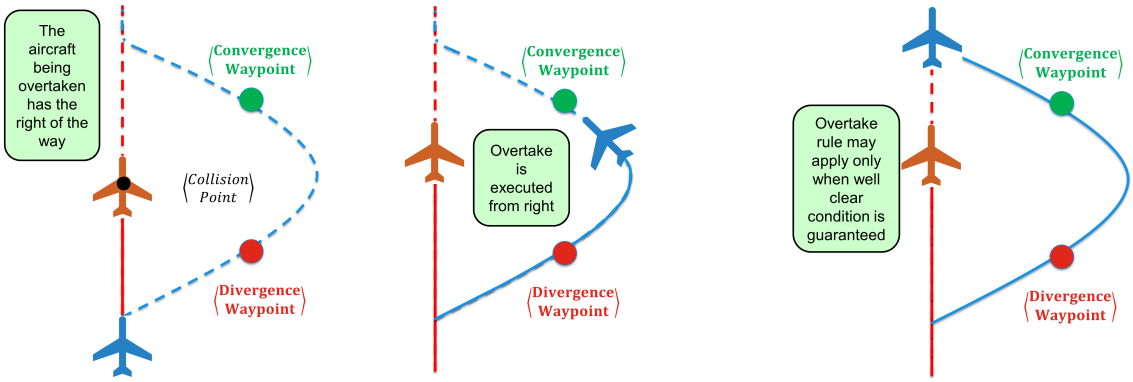
VFR: The *Visual Flight Rules* (VFR) are specified in annex 2 [44]. The rule states that faster air traffic attendant may overtake slower one, from right side keeping sufficient distance (*safety margin*). There are two forced roles:

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 the same maneuverability class [44]. 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*.



(a) Detection. (b) Resolution. (c) Closing.

Figure 6.27: Overtake maneuver Detection/Resolution/Closing

IFR: The *Instrument Fight Rules* in annex 2. [44] and 11. [45] are defining the converging manual in detail:

1. $0° \leq$ the *Angle of Approach* $< 130°$ - the minimal planar angle between aircraft position and expected collision point is in the 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.

Triggering events:

1. *Detection* (fig. 6.27a) - 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 taking *sharp right side to overtake* the *Maneuver starts*, otherwise *overtaking aircraft* (blue slows down) and keeps at least *safety margin distance* to avoid *wake turbulence*.
2. *Resolution* (fig. 6.27b) - *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. 6.27c) - 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 the original waypoint.

Constant Cruising Speed: Most of the 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 [46, 47, 48] in multiple air-traffic models.

6.7.5 Position Notification Implementation

Motivation: The *position notification* (tab. 6.4) is designed for further *collision case resolution* (sec. 6.7.6). 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.

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 a *main flight level* where *aircraft* belong physically. There is a *passing flight level* form which/to which is aircraft emerging [32].

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* [44] - the aircraft with impaired capability switched to emergency mode. The emergency mode is usually acknowledged by the authority in controlled airspace.

2. *Balloon* (manned) [44] - the aircraft with *altitude* control and very slow dynamics implying very low maneuverability.

3. *Glider* (manned) [44] - the aircraft with *full control* but without own *propulsion*. The overall *maneuverability* is good, but the *velocity* changes are impossible with sufficient flexibility.

[4] ADS-B versions and message containment: `https://mode-s.org/decode/adsb/version.html`.

4. *Aerial towing* (manned) [44] - 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) [44] - the airship have *own propulsion* and full maneuverability, the constraint is low acceleration/deceleration and huge turning radius.

6. *Other manned aviation* [44] - containing all vehicles with the required level of *airworthiness* for given operational *altitude*. They usually have required maneuverability.

7. *UAS Autonomous* (proposed) [49] - containing all autonomous UAS, the lower flexibility is expected at the beginning of integration.

8. *Remotely Piloted Aerial System (RPAS)* (proposed) [49] - has lesser priority due to 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 always be considered as a 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 made by solving *pursuit problem* using *Reach Sets* [50, 51].
The *Maneuverability categorization* is based on *original aircraft priority categorization* [44] 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 do not have any type of propulsion, and horizontal movements follow 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 of doing on-spot-turn. The typical representative is *quad-rotor copter*.

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 a 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 is 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 a head-on approach.
converging	minimal distance from other similar maneuverability class aircraft in case of the 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 6.4: Time-stamped *position notification* structure.

There are other aspects like *minimal required* acceleration/deceleration/turn ratio to operate in a 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 the *situation*. There is also a *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 the list of supported maneuvers; there is converging (sec. 6.7.3), head-on (sec. 6.7.2), overtake (sec. 6.7.4) safety margins.

6.7.6 Collision Case Implementation

Collision Case Purpose: There is a need for detection and tracking of possible *controlled airspace traffic attendants* collisions. The presented *collision case structure* (tab. 6.6) is a 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 the *life-cycle*.

3. *Life cycle* gives outlook on how collision case data are handled through a longer period, notably: *Opening*, *collision point handling*, *safety margin handling*, and, *Closure*.

4. *Merge procedure for multiple cases in a single cluster* - the naive *merge procedure* to solve *multiple collision cases* via the *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 the form of *Resolutions* and *Mandates*.

A *Resolution* is an order from the *UTM* authority which is followed by subjected UAS. The *subjected UAS* can determine own behavior to some extent. When there is an emerging threat or another destructive event, like a new non-cooperative adversary, the UAS is allowed to broke *resolution*.

A *Mandate* is an order from the *UTM* authority which cannot be broken at any cost. The example of the *mandate*: UAS is flying in the airspace, the passenger in distress needs it to safely land. The UAS must obey mandate even at the event of own destruction.

Threat Severity Evaluation: The threat severity evaluation is omitted partially, 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 in future when the anticipated situations are predicted.

1st The *position* and *orientation* are adjusted according to the *mission plan*. Our implementation uses *Movement Automaton* as a predictor:

$$\begin{aligned} adjustedPosition &= Position(Trajectory(notifiedState, futureMovements)) \\ adjustedOrientation &= Orientation(Trajectory(notifiedState, futureMovements)) \end{aligned} \tag{6.109}$$

For other cases standard linear prediction can be used:

$$\begin{aligned} adjustedPosition &= notificationPosition \times notificationVelocity \times timeDifference \\ adjustedOrientation &= notificationOrientation \end{aligned} \tag{6.110}$$

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 the right of the way is forced to the 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:

$$\begin{aligned} A &= \|velocity_1\|^2 \\ B &= 2 * ({velocity_1}^T \times position_1 - {velocity_2}^T \times position2) \\ C &= 2 \times {velocity_1}^T * velocity_2 \\ D &= 2 * ({velocity_2}^T \times position_2 - {velocity_2}' \times position_1); \\ E &= \|velocity_2\|^2 ; \\ F &= \|position_1\|^2 + \|position_2\|^2 ; \end{aligned} \tag{6.111}$$

Then the projection parameters can be calculated:

$$\begin{aligned} time &= \frac{-B - D}{2 \times A - 2 \times C + 2 \times E} \\ destination_i &= position_i + velocity_i \times time, \quad i \in \{1, 2\} \\ collisionPoint &= \frac{destination_1 + destination_2}{2} \\ collisionDistance &= \|destination_1 - destination_2\| \end{aligned} \tag{6.112}$$

If $time < 0$ the trajectories are diverging from each other (because the closest points already occurred). The procedure ends, the *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, the *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:

$$collisionMargin = \forall situation : \max \left\{ \begin{matrix} safetyMargin(situation, UAS1) \\ +safetyMargin(situation, UAS2) \end{matrix} \right\} \tag{6.113}$$

Where the *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* has the 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 that 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 initial system state and future movements. The prediction function (eq. 6.114).

$$Prediction : UAS \times state \times futureMovements \rightarrow [x, y, z, t] \in \mathbb{R}^4 \tag{6.114}$$

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. 6.114) is given as minimal distance of projected synchronized trajectories for UAS1 and UAS2. In our discrete enviroment, the *collision distance* is given as (eq. 6.115).

$$collisionDistance = \min \left\{ \|point_1 - point_2\| : \forall \begin{pmatrix} point_1 \in Prediction_1, \\ point_2, \in Prediction_2, \\ t_1 \sim t_2 \end{pmatrix} \right\} \quad (6.115)$$

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 a time when the distance is minimal. The final collision point is arithmetic mean of two positions (eq. 6.116).

$$collisionPoint = \frac{point_1 - point_2}{2} : \begin{pmatrix} point_1 \in Prediction_1, \\ point_2, \in Prediction_2, \\ t_1 \sim t_2 \text{ at minimal distance} \end{pmatrix} \quad (6.116)$$

Note. Collision point is overwritten by trajectory intersection (specific) method; the *linear intersection* is considered a *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 a center. The first step is normalization of adjusted position around collision point for both UAS:

$$normalized_i = adjustedPosition_i - collisionPoint, \quad i \in \{1, 2\} \quad (6.117)$$

Then the right-hand coordinate system internal angle calculation method is used:

$$angleOfApproach = \left| \text{atan2} \begin{pmatrix} normalized_1 \times normalized_2, \\ normalized_1 \circ normalized_2 \end{pmatrix} \right| \quad (6.118)$$

Based on the *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.6.7.2)

2. $70° \leq angleOfApproach < 130°$ - the scenario type is set as *Converging Maneuver* (sec.6.7.3)

3. $0° \leq angleOfApproach < 70°$ and *different speed* - - the scenario type is set as *Overtake Maneuver* (sec.6.7.4)

Based on *relative position* and *scenario type*, the *avoidance role* like follows:

1. *Head On Approach* enforces the following:
   a. The *avoidance role* is set as *RoundAbounting* for both UAS.
   b. None of the *UAS* does have the *Right Of the Way.*
2. *Converging Maneuver* enforces the following:
   a. *UAS* without free right side has a role set as *Converging.*
   b. *UAS* with free right side has the *Right Of the Way.*
3. *Overtake Maneuver* enforces the 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 the maximum of the sum of *default* margins or *head on* margins:

$$safetyMargin = \max \begin{Bmatrix} default(UAS1) + default(UAS2), \\ headOn(UAS_1) + headOn(UAS_2) \end{Bmatrix} \tag{6.119}$$

If *Converging Maneuver* is case type of *Head collision case* then *safety margin* is calculated based on *avoiding UAS* as the maximum of opposing UAS *default margin* and avoiding *converging margin*:

$$safetyMargin = \begin{cases} uas1.role = Converging: & \max \begin{Bmatrix} default(UAS2), \\ converging(UAS1) \end{Bmatrix} \\ uas1.role = Converging: & \max \begin{Bmatrix} default(UAS1), \\ converging(UAS2) \end{Bmatrix} \end{cases} \tag{6.120}$$

If *Overtake maneuver* is case type of *Head collision case* then *safety margin* is calculated as the maximum of *default, overtaking, overtaken* margins of both UAS:

$$safetyMargin = \max \begin{Bmatrix} default(UAS1), default(UAS2), \\ overtaken(UAS_1), overtaking(UAS_2), \\ overtaking(UAS_1), overtaken(UAS_2) \end{Bmatrix} \tag{6.121}$$

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 the *Rule engine* (sec. 6.8). The *head collision case* is first in the 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 Maneuver* is ensured by enforcing *head collision case* parameters.

Data for both attendants	
adjusted position	predicted from previous *position notifications* (6.4) data at the time of *UTM decision frame* start.
adjusted orientation	predicted from previous *position notifications* (6.4), *mission plan*, and *expected velocity*.
velocity	proclaimed velocity for given *UTM decision time frame*.
registration ID	is unique registration number issued by the local aviation authority
craft category	from *position notifications* (6.4).
maneuverability	from *position notifications* (6.4).
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* (6.4).
avoidance role	is given based on situation evaluation.
trajectory prediction	simulated based on *position notification* (6.4) and *mission plan*.

Table 6.5: Collision case structure attendant data.

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* [52]. Example of *airspace clustering* is given it [53].

The main idea is to *encapsulate multiple collision cases* into one virtual roundabout to ease *traffic load* [54]. The potential risk on *turbo roundabouts* have been outlined in [55].

There are *active collision cases* in a focused *cluster* in *controlled airspace*. The multiple collision cases can pop up at different *start times*, and they can be active for a

different *period*.

The *Collision point* is replaced with the *roundabout center* point (eq. 6.122). 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.

$$roundaboutCenter = \frac{\sum_{\in Cluster}^{\forall collisionCase} collisionCase.collisionPoint \times weight}{|collisionCase \in Cluster|} \tag{6.122}$$

Note. The weight in (eq. 6.122) 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 [30, 31] is used to determine the 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. 6.123).

$$safetyMargin = \max \begin{Bmatrix} case.UAS_i.roundaboutSafetyMargin, \\ case.UAS_i.defaultSafetyMargin \end{Bmatrix},$$
$$\forall case \in Cluster, \quad UAS_i \in \{1, 2\} \tag{6.123}$$

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 the risk of medium/short 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 the nearby surroundings of this case collision point (cluster).
angle of approach(α)	is calculated based on attendants *velocity* and *position*, the 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 a list of collision cases which are active and can have an impact on this *collision case*.
head case	is a reference to collision case in the time frame when it was first opened.
Collision case indicators	
linear intersection	indicates if there was a safety breach on linear trajectories estimation with the risk of direct collision.
trajectory intersection	indicates if there was a breach on trajectory estimation, with the 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 the case is still open.

Table 6.6: Collision case structure for given decision time-frame.

6.8 UTM Directives Implementation on UAS

This section is follow up of *UTM functionality definition* (sec. 6.7), outlining realization of *UTM directives* on *UAS* side (sec. 6.8.1, 6.8.2).

Reasoning: The *Avoidance* process and *UTM directives fulfillment* are different in every national airspace. The ICAO issues recommendation [32, 44] which are implemented by every member country, some of the 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 for *variable parts* of *obstacle avoidance process*. The *dynamic parts* need 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 [56] in the form of *Movement Restrictions* and *Waypoint Changes*.

6.8.1 Rule Engine Architecture

Purpose: The *core process* of *Avoidance Grid Run* (sec. 6.6.1) and *Mission Control Run* (sec. 6.6.2) needs to be enhanced based on the situation. The architecture is based on *aspect-oriented approach* [57]. The key ideas and concepts are taken from rule engine implementation for multiagent navigation system [58].

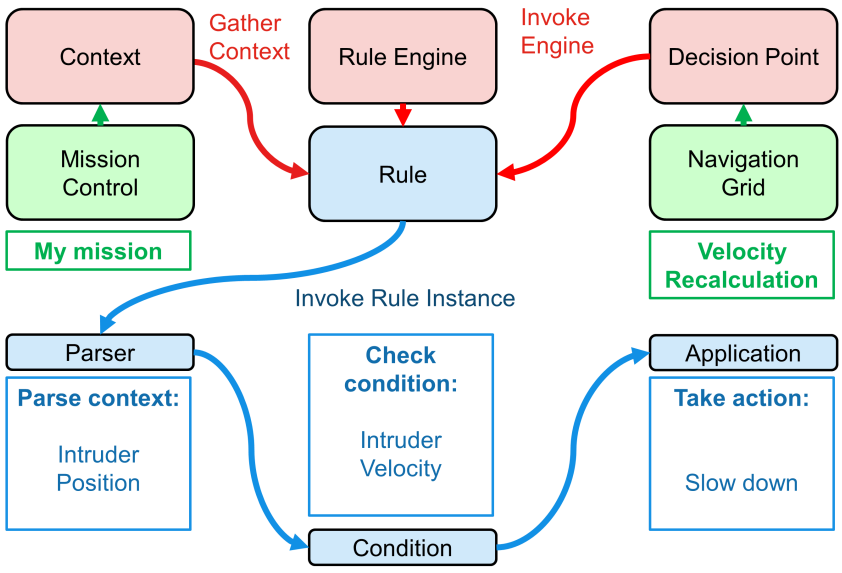


Figure 6.28: Rule engine components overview.

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*.
A rules the in context of this work are pieces of code which have a semi-static structure consisting of following parts (fig. 6.28):

1. *Decision Point* - hook point in the 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 a 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, by *disabling trajectories* in *Reach Set* in our implementation.

Example: The *UAS* is flying in controlled airspace. The *intruder* shows in front of *UAS*. The *UAS* is faster than an *intruder*. The *UAS* tries to obtain permission for *Overtake*. The *UTM* does not allow *overtake*, because of *insufficient UAS maneuverability capability*. The *Rule* (fig. 6.28) 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. *The rule is invoked:*

   a. *The parser* parses the context which is *intruder's Position Notification* containing its heading and velocity.

   b. *The condition* is checked to *relative intruder velocity*. The *evaluation* is positiv, when the UAS is *pursuing the intruder*.

   c. *Application* of *Rule* is the last step, in this case, the *UAS* will slow down.

Configurability: The *Rule Engine* enables real-time configuration. The *Enabled Rules Table* have been implemented to enforce specific rules in a specific context.

The *Rules* can be invoked from *Rule Application*; this enables effective rule chaining and piece-wise functionality split.

6.8.2 Rule Engine Setup

Configuration: The *Rule Engine Architecture* (fig. 6.28) is configured to handle *UTM functionality* for *Collision Case Resolution* (sec. 6.7.6). The overview of *Context* (Green), *Decision Points* (red) and *Rules to be Invoked* (cyan) is given in (fig. 6.29).

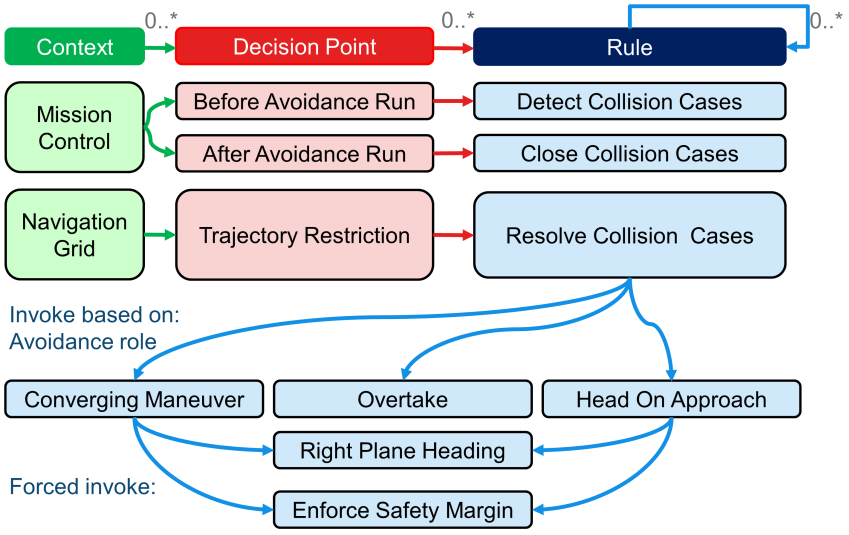


Figure 6.29: Rule engine initialization with Rules of the air.

Decision Points: The *Decisions* are bounded to *Mission Control Run Process* (fig. 6.22) in the 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) - an 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 afterward.

Note. The *Weather Case* (app. **??**) is handled similarly. The mission control loop (fig. 6.22) have rules with separate *Decision Points* to enforce *hard constraints* (before step 9.) and *soft constraints* (before step 10.).

Road map: The *implemented rules*(cyan) are separated into the following categories:

1. *Management Rules* - managing collision cases (additional control flow):

   a. *Detect Collision Cases* (app. **??**) - the detection of active participation in received *collision cases* and generation of *restrictions*.

b. *Resolve Collision Cases* (app. **??**) - the enforcement of *active avoidance roles* in *collision cases*. The one *Restriction Rule* is invoked directly.

c. *Close Collision Cases* (app. **??**) - 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* (app. **??**) implementation of *Converging Avoidance* (sec. 6.7.3).

b. *Head On Approach* (app. **??**) implementation of *Virtual Roundabout Enforcement* (sec. 6.7.2).

c. *Overtake* (app. **??**) implementation of *overtaking maneuver* for *Overtaking plane* (sec. 6.7.4).

3. *Miscellaneous Rules* - reused pieces of code in *Head-On* and *Converging Situations*:

a. *Right Plane Heading* (app. **??**) - restrict all trajectories heading to space separated by parametric plane in *Avoidance Grid* which is heading or belonging to plane.

b. *Enforce Safety Margin* (app. **??**) - restrict all *Trajectories Segments* which are in proximity of *Collision Point* lesser than *Enforced Safety Margin*.

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.

[13] John Birmingham and Peter Kent. Tree-searching and tree-pruning techniques. In *Computer chess compendium*, pages 123–128. Springer, 1988.

[14] 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.

[15] 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.

[16] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[17] Jay Shively. Uas integration in the nas: Detect and avoid. 2018.

[18] Mike Marston and Gabe Baca. Acas-xu initial self-separation flight tests, 2015.

[19] 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.

[20] 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.

[21] 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.

[22] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. The MIT press, 2009.

[23] 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.

[24] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[25] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151–162. IEEE, 1975.

[26] 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.

[27] Jon Louis Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, (9):643–647, 1979.

[28] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *17th annual symposium on foundations of computer science*, pages 208–215. IEEE, 1976.

[29] Duncan Mclaren Young Sommerville. *Analytical geometry of three dimensions*. Cambridge University Press, 2016.

[30] Jack Ritter. An efficient bounding sphere. *Graphics gems*, 1:301–303, 1990.

[31] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*, pages 359–370. Springer, 1991.

[32] ICAO. 4444: Procedures for air navigation services. Technical report, ICAO, 2018.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. A microeconomic view of data mining. *Data mining and knowledge discovery*, 2(4):311–324, 1998.

[38] Ingrid Gerdes, Annette Temme, and Michael Schultz. Dynamic airspace sectorization using controller task load. *Sixth SESAR Innovation Days*, 2016.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] Nico Zimmer and Keyvan Bayram. Selective weather notification, March 18 2014. US Patent 8,674,850.

[44] ICAO. Annex 2 (rules of the air). Technical report, ICAO, 2018.

[45] ICAO. Annex 11 (air traffic services). Technical report, ICAO, 2018.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] Nikolai Nikolaevich Krasovskij, Andrei Izmailovich Subbotin, and Samuel Kotz. *Game-theoretical control problems*. Springer-Verlag New York, Inc., 1987.

[51] NN Krasovskii and AI Subbotin. Game-theoretical control problems. translated from the russian by samuel kotz, 1988.

[52] 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.

[53] 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.

[54] M Ebrahim Fouladvand, Zeinab Sadjadi, and M Reza Shaebani. Characteristics of vehicular traffic flow at a roundabout. *Physical Review E*, 70(4):046132, 2004.

[55] 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.

[56] 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.

[57] Ernest Friedman Hill. *Jess in action: Java rule-based systems*. Manning Publications Co., 2003.

[58] Georg S Seyboth, Dimos V Dimarogonas, and Karl H Johansson. Event-based broadcasting for multi-agent average consensus. *Automatica*, 49(1):245–252, 2013.