Chapter 6

Approach

There levels of *Avoidance* are summarized in (fig. 6.1).

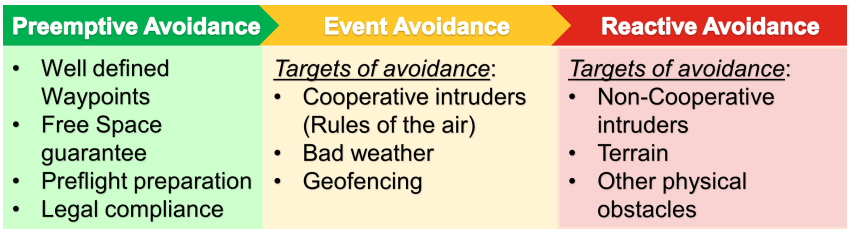


Figure 6.1: Avoidance levels based on reaction time.

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

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

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

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

Basic Idea: Create deterministic finite-time *Reactive Avoidance* based on *Reach sets* to ensure *trajectory feasibility*. Enhance method with 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:

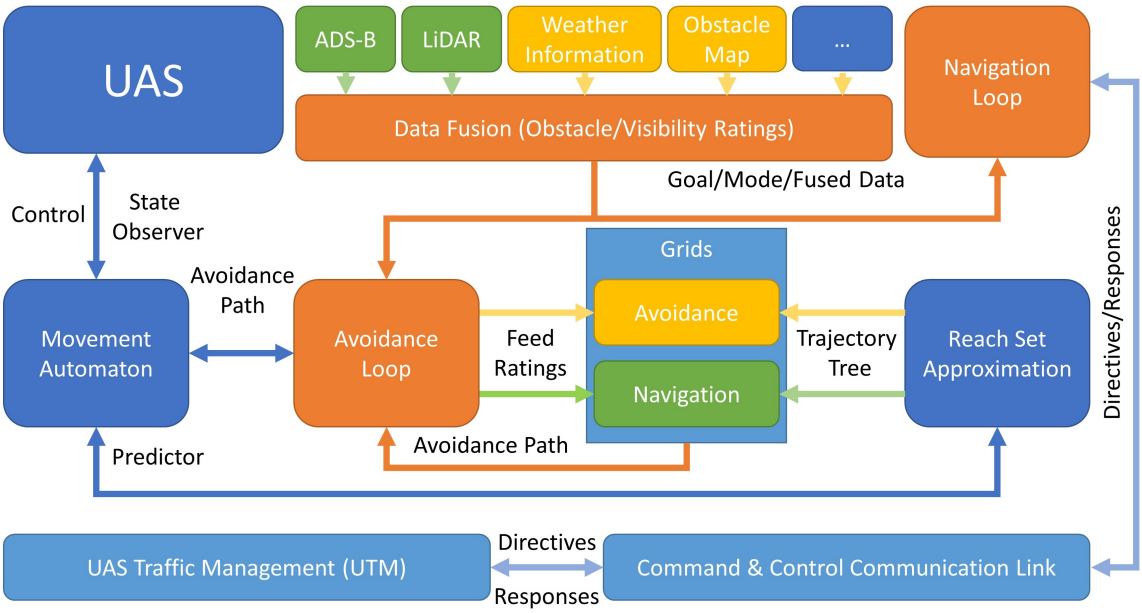


Figure 6.2: Avoidance Framework Concept.

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. 6.2.4).

2. *Movement Automaton* (Role: Control Interface/Predictor) - consumes *Discrete Command Chain* to generate discrete *reference trajectory*, it can be also used as a predictor of *future UAS states* (sec. 6.2.7). The movement Automaton used in this work is given in (sec. 6.2.5).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Communication in Avoidance Framework:

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

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

3. *Reach Set* $\leftrightarrow$ *Grids* - providing trajectory set depending on active mode (Navigation/Emergency Avoidance).

4. *Avoidance Loop* $\leftrightarrow$ *Data Fusion* - assessing the situation in *operational space* based on sensor readings/information sources.

5. *Avoidance Loop* $\leftrightarrow$ *Navigation Loop* - determining long term goal based on situation assessment and UTM directives.

6. *Avoidance Loop* $\rightarrow$ *Grids* - feeding assessment data and constraints into selected operational space Grid.

7. *Grids* $\rightarrow$ *Avoidance Loop* - returning feasible and *cost effective* avoidance path after situation assessment and *Reach set* pruning.

8. *Avoidance Loop* $\rightarrow$ *Movement Automaton* - issuing and monitoring movement commands based on actual *avoidance strategy*.

9. *Navigation Loop* $\leftrightarrow$ *C2 Link* $\leftrightarrow$ *UTM* - communication to receive directives and send fulfillment.

6.2 UAS Model and Control

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

1. *Movement Automaton Background* (sec. 6.2.1) - the listing of related work and similar approaches to ours.

2. *Specialization of Hybrid Automaton* (sec. 6.2.2) - the specialization of the hybrid automaton to fulfill control/approximation roles in our approach.

3. *Formal Movement Automaton Definition* (sec. 6.2.3) - the formal definition of *movement automaton* used in our approach.

4. *Used UAS Nonlinear Model* (sec. 6.2.4) - simple plane model used in this work as *controlled plant*.

5. *Used Movement Automaton* (sec. 6.2.5) - movement automaton for *UAS Nonlinear Model* constructed from scratch.

6. *Segmented Movement Automaton* (sec. 6.2.6) - for more complex systems the *State Space* can be *separated into Segments* and *segment movement automaton* is used to generate *thick reference trajectory*.

7. *Reference Trajectory Generator* (sec. 6.2.7) - other use of *Movement Automaton* as predictor for *reference trajectory calculation*.

6.2.1 Movement Automaton Background

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

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

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

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

1. Newton gradient method for evasive car maneuvers [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 Specialization of Hybrid Automaton

Idea: There is a need for *fast trajectory approximation* method. The basic idea is taken from pilot steering an plane. The pilot is issued a commands from an navigator in very short and precise manner. The movement has its primitive phase when steering is static and its transition phase when steering is moving from one static position to another.

Imagine having vertical and horizontal flaps on airplane (fig. 6.3). The *navigator* is issuing a command every second to a pilot. Each command is translated by hands of the pilot to an input signal (blue line). The command validity period (black frame) is split into *transition period* (red frame) when input signal is changing and primitive period (magenta frame) when the position of input signal is static.

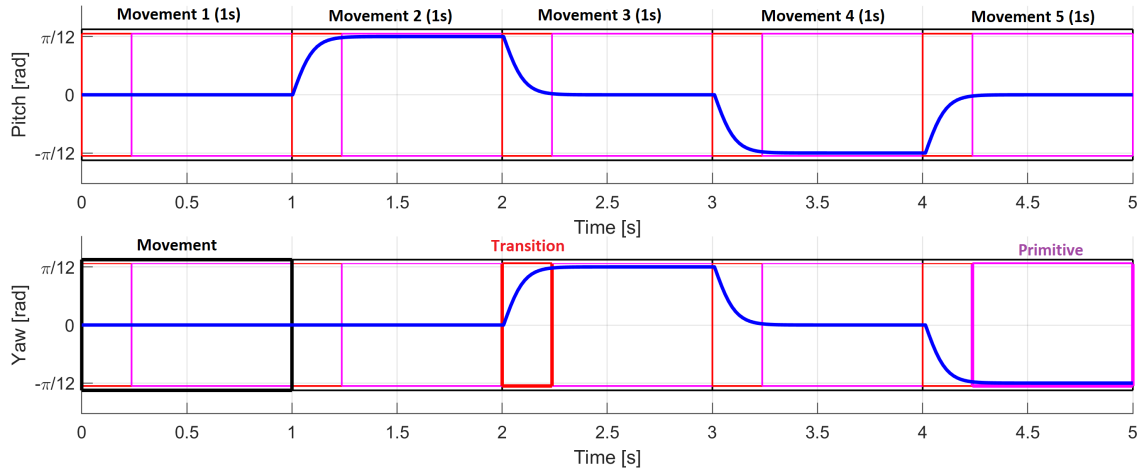


Figure 6.3: Example of input signal segmentation to movements.

Note. The hybrid automaton (sec. **??**) can be used as the base for simple control mechanism imitating navigators command execution by pilot. The automaton states can be mapped to primitives and transitions. The reset map needs to be replaced with external order issuer to ensure smooth execution of commands.

The future commands will be stacked in the buffer from which they will be picked for an execution.

Definition 1. *Movement Primitive:*
States *from* Hybrid automaton *can be taken as* Movements *in* Movement Automaton. MovementPrimitive *(eq. 6.1) is describing the* Movement *behaviour as transfer function* VectorField *enriched with parameters.*

$$\begin{aligned} &MovementPrimitive(vectorField, minimalDuration, parameters) \\ &VectorField : SystemState \times parameters \rightarrow SystemState \end{aligned} \tag{6.1}$$

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

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

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

$$position(t) = \Theta(\dots) \begin{cases} t \in [0,5] & : 10 \times t + position(0) \\ t \in (5,15] & : 0 \times (t-5) + position(5) \\ t \in (15,22] & : 4 \times (t-15) + position(15) \end{cases} \tag{6.2}$$

The *example* given by (eq. 6.2) is fairly primitive, but imagine UAS system given by nonlinear dynamics [9]. Then defining transfer function for given command chain can be impossible.

Definition 2. *Movement Transition:*
System state *can be different than intended movement application, the notion of* Transition *is therefore introduced as stabilizing element in movement chaining (eq. 6.3).*

$$Transition : MovementPrimitive \times SystemState \rightarrow MovementPrimitive \tag{6.3}$$

Trajectory with Transitions: Introducing two transitions $Transition(Move, Stay)$ and $Transition(Stay, Move)$ reflecting periods when vehicle stop moving or speed-up to desired velocity. The transfer function (eq. 6.2) can be rewritten as combination of *MovementPrimitives* (eq. 6.1) and *Transitions* (eq. 6.3):

$$\begin{aligned} &Transition(Stay, Move), Move(5s, 10m/s), \\ &\qquad Transition(Move, Stay), Stay(10s), \\ &\qquad\qquad Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{6.4}$$

.

Note. There are two types of *MovementPrimitives*:

1. *Stationary* - when system state is considered neutral and they are considered as entry point for automaton.

2. *Dynamic* - when the system state is considered evolving and they needs to be terminated with *stationary* transition.

Movement Mapping Example: Transition/MovementPrimitive pairs (eq. 6.3) can be mapped into movements (eq. 6.5).

$$\begin{aligned} Move(5s, 10m/s) : Transition(Stay, Move), Move(5s, 10m/s), \\ Stay(10s) : Transition(Move, Stay), Stay(10s), \\ Move(7s, 4m/s) : Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{6.5}$$

Definition 3. *Movement:*
Movement can consist from multiple Transitions *(eq. 6.3) and one* MovementPrimitive *(eq. 6.1), the duration of* MovementPrimitive *can be shortened by* Transitions *duration.* Movement *is defined as follows:*

$$Movement \begin{pmatrix} initialState, \\ initialTime[0..1], \\ duration, \\ parameters[0..1] \end{pmatrix} = Chain \begin{pmatrix} InitialTransition(\dots)[0..*], \\ MovementPrimitive \begin{pmatrix} transitionState, \\ remainingDuration, \\ parameters \end{pmatrix} \\ LeaveTransition(\dots)[0..*], \end{pmatrix} \tag{6.6}$$

Chain function *connects multiple* initial Transitions *which are appliead at* initialState *at* initialTime. *Then own* MovementPrimitive *(eq. 6.1) is invoked with* transitionnsState. Transitions state *is state changed by* Initial Transitions. *After* Movement Primitive *there can be* Leave Transitions Movement

Minimal Movement Time: Given by (eq. 6.7) for *movement* is given as sum of *MovementPrimitive* (eq. 6.1) minimal time, and *Transition* (eq. 6.3) in/out combined minimal time.

$$minimalTime(Movement) = \begin{matrix} minimalTime(MovementPrimitive) + \\ \max_{in/out} \{time(Transition)\} \end{matrix} \tag{6.7}$$

Movement Chaining: *Movements* can be *chained* and applied to initial *system state* to generate *system trajectory.* Example of trajectory is given by (eq. 6.2). Movements

are reversibly obtained by participation such *trajectory* into *Movement primitives* and *Transitions*. Then sample *Trajectory* for $n \in \mathbb{N}^+$ movements looks like (eq. 6.8).

$$\begin{aligned}
&Trajectory(t_0) = State(t_0)\\
&Trajectory(t_0, t_1] = Movement_1(Trajectory(t_0), t_0, duration_1, parameters_1)\\
&Trajectory(t_1, t_2] = Movement_2(Trajectory(t_1), t_1, duration_2, parameters_2)\\
&Trajectory(t_2, t_3] = Movement_3(Trajectory(t_2), t_2, duration_3, parameters_3)\\
&\vdots\\
&Trajectory(t_{n-1}, t_n] = Movement_n(Trajectory(t_{n-1}), t_{n-1}, duration_n, parameters_n)
\end{aligned} \tag{6.8}$$

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

Definition 4. *Movement Buffer:*
Movements *can be chained into* Buffer *with assumption of* continuous movement execution. Continuous movement executions *each movement in chain (eq. 6.8) is executed in time interval* $\tau_i = (t_{i-1}, t_i]$ *where* i *is movement order and* $\forall\ Movement_i$ *starting time is* t_0 *or* t_{i-1} *from previous movement. With given assumption* Buffer *is given as (eq. 6.9) with parameters* t_{i-1}, t_i *omitted, due* t_0 *and* $duration_i$ *dependency.*

$$Buffer = \{Movement_i(duration_i, parameters_i)\}\, i \in \mathbb{N}^+ \tag{6.9}$$

Definition 5. *Movement Automaton Trajectory:*
Let say system State$\in \mathbb{R}^n$ *which* Trajectory *is defined by movement chaining (eq. 6.8), applied on some* initial time $t_0 \in \mathbb{R}^+$ *and final time* $t_f = t_0 + \sum_{i=1}^{I} duration_i$, *with movements contained in* Buffer *(eq. 6.9) is given as* Trajectory *(eq. 6.10).*

$$Trajectory(t_0, State(t_0), Buffer)\ or\ Trajectory(State_0, Buffer)\ if\ t_0 = 0 \tag{6.10}$$

Note. The space dimension of *Trajectories* is $\mathbb{R}^{n+1}$ if the space dimension of state *Space* is R^n, because *Trajectory space* contains evolution of *Space* in time interval $T[t_0, t_f]$.
The transformation from *transfer function* (eq. 6.2) to *trajectory* (eq. 6.10) is natural, only set of *Movement primitives* (eq. 6.1) and set of *Transitions* (eq. 6.3) is required.

State Projection: *Trajectory* (eq. 6.10)is naturally evolution of space over time, then there exists *StateProjection* function (eq. 6.11) which returns *State* for specific *Time*.

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{6.11}$$

6.2.3 Formal Movement Automaton Definition

Definition 6. *Movement Automaton is given as follow:*

$$InitialState :\in \mathbb{R}^h, h \in \mathbb{N}^+ \tag{6.12}$$

$$System : \dot{State} = f(Time, State, Input) \ or \ vectorField \tag{6.13}$$

$$Primitives = \left\{ MovementPrimitive_i \begin{pmatrix} vectorField, \\ minimalDuration, \\ parameters \end{pmatrix} \right\} i \in \mathbb{N}^+ \tag{6.14}$$

$$Transitions = \left\{ Transition_j \begin{pmatrix} MovementPrimitive_l, \\ MovementPrimitive_k \end{pmatrix}_{k \neq l} \right\} j \in N^+ \tag{6.15}$$

$$Movements = \left\{ Movement_m \begin{bmatrix} Transition_o[0..*], \\ MovementPrimitive_p \\ Transition_r[0..*], \end{bmatrix}_{o \neq r} \right\} m \in N^+ \tag{6.16}$$

$$Buffer = \{Movement_s(duration_s, parameters_s)\} \, s \in \mathbb{N}^+ \tag{6.17}$$

$$Executed = \{Movement_s(duration_s, parameters_t)\} \, t \in \mathbb{N}^+ \tag{6.18}$$

$$Builder : Movement \times MovementPrimitive \rightarrow Movement \tag{6.19}$$

$$Trajectory : InitialState \times Movement^u \rightarrow State \times Time, u \in N^+ \tag{6.20}$$

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{6.21}$$

System *(eq. 6.13) is given in form of* differential equations $\dot{x} = f(t, x, u)$ *or* other transformable equivalent*, with* initial state *(eq. 6.12).*

Movements *(eq. 6.8) are defined as sequence of necessary* initial transitions *(eq. 6.15),* movement primitive *(eq. 6.14), and,* leave transitions *(6.15).*

Buffer *contains a set of* movement primitives *(eq. 6.14) to be executed in order to achieve desired goal.* Builder *(eq. 6.19) assures that first* movement primitive *(eq. 6.1)from* Buffer *(eq. 6.17) is transformed into* next movement *(eq. 6.16) based on* current movement *(eq.6.16).*

The system trajectory *(eq. 6.20) is defined in (eq. 6.10).* State projection *(eqs. 6.11,6.21) is giving* State *variable for time* $t \in [t_0, t_{max}]$ *where* t_max *is given by:*

$$t_{max} = t_0 + \sum_{i=1,u} Buffer.Movement(i).movementDuration \tag{6.22}$$

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

The reach set R (6.23) for system d/dt state $= model(state, input)$ with initial state $state_0 = state(t_i)$ in time interval $[t_i, t_{i+1}[$ is with existing control strategy $input(t) \in ControlStrategy(t)$. The reach set $R(state_0, t_0, t_1)$ where $t_1 > t_0$.

$$R(state_0, t_0, t_1) = \bigcup \{state(s) : input(s) \in ControlStrategy(s), s \in (t_0, t_1]\} \tag{6.23}$$

The reach set $\mathcal{R}$ (6.24) of the system under the control of the *movement automation* consist from the set of trajectories $Trajectory(initialState, buffer)$, which are executed in constrained time period $[t_i, t_{i+1}[$.

$$ReachSet(state_0, t_i, t_{i+1}) = \{Trajectory(state_0, buffer) : \text{duration}(buffer) \leq (t_{i+1} - t_i)\} \tag{6.24}$$

Note. Weak Invariance:

When the UAS is under the control of the movement automaton for the obstacle avoidance problem, by design of the avoidance algorithm, the trajectories of the UAV will not intersect any threat. This means that the controlled system d/dt state $= model(state, input)$ is *weakly invariant* with respect to the complement of the threats, and with respect to the free space. A pair $(state, SafeSpace)$, where d/dt state $= model(state, input)$ and $SafeSpace$ is a closed set, is weakly invariant if there exist controls such that a trajectory starting inside $State_0 \in SafeSpace$ remains inside $State(t) \in SafeSpace$ [10].

6.2.4 Used UAS Nonlinear Model

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

State Vector (eq. 6.25) defined as positional state in euclidean position in right-hand euclidean space, where *x, y, z* can be abstracted as latitude, longitude, altitude.

$$state = [x, y, z, roll, pitch, yaw]^T \tag{6.25}$$

Input Vector (eq. 6.26) is defined as linear velocity of UAS v and angular speed of rigid body $\omega_{roll}, \omega_{pitch}, \omega_{yaw}$.

$$input = [v, \omega_{roll}, \omega_{pitch}, \omega_{yaw}]^T \tag{6.26}$$

Velocity vector function (eq. 6.27) is is defined through standard rotation matrix and linear velocity v, oriented velocity $[v_x,\ v_y,\ v_z]$ given by (eq. 6.28).

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v\cos(pitch)\cos(yaw) \\ v\cos(pitch)\sin(yaw) \\ -v\sin(pitch) \end{bmatrix} \tag{6.27}$$

UAS Nonlinear Model (eq. 6.28) is given by *first order equations:*

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}time} &= v\cos(pitch)\cos(yaw); & \frac{\mathrm{d}roll}{\mathrm{d}time} &= \omega_{roll}; \\ \frac{\mathrm{d}y}{\mathrm{d}time} &= v\cos(pitch)\sin(yaw); & \frac{\mathrm{d}pitch}{\mathrm{d}time} &= \omega_{pitch}; \\ \frac{\mathrm{d}z}{\mathrm{d}time} &= -v\sin(pitch); & \frac{\mathrm{d}yaw}{\mathrm{d}time} &= \omega_{yaw}; \end{aligned} \tag{6.28}$$

6.2.5 Used Movement Automaton for UAS Model

Motivation: An *UAS Nonlinear Model* (eq. 6.28) can be modeled by *Movement Automaton* (def. 6).

Movement Primitives by (def. 1) are given as (eq. 6.1). To define primitives the *minimal time* is $1s$. The *maximal duration* is also $1s$.

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

Under the assumption (as. 1) the *movement transitions* (def. 2) have 0 duration.

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

Movements (def. 3) for *fixed step* k we start with discretization of the input variables. The *linear velocity* in text step is given:

$$v(k+1) = v(k) + \delta v(k) \tag{6.29}$$

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

$$\begin{aligned} roll(k+1) &= roll(k) + \delta roll(k) \\ pitch(k+1) &= pitch(k) + \delta pitch(k) \\ yaw(k+1) &= yaw(k) + \delta yaw(k) \end{aligned} \tag{6.30}$$

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:

$$\begin{aligned} x(k+1) &= x(k) + v(k+1)\cos(pitch(k+1))\cos(yaw(k+1)) &&= \delta x(k) \\ y(k+1) &= y(k) + v(k+1)\cos(pitch(k+1))\sin(yaw(k+1)) &&= \delta y(k) \\ z(k+1) &= z(k) - v(k+1)\sin(pitch(k+1)) &&= \delta z(k) \\ time(k+1) &= time(k) + 1 &&= \delta time(k) \end{aligned} \tag{6.31}$$

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

$$input(k) = \begin{bmatrix} \delta x(k), \delta y(k), \delta z(k), \delta v(k), \\ \delta roll(k), \delta pitch(k), \delta yaw(k), \delta time(k) \end{bmatrix}^T \tag{6.32}$$

The *state vector* for discrete time is given:

$$state(k) = \begin{bmatrix} x(k), y(k), z(k), v(k), \\ roll(k), pitch(k), yaw(k), time(k) \end{bmatrix}^T \tag{6.33}$$

The nonlinear model (eq. 6.28) is then reduced to *linear discrete model* (eq. 6.34) given by *apply movements* function (eq. 6.29, 6.30, 6.31).

$$state(k+1) = applyMovement(state(k), input(k)) \tag{6.34}$$

Movement Set for linear discrete model (eq. 6.34) is defined as set of extreme unitary movements on main axes (tab. 6.1) and diagonal axes (tab. 6.2).

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

Table 6.1: Input values for main axes movements.

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

Table 6.2: Input values for diagonal axes movements.

Note. Movement set in shorten form is given as

$$MovementSet = \left\{ \begin{array}{c} Straight, Left, Right, Up, Down, \\ DownLeft, DownRight, UpLeft, UpRight \end{array} \right\} \tag{6.35}$$

Trajectory by (def. 5) for initial time $time = 0$, initial state $state(0)$ and *Movement Buffer* (from def. 4):

$$Buffer \in MovementSet^{*}(eq.6.35), \quad |Buffer| \in \mathbb{N} \tag{6.36}$$

Trajectory (eq. 6.37) is then given as the time-series of discrete states:

$$Trajectory(state(0), Buffer) = \left\{ \begin{array}{c} state(0) + \sum_{j=0}^{i-1} input(movement(j)) : \\ i \in \{1 \dots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{array} \right\} \tag{6.37}$$

Trajectory (eq. 6.37) is ordered set of states bounded to discrete time $0 \dots n$, where n is member count of *Buffer*. Trajectory set has $n + 1$ members:

$$\begin{array}{l} Trajectory(state(0), Buffer) = \\ \left\{ \begin{array}{l} state(0) = state(0) + \{\} \\ state(1) = state(0) + input(movement(1)) \\ state(2) = state(0) + input(movement(1)) + input(movement(2)) \\ \qquad \vdots = \vdots \\ state(n) = state(0) + input(movement(1)) + \dots + input(movement(n)) \end{array} \right\} \end{array} \tag{6.38}$$

State Projection (eq. 6.39) for the *Trajectory* (eq. 6.37) is given as follow:

$$StateProjection(Trajectory, time) = Trajectory.getMemberByIndex(time + 1) \tag{6.39}$$

Note. Movement Automaton for system (eq. 6.28) with given (as. 1) is established with all related properties (sec. 6).

6.2.6 Segmented Movement Automaton

Motivation: Constructing *Movement Automaton* for more complex system can be tedious. Used *Movement Automaton* for *UAS system* (6.28) has decoupled control which is not true for most of the copters/planes [9].

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

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

Definition 7. *State Space partition* Maneuverability *is depending on* Initial State*. There can not be the infinite count of* Movement Automatons*.*

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

$$StateSpace = [ImpactStates, NonImpactingStates] \tag{6.40}$$

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

$$\forall impactState \in ImpactStates, \exists :$$
$$lower(impactState) \leq value(impactState) \leq upper(impactState) \tag{6.41}$$

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

$$impactState \in [lower, upper] :$$
$$\{[lower, separator_1[\cdots \cup \ldots [separator_i, separator_{i+1}[\cdots \cup \ldots$$
$$\cdots \cup \ldots [separator_n, upper]\} =$$
$$= impactStateIntervals(impactState) \tag{6.42}$$

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

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

$$partitionCount = \prod_{impactState \in}^{ImpactStates} |impactStateIntervals(impactState)| \tag{6.43}$$

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

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

$$Segment(index) = \begin{bmatrix} impactState_1 \in impactStateIntervals_1[index_1], \\ \vdots \\ impactState_n \in impactStateIntervals_n[index_n], \\ \vdots \\ NonImpactingStates \end{bmatrix} \tag{6.44}$$

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

$$StateSpace = \bigcup_{\forall \quad index \in |impactStateIntervals|^n} Segment(index) \tag{6.45}$$

Segmented Movement Automaton: The segmentation of *state space* is done in (def. 7) any *state* belongs exactly to *Segment* of *State Space*. For each *Segment* in *State Space*it is possible to assess: *Climb/Descent Rate* $\delta pitch_{max}(k)$, *Turn Rate* $\delta yaw_{max}(k)$, and, *Acceleration* $\delta v_{max}(k)$.

Definition 8. *Movement Automaton for Segment(index)*
For for Model(eq. 6.34) with State (eq. 6.33) the input vector (eq. 6.32) is for position $[x, y, z]$ *and velocity defined like:*

$$\begin{aligned} \delta x(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \cos(\delta yaw(k)) \\ \delta y(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \sin(\delta yaw(k)) \\ \delta z(k) &= -(v(k) + \delta v(k)) \cos(\delta pitch(k)) \\ \delta v(k) &\in [-\delta v(k)_{max}, \delta v(k)_{max}] \end{aligned} \tag{6.46}$$

The acceleration $\delta v(k)$ is in interval $[-\delta v(k)_{max}, \delta v(k)_{max}]$, usually set to 0 ms^{-1}. The change of the orientation angles for *Movement Set* (eq. 6.35) is given in (tab. 6.3,6.4).

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

Table 6.3: Orientation input values for main axes movements.

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

Table 6.4: Orientation input values for diagonal axes movements.

Note. The *Trajectory* is calculated same as in (eq. 6.37). The *State Projection* is given as in (eq. 6.39).

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

Definition 9. *Segmented Movement Automaton For system with segmented state space (eq. 6.45) there is for each state(k) in StateSpace injection function:*

$$ActiveMovementAutomaton : StateSpace \rightarrow MovementAutomaton \tag{6.47}$$

Selecting appropriate movement automaton *implementation (def. 8) for* state(k) $\in$ Segment $\subset$ State Space. *The mapping function (eq. 6.47) is injection mapping every state(k) to Segment then* Movement Automaton Implementation. *The trajectory generated is then given:*

$$Trajectory\begin{pmatrix} state(0), \\ Buffer \end{pmatrix} = \left\{ \begin{array}{c} state(0) + \dots \\ \sum_{j=0}^{i-1} ActiveMovementAutomaton(state(j-1)). \\ .input(movement(j)) \\ i \in \{1 \dots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{array} : \right\} \tag{6.48}$$

6.2.7 Reference Trajectory Generator

Reference Trajectory Generator: Segmented Movement Automaton (def. 9) with *trajectory function* (eq. 6.48) is used as *reference trajectory generator* for *complex systems*.

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

The *Reference trajectory* (eq. 6.49) for *Planned* movement set is given as projection of *Trajectory* time series to position time series $[x, y, z, t]$:

$$ReferenceTrajectory : Trajectory \begin{pmatrix} state(now), \\ Planned \end{pmatrix} \rightarrow \begin{bmatrix} x_{ref} \in \mathbb{R}^{|Planned|} \\ y_{ref} \in \mathbb{R}^{|Planned|} \\ z_{ref} \in \mathbb{R}^{|Planned|} \\ t_{ref} \in \mathbb{R}^{|Planned|} \end{bmatrix} \tag{6.49}$$

Predictor: The *Reference Trajectory Generator* (eq. 6.49) can be also used as predictor.

Note. The *Segmented Movement Automaton* (def. 9) is used in this work with one Segment equal to State space with input function given by (6.1, 6.2). The predictor used in *Reach set computation* is given by (eq. 6.49).

6.3 Space Discretization - Avoidance Grid

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

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

The *Main Sensor* is *LiDAR* (problems **??**.-**??**.). The *effective occupancy computation* [11] is given by clustering *LiDAR* field of vision into *polar coordinates grid*.

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

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

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

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

LiDAR Reading Space Segmentation: The *polar space* can be separated into cells, which bounds the portion the space, similar to *euclidean* space grid. The *reason* for this segmentation is *LiDAR* reading density[1]. The *polar space portions* state can be assessed directly, the *polar* $\rightarrow$ *euclidean* coordinate frame transformation is not time-effective. The *polar space* assessment of *Lidar Data* has minimal complexity and it is cost effective. [12].

Definition 10. *Space partition - cell The* cell *is a portion of space in UAS local polar coordinate frame, given by:*

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

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

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

$$BoundedSpace(cell) = \ldots$$
$$\left\{ point \in \mathbb{R}^3 \ where: \left(\begin{array}{rcl} cell.distance_{start} < & point.distance \leq & cell.distance_{end}, \\ cell.horizontal^{\circ}_{start} < & point.horizontal^{\circ} \leq & cell.horizontal^{\circ}_{end}, \\ cell.vertical^{\circ}_{start} < & point.vertical^{\circ} \leq & cell.vertical^{\circ}_{end} \end{array} \right) \right\} \quad (6.50)$$

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

$$LidarHits(cell) = \{point \in LidarScan : point \in BoundedSpace(cell)\} \quad (6.51)$$

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

$$PassingHits(cell) = \ldots$$
$$\left\{ point \in LidarScan \ where: \left(\begin{array}{rcl} cell.distance_{end} < & point.distance & , \\ cell.horizontal^{\circ}_{start} < & point.horizontal^{\circ} \leq & cell.horizontal^{\circ}_{end}, \\ cell.vertical^{\circ}_{start} < & point.vertical^{\circ} \leq & cell.vertical^{\circ}_{end} \end{array} \right) \right\} \quad (6.52)$$

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

Effective Operation Space - Avoidance Grid: Let start with example, the UAS (fig. 6.4).

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

The *frontal portion* (red line) is a set of cells where *UAS* can make maneuver. The *red portion* size is determined by [13]:

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

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

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

vicinity.

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

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

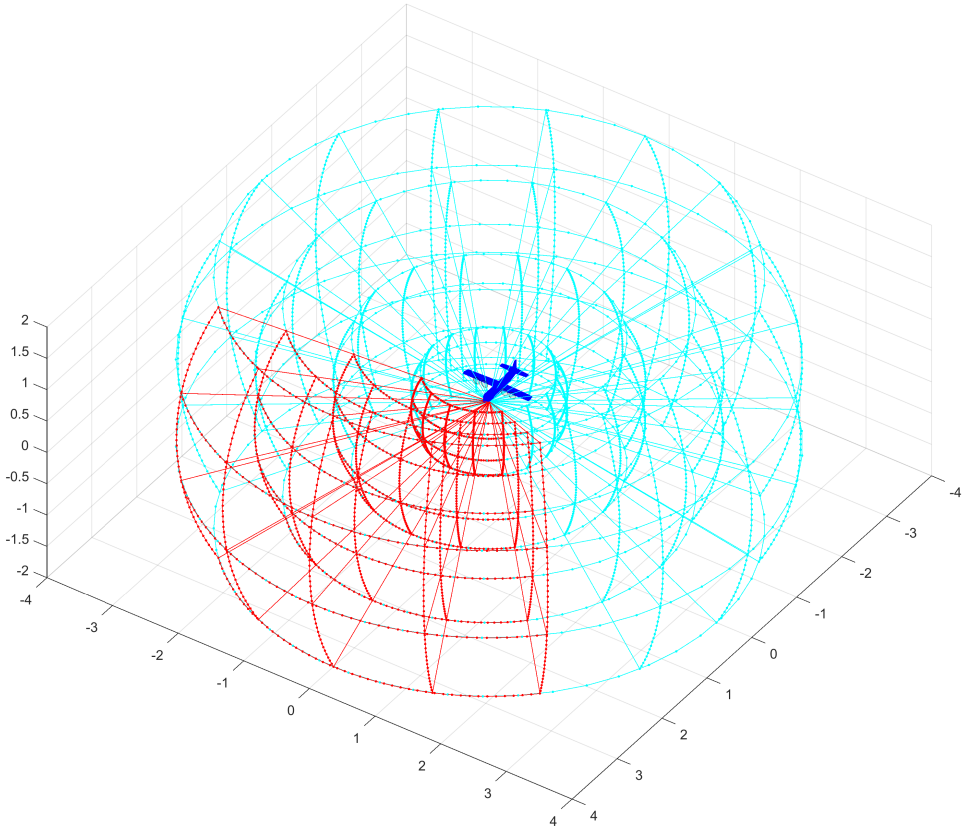


Figure 6.4: Example: The *LiDAR* reading segmentation - cells.

Definition 11. *Avoidance Grid The* effective space boundary *(fig. 6.4 red lines) given by a portion of space in UAS local polar coordinate frame, bounded by:*

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

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

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

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

$$\begin{aligned} layer^{i}_{start} &= (i-1) \times \frac{distance_{end} - distance_{start}}{layerCount} \\ layer^{i}_{end} &= i \times \frac{distance_{end} - distance_{start}}{layerCount} \end{aligned} \quad ; \quad i \in 1 \dots I \tag{6.53}$$

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

$$\begin{aligned} horizontal^{j}_{start} &= (j-1) \times \frac{horizontal^{\circ}_{end} - horizontal^{\circ}_{start}}{horizontalCount} \\ horizontal^{j}_{end} &= j \times \frac{horizontal^{\circ}_{end} - horizontal^{\circ}_{start}}{horizontalCount} \end{aligned} \quad ; \quad j \in 1 \dots J \tag{6.54}$$

$$\begin{aligned} vertical^{k}_{start} &= (k-1) \times \frac{vertical^{\circ}_{end} - vertical^{\circ}_{start}}{verticalCount} \\ vertical^{k}_{end} &= k \times \frac{vertical^{\circ}_{end} - vertical^{\circ}_{start}}{verticalCount} \end{aligned} \quad ; \quad k \in 1 \dots K \tag{6.55}$$

Then $cell_{i,j,k}$ given by (def. 10) is member cell of Avoidance Grid for boundaries:

1. Cell Distance Range *(eq. 6.53) depending on layer index i.*
2. Cell Horizontal Range *(eq. 6.54) depending on horizontal index j.*
3. Cell Vertical Range *(eq. 6.55) depending on horizontal index k.*

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

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

$$AvoidanceGrid = \bigcup cell_{i,j,k} \forall i \in 1 \dots I, j \in 1 \dots J, k \in 1 \dots K \tag{6.56}$$

Trajectory Intersection: The *trajectory* intersection with *Avoidance Grid* is solved in context of *Reach Set Approximation* (def. 13).

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

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

$$Boundary(Sensor \in SensorField) = \{points \in \mathbb{R}^3 : \text{where reliable}\} \tag{6.57}$$

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

$$Boundary(SensorField) = \bigcap_{\forall Sensors} Boundary(Sensor \in SensorField) \tag{6.58}$$

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

$$Boundary(SensorField) \rightarrow AvoidanceGrid : \begin{matrix} \max(distanceRange) \\ \max(horizontalRange) \\ \max(verticalRange) \end{matrix} \tag{6.59}$$

Our approach taken worst LiDAR performance into account [14] and following parameters for avoidance grid were calculated:

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

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

$$layerCount = \frac{|distanceRange|}{\text{avg.} \quad length(movement \in MovementSet)} \tag{6.60}$$

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

The *efficient boundary* is given by *Reach Set*. Estimate reach set coverage space using *ellipsoidal toolbox* [15] up to given *sensor field* maximal distance:

$$Boundary(ReachSet) = Ellipsoid(UASSystem, distance) \tag{6.61}$$

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

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

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

$$Boundary(AvoidanceGrid) = Boundary(ReachSet) \cap Boundary(SensorField) \tag{6.62}$$

The values for *Avoidance Grid Boundary* for our UAS system (sec. 6.2.4) following:

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

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

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

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

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

6.4 Reach Set Approximation

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

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

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

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

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

Definition 12 (Reach set Approximation by Movement Automaton)**.** A trajectory *(def. 5) for system* $\dot{state} = f(time, state, input)$ *under control of the movement automaton* $\mathcal{MA}$ *is given as execution of movement buffer (def. 4) with initial state of system* $state_0$*. Therefore notation* $Trajectory(state_0, buffer)$ *is used.*

The Complete Reach Set *(6.63) for system with initial state* $state_0$ *with existing* control strategy $control(time) \in Controls(time)$*. for time* $\tau > time_0$*.*

$$ReachSet(\tau, time_0, state_0) = \bigcup \{state(s) : control(s) \in Controls(s), s \in (time_0, \tau]\} \tag{6.63}$$

The Reach Set Approximation by Movement Automaton *(6.64) of the system under the control of the movement automation* $\mathcal{MA}$ *consist from the set of trajectories* $Trajectory\ (state_0,\ Buffer)$*, which are executed in constrained time* $\tau > time_0$*.*

$$ReachSet(\tau, time_0, state_0) = \left\{ Trajectory(state_0, buffer) : \begin{matrix} duration(buffer) \\ \leq \\ (time_0 - \tau) \end{matrix} \right\} \quad (6.64)$$

Note. Reach Set Approximation (def. 12) is subset of *Full Reach Set* (def. **??**) in continuous space $\mathbb{R}^n$ it inherits all important properties, like *Invariance* [10].

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. 12) 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 13 (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:

$$ReachSet \begin{pmatrix} time_0, \\ state_0, \\ AvoidanceGrid_0 \end{pmatrix} = \left\{ \begin{matrix} Trajectory(\dots) \\ \in \\ ReachSet(6.64) \end{matrix} : \begin{matrix} \forall segment \in AvoidanceGrid_0, \\ segment \in Trajectory(\dots) \end{matrix} \right\} \quad (6.65)$$

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. 6.65) 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: (6.66) *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 $Trajectory(state_0, \varnothing)$.

$$ReachSet(state_0, \dots) = \left\{ Trajectory(state_0, buffer) : \begin{matrix} buffer \in Movements^i, \\ i \in \{1, \dots, k\} \end{matrix} \right\} \tag{6.66}$$

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:

$$intersection(segment, Set) : \begin{cases} \exists point \in segment, & : true \\ \quad point \in Set & \\ Otherwise & : false \end{cases} \tag{6.67}$$

Definition 14 (Pruned Reach Set)**.** *For* Reach set *represented as* Trajectory Tree *(eq. 6.66) and some constraint set (Set) where exist* intersection function *(eq. 6.67). The* Pruned Reach set *is given as follows:*

$$Prune(ReachSet, Set) = \left\{ Trajectory(\dots) : \begin{matrix} \forall segment \in Trajectory, \\ \neg intersection(segment, Set) \end{matrix} \right\} \tag{6.68}$$

Note. Pruning(def. 14) [16] is applicable multiple times for various *Constraints Set*.

Example of *Approximated Reach set Calculation* (def. 12), *Reach Set Containment* (def. 13), and, *Pruning* is given in [2].

6.4.1 Reach Set Performance Criteria

Motivation: The need to Make *Reach Set* scalable approach. This may be a problem due the *Expansion rate*. *Reach set* represented as a *Trajectory Tree* (eq. 6.66) for Avoidance Grid with $layerCount$ and Movement automaton with $movementCount$, the *Node count* is given as:

$$1 + \left(\sum_{i \in \{1 \dots layerCount\}} (movementCount)^i \right) \tag{6.69}$$

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

Core Performance Criteria: The scaling factor (eq. 6.69) 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 15 (Trajectory footprint)**.** *For* Trajectory *from* Reach set *(def. 13) defined for* Avoidance Grid *has membership function.* Membership Function *returns* ordered set of passing cells*:*

$$footprint\begin{pmatrix} Trajectory, \\ AvoidanceGrid \end{pmatrix} = \left\{ \begin{matrix} cell \in AvoidanceGrid : \\ isMember(trajectory, cell) \end{matrix} \right\} \tag{6.70}$$

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

$$isEqual\begin{pmatrix} Trajectory_1, \\ Trajectory_2, \\ AvoidanceGrid \end{pmatrix} : \begin{cases} \begin{pmatrix} footprint(Trajectory_1, \dots) \\ = \\ footprint(Trajectory_2, \dots) \end{pmatrix} & : true \\ Otherwise & : false \end{cases} \tag{6.71}$$

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. 6.70). Similarly there is a possibility to create *Reach set skeleton* containing unique trajectories, by using *equality function* (eq. 6.71). *Coverage set* is sufficient for now.

Definition 16 (Coverage Set)**.** Coverage set *(6.72) is defined for* Avoidance Grid *and* Reach Set *pair as set of unique* Trajectory footprints*:*

$$CoverageSet\begin{pmatrix} AvoidanceGrid, \\ ReachSet \end{pmatrix} = \left\{ footprint\begin{pmatrix} Trajectory, \\ AvoidanceGrid \end{pmatrix} : \begin{matrix} \forall Trajectory \\ \in ReachSet \end{matrix} \right\} \tag{6.72}$$

Coverage set properties: Trajectory footprint (eq. 6.70)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 17 (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*.*

$$\begin{aligned} referenceCoverage &= |CoverageSet(ReferenceReachSet, AvoidanceGrid)| \\ reducedCoverage &= |CoverageSet(ReducedReachSet, AvoidanceGrid)| \\ CoverageRatio &= \frac{reducedCoverage}{referenceCoverage} \in [0, 1] \end{aligned} \tag{6.73}$$

Note. Reference Reach Set is usually *Full Reach Set* containing all possible trajectories in space contained by *Avoidance Grid*. In case *Full Reach Set* can not be computed, Avoidance Grid is too large, most complex *Reach Set* is used as *Reference Reach Set*.

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

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

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

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

$$\begin{gathered} MovementSet = SmoothMovements \cup ChaoticMovements \\ SmoothMovements \cap ChaoticMovements = \varnothing \\ |SmoothMovements| > 0, \quad |ChaoticMovements| > 0 \end{gathered} \tag{6.74}$$

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

$$\begin{aligned} isSmooth(movement) &= \begin{cases} movement \in SmoothMovements & : 1 \\ movement \in ChaoticMovements & : 0 \end{cases} \\ smoothCount(Trajectory(\ldots, buffer)) &= \sum isSmooth(movement), \\ &\qquad \forall movement \in Buffer \end{aligned} \tag{6.75}$$

Definition 18 (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. 6.74), with existing* classification *and* counter *functionals (eq. 6.75) is given as follows:*

$$Smoothness(Trajectory(\ldots, buffer)) = \frac{isSmooth(Trajectory)}{movementCount(Trajectory)} \in [0, 1] \tag{6.76}$$

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

6.4.2 Constrained Trajectory Expansion

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

Survey of motion planning algorithm was given in [17]. 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 .

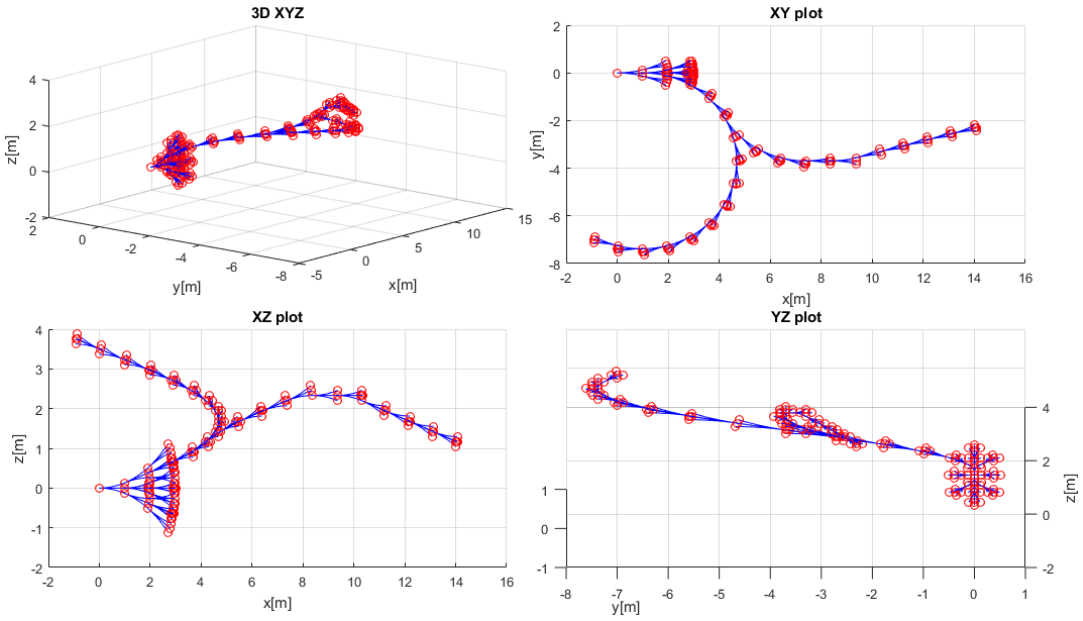


Figure 6.5: *Rapid Exploration tree as result of* Constrained trajectory expansion.

Rapid Exploration Tree (fig. 6.5) was selected, because it enables *Movement Automaton Utilization* and *Property Binding*. Similar approach was used for space exploration [18].

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 *Reachibility* evaluation algorithm.

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

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

Step: Initialization contains base structure preparation like follows:

1. *Avoidance Grid* - Space containing *Reach set* (def. 13).
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. 18).
2. *Expansion of Candidates* - for each *candidate* function *candidate.expandNode (automaton)* is invoked. This function will expand *Candidate Node structure* by appending *Full Trajectory Tree Evolution* until each *Leaf Trajectory* reaches *Next Layer*. Simply put *Par rent Node Node(initialState, buffer, cost, footprint)* buffer is appended by movements until the next layer is reached.
3. *Leftovers purge* - function *reachSet.purge(leftovers)* removes unexpanded *Nodes* leading to cell, effectively removing trajectories which does not lead to *next layer*.
4. *Append Reach Set* - function *reachSet.append(leafs)* puts newly created *Nodes (Trees)* into *Reach Set* structure. The *Wave-front Propagation* for one cell is finished.

Step: After Layer Propagation Purge is covered by function *reachSet.purgeSameFootprint()* which takes trajectories with same footprint and keeps some of them based on *Selection criteria*, more in (sec. 6.4.3, 6.4.4). *Pruning methods* over *Large Decision Trees* are *fast* and *viable* [19].

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
```

6.4.3 Chaotic Reach set

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

Background: There is *Coverage Ratio* property of *Reach Set* (def. 17). It has been shown that creating *Reach Set* via *greedy approach* is not feasible due the *Scaling Factor*. *Contracted Expansion* (sec. 6.4.2) is enabling to apply selection criteria while building *Reach Set* in given *Cell*.

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

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

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

1. *Considered Footprint Length* - how much last cells in footprint should be considered in unique path track, minimal value 1, default value 3, maximal value ∞. 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 *Chaotic 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*.

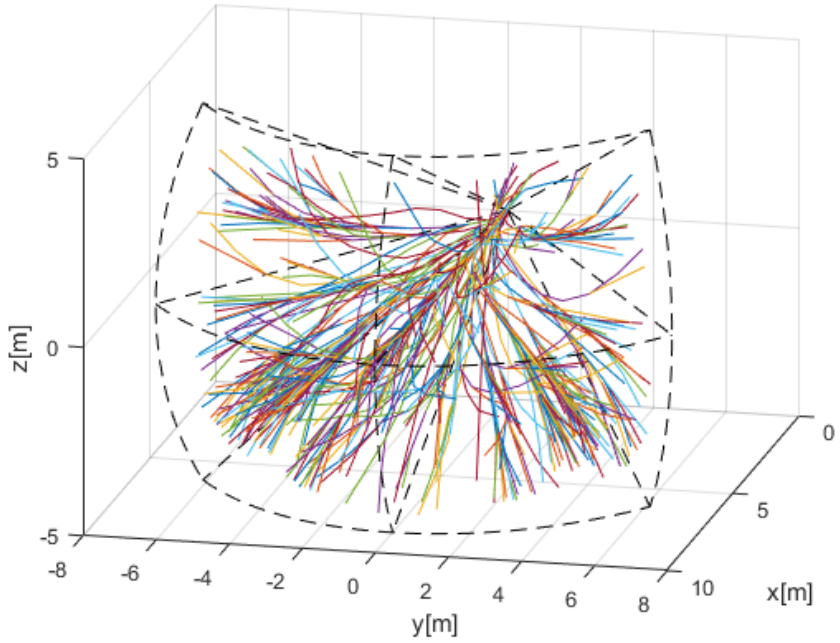


Figure 6.6: *Chaotic* reach set *approximation*.

Pros and Cons: It can be seen from example (fig. 6.6) that *Chaotic Reach Set Approximation Method* (alg. 6.2) generates a lot of *turning* and *shaky trajectories*.

High Coverage Ratio (~ 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:

$$countTrajectories(ReachSet) \leq layerCellCount \times spreadLimit \times size(Movements) \quad (6.77)$$

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

$$countNodes(ReachSet) \leq layerCount \times layerCellCount \times size(Movements) \times spreadLimit \quad (6.78)$$

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

6.4.4 Harmonic Reach set

Motivation: Imagine having an *Avoidance Grid* like (fig. 6.4). There is a need of *Reach Set Approximation* which will have *Smooth Trajectories* (def. 18) going nearby *cell* centers.

Background: The *Smoothness Rating for Trajectory* (def. 18) uses two distinct sets *Smooth Movements* and *Chaotic Movements* (eq. 6.74) which are defined for our *Movement Automaton* (sec. 6.2) like follow:

$$\begin{aligned} SmoothMovements &= \{Straight\} \\ ChaoticMovements &= Movements - SmoothMovements \end{aligned} \tag{6.79}$$

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 *Harmonic 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.

$$getPerformance(Node, Cell) = \frac{distance(Node.Trajectory, Cell.Center)}{SmoothnessRate(Node.Trajectory)} \tag{6.80}$$

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

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

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

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*.

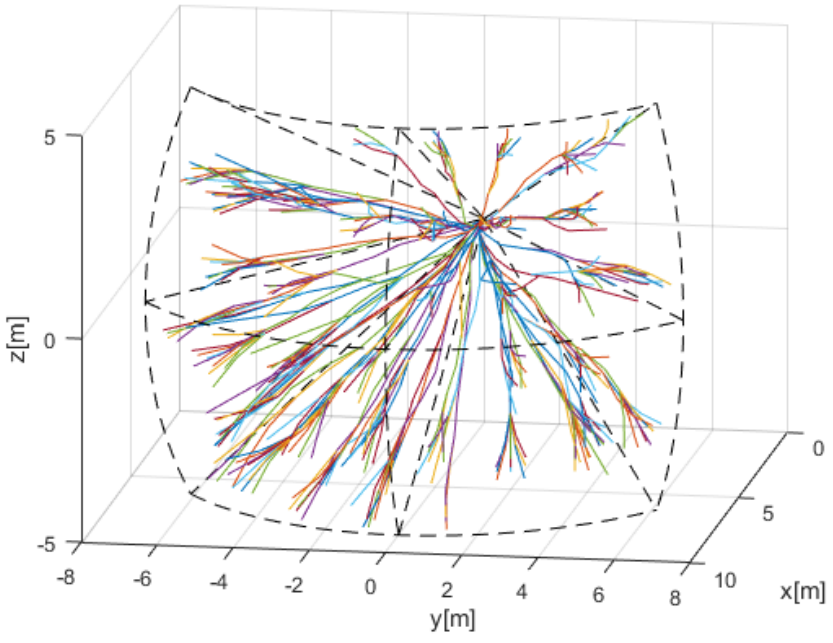


Figure 6.7: *Harmonic* reach set *approximation*.

Pros and Cons: It can bee seen from example (fig. 6.7) that *Harmonic 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:

$$countTrajectories(ReachSet) \leq layerCellCount \times spreadLimit \times size(Movements) \quad (6.81)$$

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

$$countNodes(ReachSet) \leq layerCount \times layerCellCount \times spreadLimit \quad (6.82)$$

Absence of *High Coverage Ratio* disqualifies *Harmonic Reach Set Approximation* to be used for *Emergency Avoidance*. This type of *Reach Set* is feasible for *Open Space Navigation* or *Controlled Airspace Navigation*. Its low turning rate in contained *Trajectories* are desired for such tasks.

6.4.5 Combined Reach Set

Motivation: Harmonic reach set (sec. 6.4.4) is *efficient* for *Navigation in* Controlled Airspace. Chaotic reach set (sec. 6.4.3) is good for *Emergency avoidance*. The need to differentiation between *Navigation* and *Emergency Avoidance* mode is necessary in *Controlled Airspace*. but not in *Non-controlled Airspace*. The combination of *Harmonic* and *Chaotic* reach sets is obvious solution.

Automatic mode switch can be provided by combination of *Navigation Reach Set* and *Avoidance Reach Set* with elevated cost function. Overall having a method to merge multiple trees would be beneficial.

Background: If two *Reach Set Approximation* were calculated for same *Avoidance Grid* and *Initial State*, using same *Movement Automaton* and *UAS model* are possible to merge.

The *Reach Set Approximation* is *tree* with *Root Node* in *initail state* with movement buffer $= \varnothing$. The *movement buffer* in each node can be used as *route trace* during merging procedure. The example two reach set merge can be given as follow, where only *latest* applied movement is taken into account.

$$\left[\begin{array}{l}\texttt{First Reach Set}\\ \quad \varnothing \to \left\langle \begin{array}{l} left \to \left\langle \begin{array}{l} left \\ right \end{array} \right. \\ \varnothing \end{array} \right. \\ \texttt{Second Reach Set} \\ \quad \varnothing \to \left\langle \begin{array}{l} \varnothing \\ right \to \left\langle \begin{array}{l} left \\ right \end{array} \right. \end{array} \right. \end{array}\right] \to \left[\begin{array}{l}\texttt{Combined Reach Set}\\ \quad \varnothing \to \left\langle \begin{array}{ll} left & \to \left\langle \begin{array}{l} left \\ right \end{array} \right. \\ right & \to \left\langle \begin{array}{l} left \\ right \end{array} \right. \end{array} \right. \end{array}\right] \tag{6.83}$$

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. 6.4) is implemented as function $NodecombinedReachSet(\dots)$ which takes root Node with *initial State*, *Avoidance Grid* and respective parameters for each calculation method. *Harmonic spread* for *Harmonic Reach set calculation* and *Chaotic Spread*, *Footprint Length* for *Chaotic Reach set calculation.*

Separate Reach Sets are calculated using *Wave-front propagation* (alg. 6.1) using respective *Constrained Expansion* functions for *Harmonic* (alg. 6.3) and *Chaotic* (alg. 6.2) reach sets.

Combined Reach Set is created using *Node mergeTree(...)* function. Because different cost function or *Bounded Parameters Calculation* may be applied on *Original Reach Sets.*

Cost for *each node* needs to be recalculated due to original reach sets disparity. Function *combined.applyCostFunction()* will recalculate the new cost for each node.

The Goal is to have penalization for *Chaotic behaviour*, implementation of *Automatic Mode Switch* can be done like follows:

1. *Calculate Normal Cost* for Node $Cost(Node)$ for associated trajectory: $Cost(Node.Trajectory)$.

2. *Calculate Penalization for* Chaotic Behaviour, calculate *Smoothness Rating for Trajectory* (def. 18) 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:

$$Cost(Node) = Cost(Node.Trajectory) \times \ldots \\ \cdots \times (1 + (1 - SmoothnessRate(Node.Trajectory))) \quad (6.84)$$

Merge Tree Function $mergeTree(\ldots)$ implements *Outer Join* operation on two trees. Example was given in (eq. 6.83). Function is applied on *root Node* iterating over *Movements in Movement Set*, because *Movement is pivot*.

Algorithm 6.4: Reach Set Merge Function and Combined Reach Set calculation

```
# Tree merge function;
Node mergeTree(Node firstNode, Node secondNode)
    # Try to copy reference node or return null;
    Node referenceNode = (firstNode?:(secondNode?: return null));
    Node merged = new Node(referenceNode);
    merged.leafs= [];
    # Try to fetch movement nodes if exist in any sub tree;
    for movement ∈ Movements do
        firstLeaf = firstNode.getLeafFor(movement);
        secondLeaf = secondNode.getLeafFor(movement);
        newLeaf = mergeTree(firstLeaf,secondLeaf);
        if newLeaf ∼= null then
            merged.leafs.append(newLeaf);
        end
    end
    return merged
# Combined Reach Set calculation function;
Node combinedReachSet(Node root, AvoidanceGrid grid,int+ chaoticSpread,
 int+ harmonicSpread, int+ footprintLength)
    Node chaotic = chaoticReachSet(root,grid, footprintLength,chaoticSpread);
    Node harmonic = harmonicReachSet(root,grid, harmonicSpread);
    Node combined = mergeTree(chaotic,harmonic);
    combined.applyCostFunction();
    return combined
```

Example: for *Avoidance Grid* with *Distance 10 m*, *Layer count 10*, *Horizontal range* $[-45°, +45°]$, *Horizontal Cell Count 7*, *Vertical range* $[-30°, +30°]$, and *Vertical Cell Count 5*. Is given in (fig. 6.8). 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*. The *Chaotic Spread* was set to 8, *Footprint Length* to 3 and *Harmonic Spread* to 1.

Note. Notice there are typical trajectories from both *Harmonic* (fig. 6.7) and *Chaotic* (fig. 6.6) *Reach Set Approximations*.

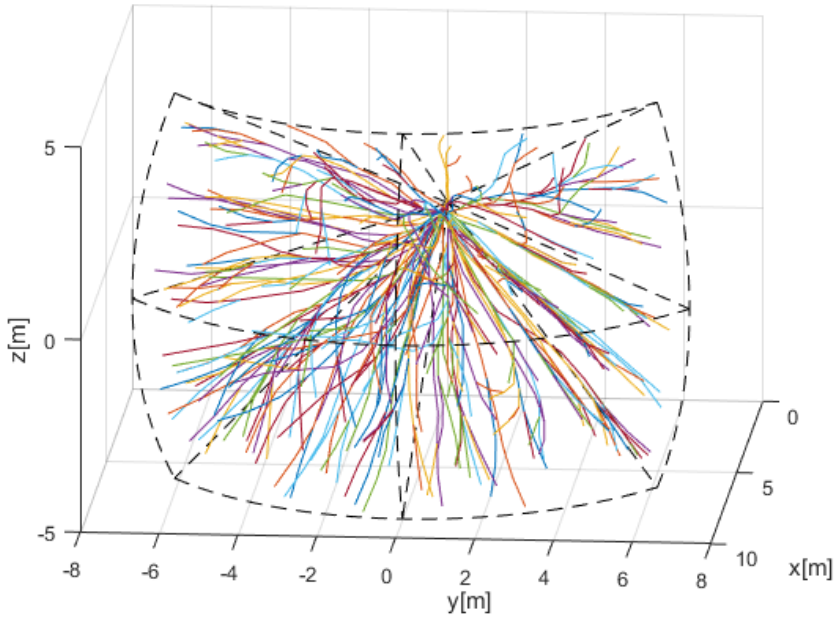


Figure 6.8: *Combined* reach set *approximation*.

Pros and Cons: It can bee seen from example (fig. 6.8) that *Combined Reach Set Approximation* (alg. 6.4) contains both types of maneuvers. *Cheaper Smooth* for navigation and *More Expensive Chaotic* for *Emergency Avoidance*. The upper limit of trajectories is given as follow:

$$countTrajectories(ReachSet) \leq countTrajectories(Chaotic) + countTrajectories(Harmonic) \quad (6.85)$$

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

$$countNodes(ReachSet) \leq countNodes(Chaotic) + countNodes(Harmonic) \quad (6.86)$$

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

6.4.6 ACAS-X like Reach set

Motivation: The implementation of *ACAS-Xu* behavior in DAA system will be mandatory for *National Airspace System Integration* in United spaces [21].

Implementation of ACAS-Xu like behaviour increase usability of approach, if it can be achieved without major concept changes.

Background: The *ACAS-Xu* system on operational level has been described in [22]. The *Policy for Collision Avoidance* proposal has been given in [23].

Some behavioural patterns can be encoded into *Reach Set*. ACAS-Xu navigation part is basically *Look-up table of Maneuvers for Allowed Separations*.

The *Evasive Maneuver* selection process in ACAS-Xu is similar to our approach: *Select most energy efficient maneuver in compliance with space-time constraints*. ACAS-Xu intruder model is similar to our *Body Volume Intersection Model* (sec. 6.6.3). The *ACAS-Xu* defines following base separations:

1. *Horizontal* - movements on *Horizontal Plane* in *Global Coordinate System*.
2. *Vertical* - movements on *Vertical Plane* in *Global Coordinate System*.

There are allowed custom separations which can be used, for further experimentation:

1. *Slash* - movement on $+45°$ *Tilted Plane to Horizontal Plane* in *Global Coordinate System*.
2. *Backslash* - movement on $-45°$ *Tilted Plane to Horizontal Plane* in *Global Coordinate System*.

For given *Movement Automaton* implementation (sec. 6.2) the separations are given as follow:

$$\begin{aligned} Horizontal &= \{Straight, Left, Right\} \\ Vertical &= \{Straight, Up, Down\} \\ Slash &= \{Straight, UpLeft, DownRight\} \\ Backslash &= \{Straight, UpRight, DownLeft\} \end{aligned} \tag{6.87}$$

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:

$$isSeparation(buffer, separation) = \begin{cases} \forall movement \in buffer, & \\ \quad movement \in separation & : true \\ otherwise & : false \end{cases} \tag{6.88}$$

Following *Separation Modes* can be defined with given *separations*:

1. *Horizontal* (ACAS-X defined mode) containing *horizontal* separation.
2. *Vertical* (ACAS-X defined mode) containing *vertical* separation.
3. *Horizontal-Vertical* (ACAS-X defined mode) containing *horizontal, vertical* separations.
4. *Full* (custom defined mode) containing all *Separation Modes*.

Note. Every separation modes generates 2D trajectories set on *Respective plane*. There is no need for *Tuning parameters* for further *Expansion Constraint*.

Expansion Constraint Function Implementation (alg. 6.5) is based on simple principle: *Select only candidate Nodes which Trajectories have at least one desired Separation Mode*.

Step: Initialization sets candidate *Nodes* as empty set, leftover *Nodes* as empty set, and, select all nodes form *stack* which represents *Finishing Trajectories* in working $cell_{i,j,k}$,

Step: Candidate Selection Process is evaluated for each *test Node* from *passing Node Set*.

For each *applicable separation*, given as input parameter *separations*, The test function *isSeparation* (eq. 6.88) 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. 6.87) are not *exclusive sets* in *Movement Automaton* domain. One *Trajectory* contained by Node can belong to multiple *Separations*.

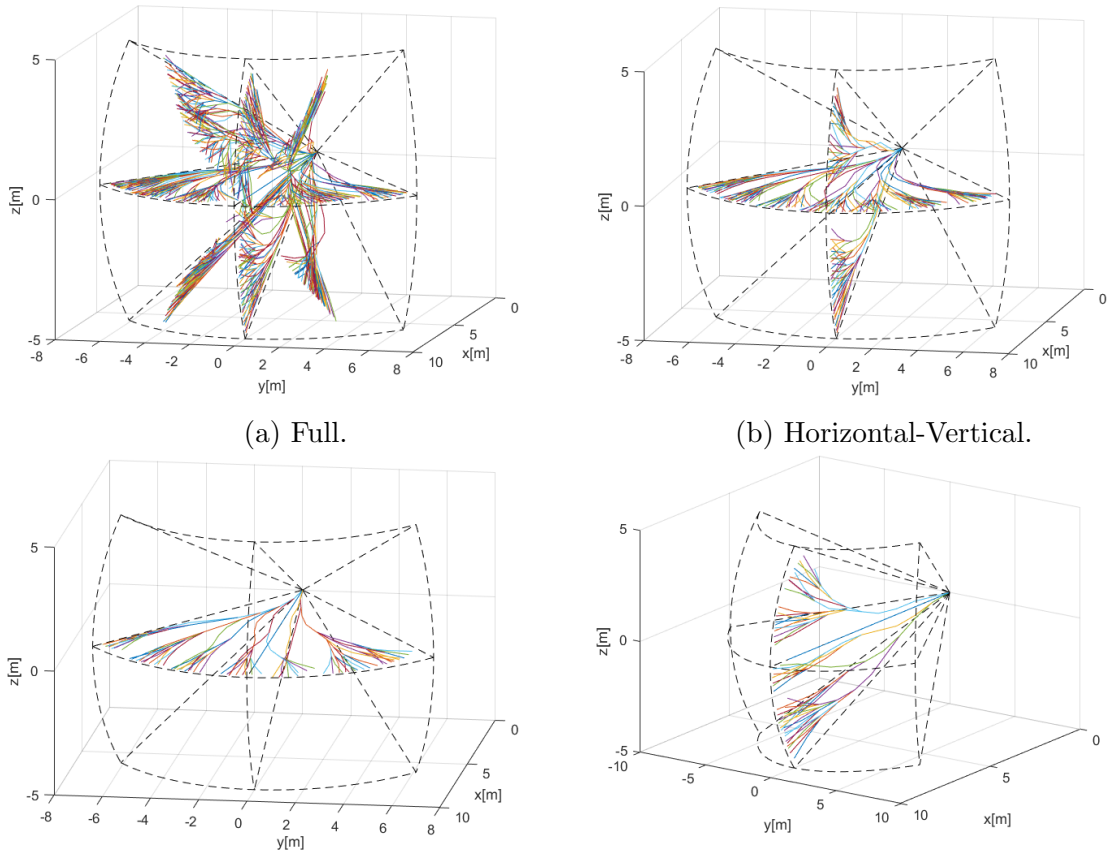
Algorithm 6.5: Expansion Constraint function for *ACAS-like Reach Set Approximation*

Input : Node[] stack, Cell $\text{cell}_{i,j,k}$, 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)(6.88) 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. 6.9). 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*.

Full separation mode given in (fig. 6.9a). *Horizontal-Vertical* separation mode, used in original *ACAS-Xu* testing [22], given in (fig. 6.9b). *Horizontal* separation mode given in (fig. 6.9c) is usually used by planes. *Vertical* separation mode given in (fig. 6.9d) is usually used by copters.



(a) Full.

(b) Horizontal-Vertical.

(c) Horizontal.

(d) Vertical.

Figure 6.9: ACAS-X imitation *reach set* approximation for various *separation modes*.

Pros and Cons: It can be seen from examples (fig. 6.9) that *ACAS-like Reach Set Approximation Method* (alg. 6.5) generates full reach set for 2D plane located in 3D space.

The *Reach Set* contains trajectories with *high coverage ratio* and *high smoothness rating* for selected 2D separation plane. Overall performance compared to full 3D reach sets (sec. 6.4.3, 6.4.4 6.4.5) is poor.

The *node* and *trajectory* count boundary was not implemented. It is common knowledge that *2D* avoidance sets does not require scaling [22]. Otherwise trajectory footprint mechanism like in *Harmonic Reach Set Approximation* (alg. 6.3) can be introduced.

This reach set implements *Planar-Separation* as native feature, it can be used for both *navigation* and *avoidance* tasks in *Controlled Airspace*. For *Non-controlled Airspace* there are far more superior *Combined Reach Set* (sec. 6.4.5).

6.5 Static Obstacles and Constraints

Introduction: The *static obstacles* were used in original concept [24], 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 [25].

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$\partial horizontal^\circ, \mathrm{d}vertical^\circ$ 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).

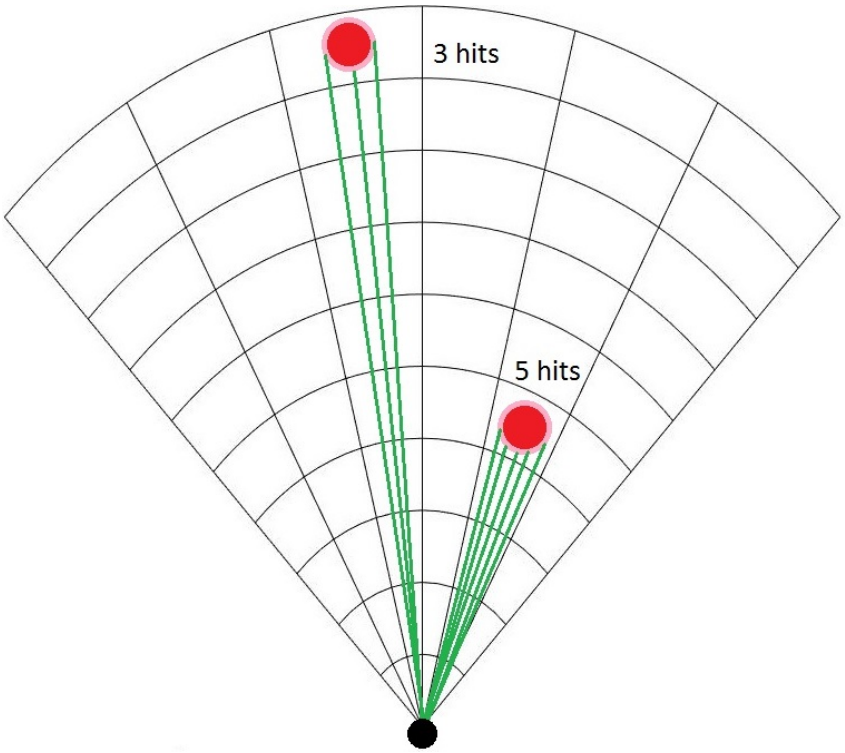


Figure 6.10: Different count of LiDAR hits with different distance from UAS.

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 [24].

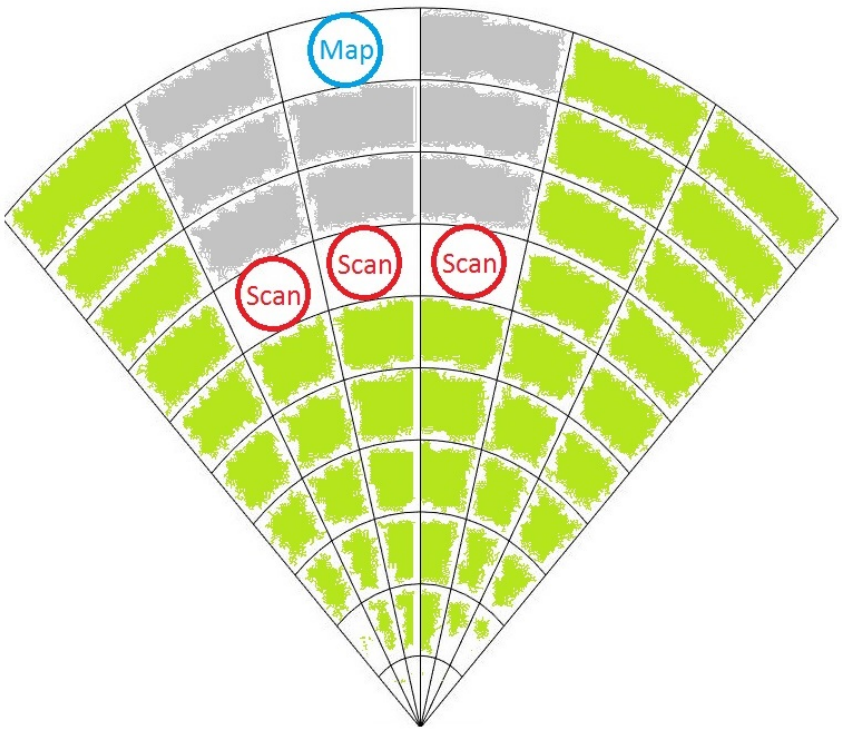


Figure 6.11: Overshadowed map obstacle by detected obstacles.

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.

6.5.1 Detected Obstacles

Idea: The *visibility* inside avoidance grid and *obstacle* probability are interconnected for most ranging sensors (ex. LiDAR). The goal of this section is to introduce *visibility hindrance* concept which includes space uncertainty assessment and detected obstacle processing.

Detected Obstacle Rating: The *detected obstacle rating* defines UAS chances to encounter detected obstacle in avoidance grid $cell_{i,j,k}$. Final *detected obstacle rating* is merged information (eq. 6.157). 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:

$$lidarRays(cell_{i,j,k}) = \left\{ \begin{bmatrix} horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \quad (6.89)$$

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. 6.89) is finite countable and nonempty for any $c_{i,j,k}$, otherwise it will contradict the definition of avoidance grid (def. 11).

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. 6.90) in cell $cell_{i,j,k}$ is defined like follow:

$$lidarHits(cell_{i,j,k}) = \left\{ \begin{bmatrix} distance = lidarScan(), \\ horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} distance \in cell_{i,j,k}.distanceRange, \\ horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \quad (6.90)$$

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

$$obstacle^{LiDAR}_{cell_{i,j,k}} = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \quad (6.91)$$

Note. The *naive obstacle rate* (eq. 6.91) 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. 6.92) rate is naturally defined as supplement to naive obstacle rate. This definition is sufficient, because its reflecting the *remaining sensing capability* of LiDAR.

$$hindrance^{LiDAR}_{cell_{i,j,k}} = 1 - \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \quad (6.92)$$

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. 6.93).

$$Area(cell_{i,j,k}) = \int_{horizontal^\circ_{start}}^{horizontal^\circ_{end}} \int_{vertical^\circ_{end}}^{vertical^\circ_{start}} radius^2 \cos(vertical^\circ) \quad \mathrm{d}vertical^\circ \mathrm{d}horizontal^\circ \quad (6.93)$$

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:

$$\begin{aligned} &Area(radius, horizontalRange, vertical^{\circ}_{start}, vertical^{\circ}_{end}) = \dots \\ &= \begin{cases} vertical^{\circ}_{start} < 0, vertical^{\circ}_{end} \leq 0 : \\ \qquad radius^2(\sin|vertical^{\circ}_{start}| - \sin|vertical^{\circ}_{end}|) \times horizontalRange) \\ vertical^{\circ}_{start} < 0, vertical^{\circ}_{end} > 0 : \\ \qquad r^2(\sin|vertical^{\circ}_{start}| + \sin|vertical^{\circ}_{end}|) \times horizontalRange) \\ vertical^{\circ}_{start} \geq 0 vertical^{\circ}_{end} < 0 : \\ \qquad r^2(\sin vertical^{\circ}_{end} - \sin vertical^{\circ}_{start}) \times horizontalRange) \end{cases} \end{aligned} \tag{6.94}$$

An intersection surface for cell is defined in (eq. 6.94). Area covered by LiDAR hits (eq. 6.95) is defined as LiDAR hit rate (hits to passing rays ratio) multiplied by *Average* cell intersection surface (eq. 6.94).

$$lidarHitArea(cell_{i,j,k}) = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \times Area \begin{pmatrix} radius, horizontalRange, \\ vertical^{\circ}_{start}, vertical^{\circ}_{end} \end{pmatrix} \tag{6.95}$$

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. 6.96) and it removes bias of naive approach (eq.6.91).

$$obstacle(LiDAR, cell_{i,j,k}) = \min\left\{\frac{lidarHitArea(cell_{i,j,k})}{UAS.lidarThresholdArea}, 1\right\} \tag{6.96}$$

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. 6.92). 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. 6.97).

$$cellRow(j_{fix}, k_{fix}) = \left\{ cell_{i,j,k} \in AvoidanceGrid : \begin{matrix} i \in \{1, .., layersCount\}, \\ j = j_{fix}, k = k_{fix} \end{matrix} \right\} \tag{6.97}$$

For each $cell_{i,j,k}$ there exists a function which calculates final visibility hindrance rate. Then for ordered cell row:

$$cellRow(j_{fix}, k_{fix}) = \{cell_{1,j_{fix},k_{fix}}, cell_{2,j_{fix},k_{fix}}, \dots, cell_{layersCount,j_{fix},k_{fix}}\}$$

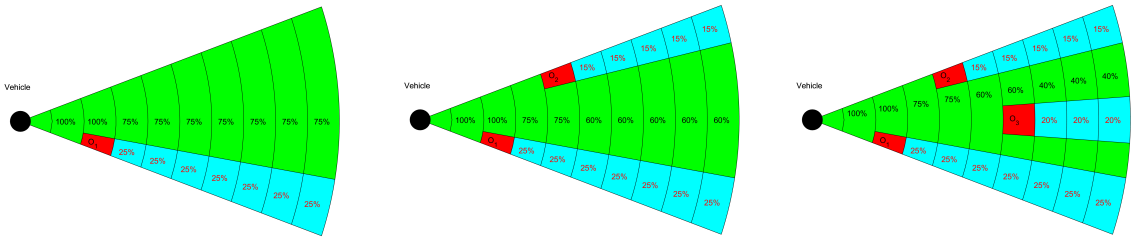
and for one selected $cell_{i,j,k}$ the visibility rate is naturally defined as a supplement to hindrance from previous cells. The visibility is defined in (eq. 6.98).

$$visibility(cell_{i_c,j_c,k_c}) = \ldots$$
$$\ldots = 1 - \sum_{index \in \mathbb{N}^+}^{index<i_c} hindrance(cell_{a,j_c,k_c} : cell_{a,j_c,k_c} \in cellRow(j_c, k_c) \quad (6.98)$$

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:

$$0 \leq \sum_{cell \in cellRow(j_{fix},k_{fix})} visibility(cell) \leq 1 \quad (6.99)$$

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.

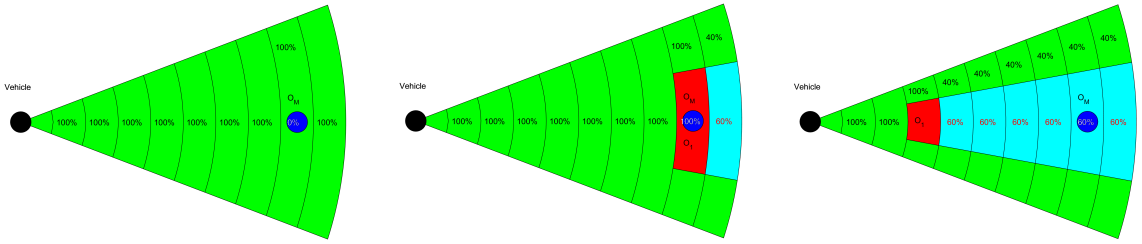
Figure 6.12: Obstacle hindrance impact on visibility in *Avoidance Grid Slice*.

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.

6.5.2 Map Obstacles

Idea: Use *stored LiDAR readings* from previous mission to build an compact obstacle map [26]. 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. 6.7.1).

First we start with obstacle map definition. The obstacle map (eq. 6.100) 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.

$$obstacleMap = \left\{ \begin{bmatrix} position, \\ orientation, \\ safetyMargin, \\ parameters \end{bmatrix} : \begin{array}{l} position \in \mathbb{R}^3(GCF), \\ orientation \in \mathbb{R}^3(GCF), \\ safetyMargin \in \mathbb{R}^+(m), \\ parameters \in \{\dots\} \end{array} \right\} \tag{6.100}$$

The *Map Obstacle* concept is taken from my *master student work* [26], implementing *compact representation* of point-cloud obstacle map. Te example of *cuboid obstacles* with *safe zone* is given in (fig. 6.14).

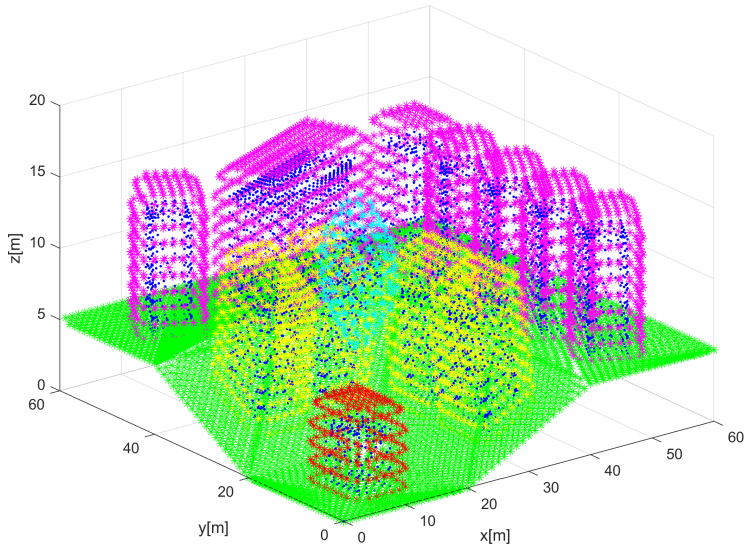


Figure 6.14: Example of Extracted Map Obstacle [26].

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. 6.101).

$$avoidanceGrid.radius \geq distance(UAS.position, mapObstacle) - totalMargin \quad (6.101)$$

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

$$bodyPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapObstacleBody\} \quad (6.102)$$

2. *Safety Margin Point Cloud* - representing safety coating around mapped obstacle body approximation (eq. 6.103). This point cloud is considered as soft constraint.

$$marginPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapSafetyMargin\} \quad (6.103)$$

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. 6.102, 6.103) are homogeneous [26]. That means *each point* in point clouds covers similar portion of object volume. There is *threshold volume* (eq. 6.104) which represents minimal object volume to be considered as an *obstacle*.

$$0 < thresholdVolume \leq \frac{volume(pointCloud)}{|pointCloud|} \quad (6.104)$$

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. 6.105).

$$point.rate = \frac{pointCloudVolume}{thresholdVolume} \times \frac{1}{|pointCloud|} \quad (6.105)$$

The *intersection set* between *point cloud* and $cell_{i,j,k}$ is defined in (eq. 6.105). The *cell* intersection with points is defined in (eq. 6.50).

$$intersection(map, cell_{i,j,k}) = \ldots$$
$$\ldots \{points \in \mathbb{R}^3 : (point \rightarrow AvoidanceGridFrame) \in cell_{i,j,k}\} \quad (6.106)$$

The *map obstacle rating* for $cell_{i,j,k}$ and obstacle for our *information source* is defined in (eq. 6.107).

$$map(cell_{i,j,k}, obstacle) = \max \left\{ \sum_{\forall point \in intersection(map, cell_{i,j,k})} point.rate, 1 \right\} \quad (6.107)$$

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. 6.108).

$$map(cell_{i,j,k} = \max \{map(cell_{i,j,k}, obstacle) : \forall obstacle \in ActiveMapObstacles\} \quad (6.108)$$

Note. The *body point clouds* (eq. 6.102) never intersects, because they are created for inclusive obstacles. The *safety margin point clouds* (eq. 6.103) can intersects, because they represents protection zones around physical obstacles. Therefore the *maximum obstacle rating* (eq. 6.108) needs to be selected.

6.5.3 Static Constraints

Idea: The *constraints* (ex. weather, airspace) usually covers large portion of the *operation airspace.*

Converting constraints into valued *point-cloud* is not feasible, due the *huge amount of created points* and low *intersection rate.* The *polygon intersection* or *circular boundary of 2D polygon* is simple and effective solution [27, 28].

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. 6.109) 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.

$$constraint = \{position, boundary, altitude_{start}, altitude_{end}, safetyMargin\} \quad (6.109)$$

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

The *active constraints set* (eq. 6.110) 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*.

$$\begin{aligned} &ActiveConstraints = \ldots \\ &\cdots = \left\{ \begin{array}{l} constraint \in InformationSource : \\ \qquad distance(constraint, UAS) \leq AvoidanceGrid.distance, \\ \qquad constraint.altitudeRange \cap UAS.altitudeRange \neq \varnothing \end{array} \right\} \end{aligned} \quad (6.110)$$

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* [28] is used.

First the *set of contraint edges* (eq. 6.111) is a enclosed set of 2D edges between neighboring points defined as follow:

$$edges(constraint) = \left\{ \left[point_i, point_j \right] : \begin{array}{l} point \in boundary, \\ i \in \{1, \ldots, |boundary|\}, \\ j \in \{2, \ldots, |boundary|, 1\} \end{array} \right\} \quad (6.111)$$

The *constraint circle boundary* with calculated center on 2D horizontal plane and radius (representing body margin) is defined in (eq. 6.112).

$$circle(constraint) = \begin{bmatrix} center = \dfrac{\sum boundary.point}{|boundary.point|} + correction \\ radius = smallestCircle(edges(constraints)) \end{bmatrix} \quad (6.112)$$

The ($cell_{i,j,k}$ and *constraint* intersection (eq. 6.113) 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 $]radius, radius + safetyMargin]$.

$$
\begin{aligned}
&intersection, constraint) = \ldots \\
&\cdots = \begin{cases}
hard & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) \leq \ldots \\ \quad \cdots \leq circle(constraint).radius, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\
soft & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) > \ldots \\ \quad \cdots > circle(constraint).radius, \\ distance(cell_{i,j,k}, circle(constraint)) \leq \ldots \\ \quad \cdots \leq circle(constraint).radius + safetyMargin, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\
none & : otherwise
\end{cases}
\end{aligned}
\tag{6.113}
$$

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:

$$
cell_{i,j,k}.softConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = soft \end{matrix} \right\} \tag{6.114}
$$

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

$$
cell_{i,j,k}.hardConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = hard \end{matrix} \right\} \tag{6.115}
$$

Note. The final *constraint rate value* (eq. 6.160) is determined based on *mission control run* feed to *avoidance grid* (fig. 6.26) defined in 7^{th} to 10^{th} step.

6.6 Intruders and Moving Constraints

Intruder behaviour: *Adversarial behaviour* of moving obstacle is trying to destroy avoiding our UAS. The *Intruder* UAS [29] 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*.

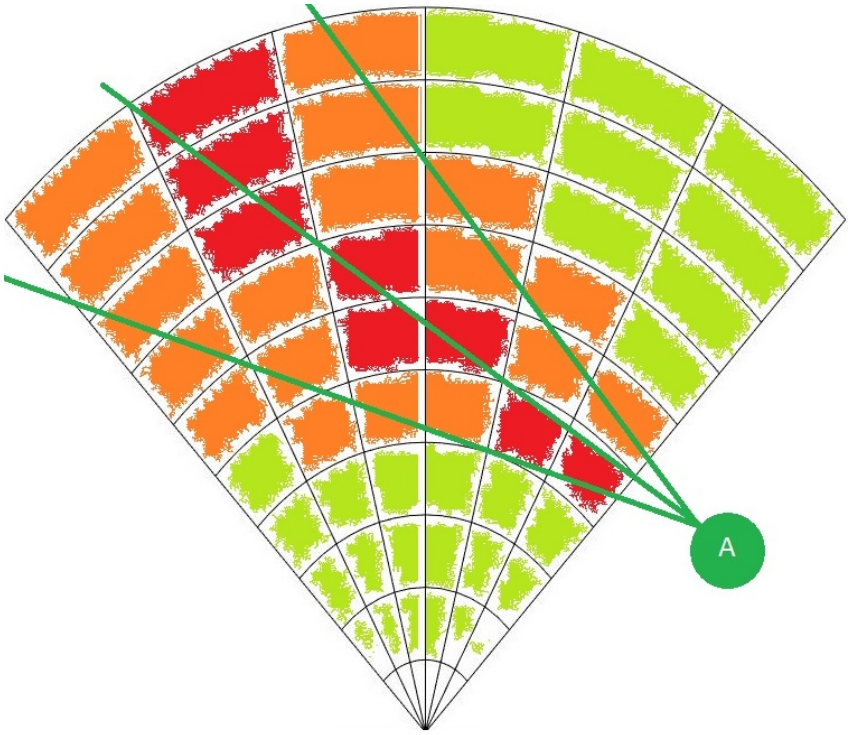


Figure 6.15: Intruder UAS intersection rate along expected trajectory.

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. 11) 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 [30] was solved by application of method [31].
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:[32] the selected algorithm *Shamos-Hoey* [33].

Note. The *Intruder Intersection* models are based on *analytically geometry* for *cones and ellipsoids* taken from [34].

6.6.1 Intruder Behaviour Prediction

Idea: *Intruder Intersection Models* is about space-time intersection of *intruder body* with *avoidance Grid* and *Reach Set*:

1. The *UAS* reach set defines *time boundaries* to *enter/leave* cell in avoidance grid.

2. The *Intruder* behavioral pattern defines *rate* of *space intersection* with cell bounded space in avoidance grid.

The multiplication of *space intersection rate* and *time intersection rate* will give us *intruder intersection* rate for our *UAS* and intruder.

Intruder Dynamic Model: The definition of avoidance grid enforces the most of these methods to be numeric. Let us introduce intruder dynamic model:

$$\mathrm{d}position/\mathrm{d}time = velocity \quad | \quad \begin{aligned} position_x(t) &= position_x(0) + velocity_x \times t \\ position_y(t) &= position_y(0) + velocity_y \times t \\ position_z(t) &= position_z(0) + velocity_z \times t \end{aligned} \tag{6.116}$$

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. 6.50). There is *Trajectory Intersection* function from (def. 13) 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. 6.117), if there is no *passing trajectories* the UAS *entry time* is set to 0.

$$UAS_{entry}(cell_{i,j,k}) = \min \left\{ \begin{matrix} 0, entry(Trajectory, cell_{i,j,k}) : \\ \quad Trajectory \in PassingTrajectories \end{matrix} \right\} \tag{6.117}$$

The UAS *Cell Leave* time is given as maximum of all *passing trajectory segments* entry times (eq. 6.118), if there is no *passing trajectories* the UAS *leave time* is set to 0.

$$UAS_{leave}(cell_{i,j,k}) = \max \left\{ \begin{matrix} 0, leave(Trajectory, cell_{i,j,k}) : \\ Trajectory \in PassingTrajectories \end{matrix} \right\} \tag{6.118}$$

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. 6.117) and leave (eq. 6.118).

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. 6.119).

$$time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} = \circ \end{pmatrix} = \frac{\left| \begin{matrix} [intruder_{entry}(\circ), intruder_{leave}(\circ)] \\ \cap \\ [UAS_{entry}(\circ), UAS_{leave}(\circ)] \end{matrix} \right|}{|[UAS_{entry}(\circ), UAS_{leave}(cell_{\circ})]|} \tag{6.119}$$

Intruder Intersection Rate: The *Intruder Intersection Rate* (eq. 6.120) is calculated as *multiplication* of *space intersection rate* (defined later) and *time intersection rate* (eq. 6.119).

$$intruder \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} = time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \times space \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \tag{6.120}$$

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. 6.121) is bounded to discrete point in intersection model [30, 31]. The intruder *entry/leave time* is calculated similar to *UAS cell entry (eq. 6.117)/leave (eq. 6.118) time.*

$$pointReachTime(Intruder, point) = \frac{distance(Intruder.initialPosition, point)}{|Intruder.velocity} \tag{6.121}$$

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. 6.116), 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.6.116).

   b. *Volume line* - intersection is going along the trajectory line defined by intruder model (eq. 6.116) 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. 6.116) $time \in [0, \infty]$. The cone width is set by horizontal and vertical spread.

6.6.2 Linear Intersection

Idea: There are *small intruders* which have body *smaller* than average $cell_{i,j,k}$ cell size. Its trajectory will stick to *linear trajectory* prediction with high probability.

Space Intersection Rate: The *Space Intersection Rate* for $cell_{i,j,k}$ is implemented as simple point cloud intersection. Where *sufficiently thick* point cloud is defined along *line* (eq. 6.122):

$$position(time) = position(time_0) + velocity \times time, \quad time \in [0, \infty[\tag{6.122}$$

Then there exist projection function from local euclidean coordinates to local polar coordinates (eq. 6.123. The function projects intruder trajectory (eq. 6.122) to planar coordinates [$distance$, $horizontal^\circ$, $vertical^\circ$] as a set of sufficiently thick point cloud.

$$polarSet : position(t) \rightarrow \{[distance, horizontal^\circ], vertical^\circ\} \tag{6.123}$$

The *space intersection rating* $SpaceIntersection(\circ)$ for line type is given as (eq. 6.124). If there exist non empty intersection of $polarSet \cap cell_{i,j,k}$ there is space intersection rate equal to 1, if intersection $polarSet \cap cell_{i,j,k} = \varnothing$ then the rate is zero.

$$space \begin{pmatrix} Intruder, \\ cell_{i,j,k} \end{pmatrix} = \begin{cases} 1: & \exists point \in polarSet(eq.6.123) : point \in c_{i,j,k} \\ 0: & \text{otherwise} \end{cases} \tag{6.124}$$

Note. The *intruder intersection rate* is multiplication of *space intersection rate* and time intersection rate. The *intersection rate* is calculated for *every intruder* and *selected intersection model* separately.

6.6.3 Body-volume Intersection

Idea: The *Intruder* has body volume greater than *average* $cell_{i,j,k}$ volume. The *intruder body* is considered as the ball moving along *intruder position*. The *intersection* of the intruder body is realized as sufficiently thick *point-cloud intersection*.

Space Intersection Rate - Body Volume: The *body volume mass* with center at $position(t)$ is moving along intruder trajectory prediction (eq. 6.125) in time interval $[0, \infty[$:

$$position(time) = position(time_0) + velocity \times time \tag{6.125}$$

The body *Volume ball* $Body(position(t), radius)$ (eq. 6.126) is defined as set of points in $\mathbb{R}^3$ euclidean space. The center is moving along the *position(t)*. The body *volume ball* is a set of points sufficiently thick including also inner points. The *thickness* is guaranteed by existence of neighbour point which is close enough.

$$Body(position(t), radius) = \left\{ point \in \mathbb{R}^3 : \begin{matrix} \|position(t) - point\| \leq radius \\ \forall point_i \exists point_{j \neq i}, \\ distance(point_i, point_j) \leq thickness \end{matrix} \right\} \tag{6.126}$$

The *polar volume ball* $polarBody$ (eq. 6.127) is projection of body volume ball set $Body(position(t), radius)$ to a set of planar coordinates in avoidance grid coordinate frame:

$$polarBall(t) : Body(position(t), radius) \rightarrow \left\{ \begin{bmatrix} distance, horizontal^\circ, \\ vertical^\circ, intersectionTime \end{bmatrix} \right\} \tag{6.127}$$

The *space intersection rate for vehicle body* $space(Intruder, cell_{i,j,k})$ (eq. 6.128) is calculated as intersection of polar body volume ball and $cell_{i,j,k}$. If intersection is non empty then base probability is one, zero otherwise:

$$space \begin{pmatrix} Intruder, \\ cell_{i,j,k} \end{pmatrix} = \begin{cases} 1: & \exists point \in polarBall(eq.6.127) : point \in c_{i,j,k} \\ 0: & \text{otherwise} \end{cases} \tag{6.128}$$

Intersection Time: The *intersection time* id depending on point cloud (eq. 6.127) where each point *have intersection time* given as *body-center position* time (eq. 6.125).

Note. The *body-volume* intersection model, can insert the *multiple intersection times* into one $cell_{i,j,k}$. the *interval length* considers all of these for intersection rates (eq. 6.119).

6.6.4 Maneuverability Uncertainty Intersection

Idea: The *intruders* are not bullets they are not sticking to predicted linear paths. The *intruder* maneuverability is given as horizontal and vertical spread. Therefore *intruder reach set* will form a *elliptic cone*. This cone can be transformed into *finite discrete* point-cloud, each *point* should have assigned *severity* impact value. The point cloud intersection with *Avoidance Grid* will give us space impact of *uncertain* intruder.

Note. Following section will use condensed notation, due the equation complexity. The *terminology* is consistent with rest of section.

Sprace Intersection Rate - Body Volume Intersection: $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ computation is less straight-forward than other space intersection rates. First let us define the linear intruder i_k positions x at time t (eq. 6.129) model, where $x(t)$ defines intruder position in *avoidance grid euclidean coordinate frame* at time t_i, v defines intruder velocity, and t is time offset.

$$x(t) = x_s + v_I.t \tag{6.129}$$

Intruder *horizontal spread* θ and *vertical spread* φ are introduced. These spreads represents intruder deviation limits along from linear trajectory prediction $x(t) \in \mathbb{R}^3$. The example is given by (fig. 6.16) where the intruder starts at point x_s with fixed velocity v, the linear trajectory prediction is outlined by blue line. The *predicted intruder position* at time $t = 10s$ is given by $x(10)$ (blue point). The ellipsoidal space $E(x)$ is projected on the plane $D(x(t))$. The plane D (eq. 6.130) for point $x(t)$ and velocity v is defined as an orthogonal plane to velocity vector $v \in \mathbb{R}^3$ with origin at intruder position $x(t)$.

$$D(x(t), v) = \left\{ a \in \mathbb{R}^3 : (a - x(t)) \perp v, \right\} \tag{6.130}$$

To construct ellipsoidal space boundary on orthogonal plane $D(x(t), v)$ some parameters are defined in (eq. 6.131). The *scalar distance* $d_d x(t)$ is simple euclidean norm, *maximal horizontal offset* $d_\theta(x_t)$ is given as product of sinus of horizontal offset angle θ and scalar distance d_d, and *maximal vertical offset* $d_\varphi(x(t))$ is given a product of sinus of vertical offset angle φ and scalar distance d_d.

$$\begin{aligned} d_d &= d_d(x(t), x_s) = \|x(t) - x_s\|_2 \\ d_{\theta_{\max}} &= d_\theta(x(t)) = \sin\theta(i_k).d_d(x(t)) \\ d_{\varphi_{\max}} &= d_\varphi(x(t)) = \sin\varphi(i_k).d_d(x(t)) \end{aligned} \tag{6.131}$$

The *Ellipsoid* $E(x(t), v)$ (eq. 6.132) for fixed intruder position $x(t)$ and fixed intruder velocity v is given as constrained portion of orthogonal plane $D(x(t), v)$. The constraint is defined by an internal coordinate frame $p \in \mathbb{R}^2$ which is space reduction of plane $D(x(t), v)$.

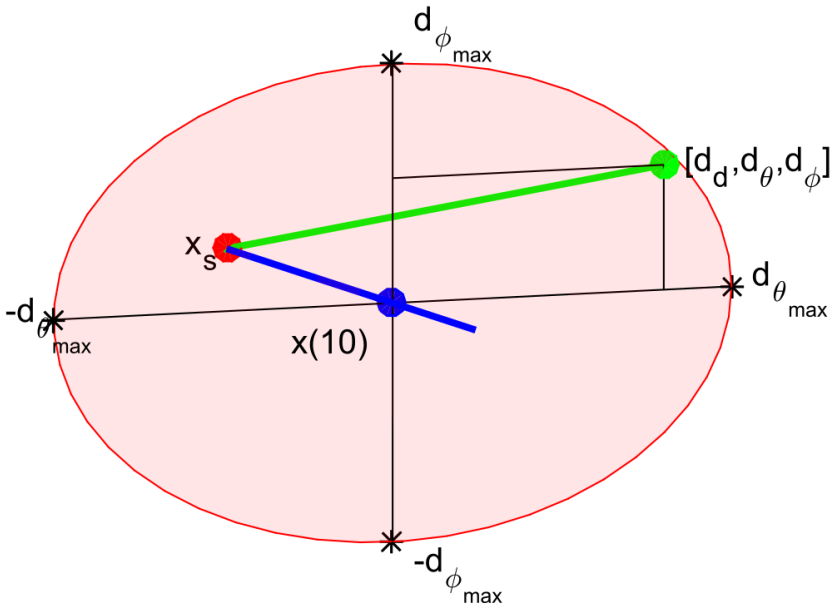


Figure 6.16: One rate position $[d_d, d_\theta, d_\varphi]$ (green). deviated from linear trajectory (blue line) at point $x(10)$.(blue) with initial position x_s (red)

The internal coordinate frame $p \in \mathbb{R}^2$ has origin in $x(t) \to \mathbb{R}^2$. The points of plane p are bounded by projection $p = (b - x(t)) \to \mathbb{R}^2$, where $b \in D(x(t), v)$. The point of ellipsoidal p is then given as standard ellipse boundary with vertical span $d_\theta(x(t))$ and horizontal span $d_\varphi(x(t))$.

The 2D *Ellipsoid* $E(x(t), v)$ for specific time $t = 10s$ example is portrayed as red ellipsoid (in fig. 6.16).

$$E(x(t), v) = \left\{ \begin{array}{c} b \in \mathbb{R}^3 : b \in D(x(t), v), p = (b - x(t)) \to \mathbb{R}^2, \\ \left(\frac{p(1)^2}{d_\theta(x(t))^2} + \frac{p(2)^2}{d_\varphi(x(t))^2} \right) \leq 1 \end{array} \right\} \tag{6.132}$$

The expected behaviour of an intruder i_k is to stick to predicted linear trajectory $x(t)$ (6.129). The probability of deviation should be decreasing with distance from ellipse center (fig. 6.17.).

Probability density function for ellipsoid $E(x(t), v)$defined in (eq. 6.132) is depending on maximal horizontal spread $d_\theta(x(t))$, maximal vertical spread $d_\varphi(x(t))$, defined by (eq. 6.131).

Two standard probabilistic distributions are established $\mathcal{N}(\mu_\theta, \sigma_\theta)$ (eq. 6.133) for horizontal spread $\theta(x(t))$ and $\mathcal{N}(\mu_\varphi, \sigma_\varphi)$ (eq. 6.134) for vertical spread $\varphi(x(t))$. The means μ_θ and μ_φ are set to zero, and internal coordinate frame $p \in \mathbb{R}^2$ where $x(t) \to \mathbb{R}^2$ is frame center. The variances σ_θ and σ_φ are set as maximal distances on horizontal/vertical spread axes $d_\theta(x(t))$ and $d_\varphi(x(t))$.

$$P(x(t), d_\theta) = \mathcal{N}(\mu_\theta, \sigma_\theta) = \mathcal{N}(0, d_\theta(x(t))) \tag{6.133}$$

$$P(x(t), d_\varphi) = \mathcal{N}(\mu_\varphi, \sigma_\varphi) = \mathcal{N}(0, d_\varphi(x(t))) \tag{6.134}$$

The combined *probability density function* for maximal spreads d_θ and d_φ is given by (eq. 6.135). Because probability density function is defined for internal space $p \in \mathbb{R}^2$ and one may need to calculate impact rate for cell space $c_{i,j,k} \in \mathbb{R}^3$.

The reduction from two parameter probability distribution function to scalar rate distribution function is needed. An scalar rate distribution function $P(x(t), d_\theta, d_\varphi)$ over ellipsoid $E(x(t), v)$ is defined as (eq.6.135), where final rate is given as average of two partial probabilities.

Final space intersection rate $P(x(t), d_\theta, d_\varphi)$ needs to be normalized to hold *normal distribution condition* (eq. 6.136). Normal distribution condition value (eq. 6.136) is given as surface integral over ellipsoid $E(x(0), v)$ with rate distribution function $P(x(t), d_\theta, d_\varphi)$.

$$P(x(t), d_\theta, d_\varphi) = \frac{\mathcal{N}(\mu_\theta, \sigma_\theta) + \mathcal{N}(\mu_\varphi, \sigma_\varphi)}{2} \tag{6.135}$$

$$\iint_{E(x(\tau))} P(x(t), d_\theta, d_\varphi)\, \mathrm{d}d_\theta\, \mathrm{d}d_\varphi = 1 \tag{6.136}$$

Final space intersection rate $P(x(t), c_{i,j,k}, \theta, \varphi)$ (space portion, time portion is calculated in (eq.6.120) is given by (eq. 6.138). Its mean value of all intersection rates $P(x(\tau), c_{i,j,k}, \theta, \varphi)$ where $\tau \in [i_e(c_{i,j,k}), i_l(c_{i,j,k})]$ is fixed point in intersection time interval.

An $P(x(\tau), c_{i,j,k}, \theta, \varphi)$ (6.137) is integration of rate density function $P(x(\tau), d_\theta, d_\varphi)$ (eq. 6.135) in surface $E(x(\tau), v)$ to cell $c_{i,j,k}$ volume intersection.

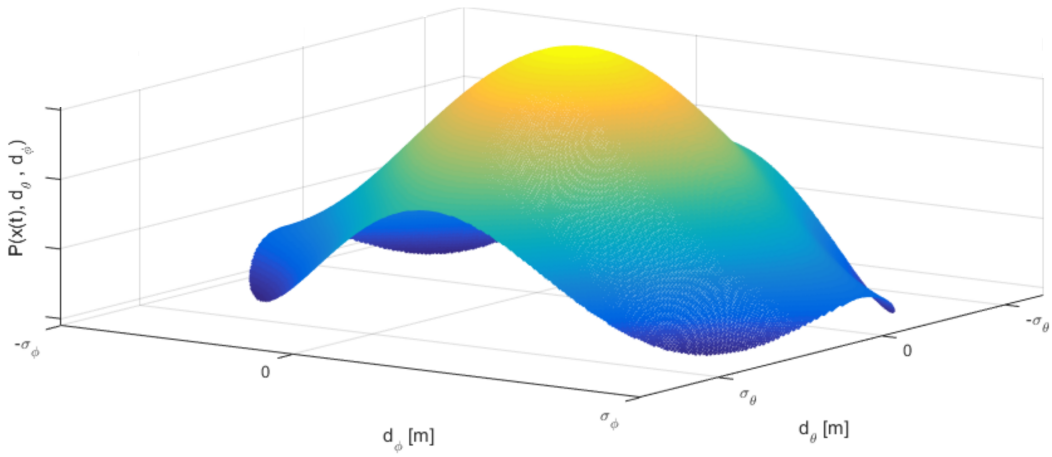


Figure 6.17: Probability of intruder i_k position in ellipsoid $E(x(t), v)$

To get a volume integration partial rate in surface intersection must be integrated and normalized in time interval $\tau \in [i_e(c_{i,j,k}), i_l(c_{i,j,k})]$, the *base intersection probability* $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ is given by (eq. 6.138). Example of intersection of intruder i_r uncertain ellipsoid cone with avoidance grid $\mathcal{A}(t_i)$ is given in (fig. 6.18).

$$P(x(\tau), c_{i,j,k}, \theta, \varphi) = \iint_{E(x(\tau),v)\cap c_{i,j,k}} P(x(\tau), d_\theta, d_\varphi) \tag{6.137}$$

$$P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \frac{\int_{i_e(c_{i,j,k})}^{i_l(c_{i,j,k})} P(x(\tau), c_{i,j,k}, \theta, \varphi)\,\mathrm{d}\tau}{i_l(c_{i,j,k}) - i_e(c_{i,j,k})} \tag{6.138}$$

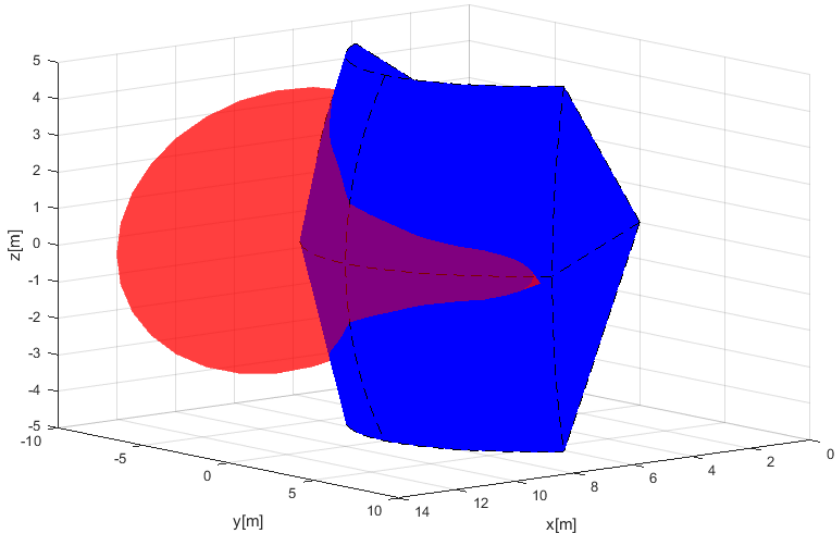


Figure 6.18: Avoidance grid $\mathcal{A}(t_i)$ (blue) intersection with elliptic cone intruder $i_k(x, v, \theta, \varphi)$ (red) example.

An *numeric approximation* of space intersection rate $P_T(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ is more implementation feasible than symbolic calculation due the multiple intersection constraints and bad intersection algorithm complexity.

Let us define homogeneous discrete subset of real numbers $\mathcal{R}$ which is non empty subset of real numbers $\mathbb{R}$. The set $\mathcal{R}$ (eq. 6.139) is homogeneous, that means for any equal interval $(i, i+1], i \in \mathbb{Z}$ subset the count of members is equal to some positive natural number k. The parameter k can be understand as *unit approximation density.*

Similarly the power sets $\mathcal{R}^2 \subset \mathbb{R}^2$, $\mathcal{R}^3 \subset \mathbb{R}^3$, ... $\mathcal{R}^i \subset \mathbb{R}^i, i \in \mathbb{N}^+$ keeps homogeneous distribution.

$$\mathcal{R} = \left\{ \begin{array}{c} a \in \mathbb{R} : \forall i \in \mathbb{Z}, |i < a \leq i+1| = k, k \in \mathbb{N}^+, \\ \forall j \in \mathbb{N}^+ a_{j+1} - a_j = m, m \in \mathbb{R}^+ \end{array} \right\}, \mathcal{R} \subset \mathbb{R} \tag{6.139}$$

The orthogonal plane for $x(t), v, t \in \mathbb{R}$ is defined by (eq. 6.130). The orthogonality property is also kept for any subspace $\mathcal{R}^n \in \mathbb{R}^n, n \in \mathbb{N}^+$. Numeric approximation of $D(x(t), v)$ is given as $D_D(x(t), v)$ (eq. 6.140).

The only difference is that discrete approximation is countable $|D_D| = m, m \in \mathbb{N}^+$, but continuous representation $|D| \approx \infty$ is uncountable. Because ellipsoid is subset of orthogonal plane it keep its countability property, therefore E_D is also countable and

must contains at-least one member.

$$D_D(x(t), v) = \left\{ a \in \mathcal{R}^3 : (a - x(t)) \perp v, \right\}, t \in \mathcal{R} \tag{6.140}$$

The *base ellipsoid* $E(x(t), v)$ for continuous-space is given by (eq. 6.132). Every element, expect the base of internal projection $\mathcal{R}^2$ and orthogonal plane D_D is same in discrete case $E_D(x(t), v)$ (eq. 6.141).

$$\bar{E}_D(x(t), v) = \left\{ \begin{matrix} b \in \mathcal{R}^3 : b \in D_D(x(t), v), p = (b - x(t)) \rightarrow \mathcal{R}^2, \\ \left(\frac{p(1)^2}{d_\theta(x(t))^2} + \frac{p(2)^2}{d_\varphi(x(t))^2} \right) \leq 1 \end{matrix} \right\}, t \in \mathcal{R} \tag{6.141}$$

The *numeric calculation disproportion* can occur in case that ellipsoid $\bar{E}_D(x(t), v)$ (6.141) in case of $d_\theta(x(t)) \approx 0$ and $d_\varphi(x(t)) \approx 0$. The count of ellipsoid members can be $|\bar{E}_D(x(t), v)| = 0$, which is in contradiction with assumption $|\bar{E}_D(x(t), v)| \neq 0$.

Let assume for discrete times $\tau = \{t_1, t_2, \ldots, t_i\}$, $i \in \mathbb{N}^+$ there exists ellipsoids $\bar{E}_D(x(t_1), v)$,$\bar{E}_D(x(t_1), v)$, $\ldots$, $\bar{E}_D(x(t_i), v)$ which are non empty and in space $\mathcal{R}^2$ in internal coordinate frame and space $\mathcal{R}^3$ in avoidance grid $\mathcal{A}(t_i)$ coordinate frame. The intersection of these partial ellipsoids in both spaces is equal to:

$$\bar{E}_D(x(t_1), v) \cap \bar{E}_D(x(t_2), v) \cdots \cap \ldots \bar{E}_D(x(t_i), v) = \varnothing \tag{6.142}$$

An *empty intersection* enables us to keep homogeneity property of ellipsoids by adding points so it is safe to add specific point $x(t)$ into empty ellipsoid. But only one, because it does not impact probability density functions $\mathcal{N}(\mu_\theta, \sigma_\theta)$ and $\mathcal{N}(\mu_\varphi, \sigma_\varphi)$, neither space intersection rate density function $P(x, d_\theta, d_\varphi)$.

The final ellipsoid used forward $E_D(x(t), v)$(eq. 6.143) is keeping all properties of ellipsoid $E(x(t), v)$ (eq. 6.143).

$$E_D(x(t), v) = \begin{cases} |\bar{E}_D(x(t), v)| = 0 & : \{x(t)\} \\ |\bar{E}_D(x(t), v)| \geq 0 & : \bar{E}_D(x(t), v) \end{cases} \tag{6.143}$$

The normal distribution condition for rate distribution function $P_D(x(t), d_\theta, d_\varphi, p)$, which is instance of to rate density function $P(x(y), d_\theta, d_\varphi)$ (eq. 6.135) is used. This rate distribution must be normalized according to (eq. 6.144).

$$\sum_{p \in E_D(x(t))} P_D(x(t), d_\theta, d_\varphi, p) = 1, \forall t \in \mathcal{R}^+ \tag{6.144}$$

The equations for *space intersection rate* are similar to (eq. 6.137, 6.138). For cell $c_{i,j,k}$ there exist intruder entry time $i_e(c_{i,j,k})$ its the earliest intersection with ellipsoid $E_D(x(i_e(c_{i,j,k}))), v$. Same situation occurs with intruder leave time $i_l(c_i, j, k)$. Because E_D is countable set, it means additional attributes can be attached to each point $p \in E_D$.

Based on system dynamic (eq. 6.116) the *Time Of Arrival* (TOA) can be calculated. The example of TOA is given in fig. 6.19.

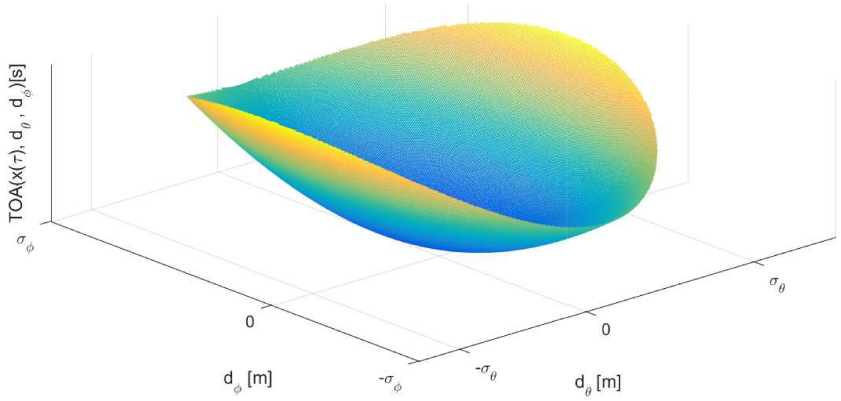


Figure 6.19: Time Of Arrival (TOA) for one ellipsoid $E_D(x(\tau), v)$.

The intersection rate $P_D(x(\tau), c_{i,j,k}, \theta, \varphi)$ for one time sample τ is given by (eq. 6.145), which has similar notation to (eq. 6.137), sums are used instead of integrals and discrete rate density function $P_D(x(\tau), d_\theta, d_\varphi, p)$ for points form ellipse and cell intersection are used as iterator base set $p \in \{E_D(x(\tau), v) \cap c_{i,j,k}\}$.

$$P_D(x(\tau), c_{i,j,k}, \theta, \varphi) = \sum_{p \in \left\{E_D(x(\tau),v) \cap c_{i,j,k}\right\}} P_D(x(\tau), d_\theta, d_\varphi, p) \tag{6.145}$$

The *space intersection rate* $P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.146) is given as mean intersection rate of partial intersections $P_D(x(\tau), c_{i,j,k}, \theta, \varphi)$ where step set $T = \{ i_e(c_{i,j,k}), \ldots, i_l(c_{i,j,k})\}$ contains all viable intersection times with ellipsoids $E(x(\tau \in T), v)$. The denominator is basically count of samples in sample time set T.

$$P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \frac{\sum_{\tau=i_e(c_{i,j,k})}^{i_l(c_{i,j,k})} \sum_{p \in E_D(x(\tau),v)} P_D(x(\tau), c_{i,j,k}, \theta, \varphi, p)}{\sum_{\tau i_l(c_{i,j,k})}^{i_e(c_{i,j,k})} 1} \tag{6.146}$$

An *intersection of intruder cone and cell* $c_{i,j,k}$ cell is defined by (eq. 6.147) The set of point $p \in \mathcal{R}^3$ where condition of intersection between ellipsoids $E_D(x(\tau), v)$ for times $\tau \in \mathcal{R}^+$ and cell space $c_{i,j,k}$ is met.

$$\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k}) = \bigcup_{\forall \tau \in \mathcal{R}^+} \left\{p \in \mathcal{R}^3 : p \in c_{i,j,k} \cap E_D(x(\tau), v)\right\} \tag{6.147}$$

An *intruder time of entry* $i_e(i_k, c_{i,j,k})$ (eq. 6.148), for intruder i, k and cell $c_{i,j,k}$ is approximated for discrete point set $\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.147) as minimal time of arrival $t_{TOA}(p)$ of member points p.

$$i_e(i_k, c_{i,j,k}) \approx \min \left\{t_{TOA}(p) : p \in \mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})\right\} \tag{6.148}$$

An *intruder time of leave* $i_l(i_k, c_{i,j,k})$ (eq. 6.149), for intruder i, k and cell $c_{i,j,k}$ is approximated for discrete point set $\mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.147) as maximal time of arrival $t_{TOA}(p)$ of member points p.

$$i_l(i_k, c_{i,j,k}) \approx \max\{t_{TOA}(p) : p \in \mathcal{P}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})\} \tag{6.149}$$

Combined intersection model: The *combined intersection model* $P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau)$ is defined for intruder i_k with parameters:

1. *Starting position* x_s - expected position of intruder i_r in 3D space at time of avoidance t_i in avoidance grid frame $\mathcal{A}(t_i)$.

2. *Velocity vector* v - oriented velocity of intruder i_r at time of avoidance t_i in avoidance grid frame $\mathcal{A}(t_i)$.

3. *Horizontal uncertainty spread* θ - defines how much can intruder i_r deviate on horizontal axis of intruder local coordinate frame (if X+ is main axis, then Y is horizontal axis in right-hand euclidean coordinate frame), due the properties of intersection definition, the horizontal uncertainty spread can have following values $\theta \in [0, \pi/2]$.

4. *Vertical uncertainty spread* φ -defines how much can intruder i_r deviate on vertical axis of intruder local coordinate frame (if X+ is main axis in local right-hand euclidean intruder coordinate frame, then Z is horizontal vertical axis), due the intersection definition, the vertical uncertainty spread can have following values $\varphi \in [0, \pi/2]$.

5. *Body volume radius* r - defines the body volume of intruder in meters and it is having $\mathbb{R}^+$ value.

The *flag vector* $l, b, s, \tau \in \{0, 1\}$ is parametrization of rate calculation: l stands for *lined intersection*, b stands for *body intersection*, s stands for *spread intersection*, τ stands for *time account*.

The *space intersection for line* $P_L(i_k, c_{i,j,k})$ is defined as $P_T(i_k(x, v), c_{i,j,k})$, where i_k is intruder with properties of initial position x, velocity vector v and $c_{i,j,k}$ is target cell. (eq. 6.124).

The *space intersection rate for body volume* $P_B(i_k, c_{i,j,k})$ is defined as $P_T(i_k(x, v, r), c_{i,j,k})$ (eq. 6.128), where intruder i_r has additional property of the intruder body volume radius r.

The *space intersection probability for maneuverability uncertainty* $P_S(i_k, c_{i,j,k})$ is defined as $P_{TD}(i_k(x_s, v, \theta, \varphi), c_{i,j,k})$ (eq. 6.146), where intruder properties θ, φ stands for intruder horizontal and vertical uncertainty spread.

The *time intersection rate* $P_{\tau,x}(i_k, c_{i,j,k}) \in [0,1]$ is defined in (eq. 6.120). This probability has two calculation modes, first is for 1D intersection (line), second is for volume intersection (body volume, spread elliptic cone).

UAS cell entry time t_e and cell leave time t_l time for vehicle in avoidance grid $\mathcal{A}(t_i)$ are given by (eq. 6.117) and (eq. 6.118).

Intruder leave and entry time for 1D intersections is trivial and is omitted in this section. Intruder entry i_e and intruder leave i_l for 3D intersection are given by (eq. 6.148, 6.149).

All partial rates with respective definition references are summarized in (eq. 6.150)

$$\begin{aligned} P_L(i_k, c_{i,j,k}) &= P_T(i_k(x,v), c_{i,j,k}) && (6.124)\\ P_B(i_k, c_{i,j,k}) &= P_T(i_k(x,v,r), c_{i,j,k}) && (6.128)\\ P_S(i_k, c_{i,j,k}) &= P_{TD}(i_k(x_s,v,\theta,\varphi), c_{i,j,k}) && (6.146)\\ P_{\tau,x}(i_k, c_{i,j,k}) &= \frac{\|[i_e(c_{i,j,k}), i_l(c_{i,j,k})] \cap [t_e, t_l]\|}{\|[t_e, t_l]\|} && (6.120) \end{aligned} \tag{6.150}$$

With definition of all space and time intersection rates (eq. 6.150) and given flag vector $l, b, s, \tau \in \{0,1\}$ one can formulate combined intersection rate $P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau)$ (eq. 6.151) for intruder i_k and cell $c_{i,j,k}$. The principle is following: *maximum of selected rates product based on flag vector is final intersection rate of intruder i_k in cell.*

The time-use flag τ is adding time intersection rate $P_{\tau,x}(i_k, c_{i,j,k})$, where time intersection rate is defined by $x = \{L, B, S\}$ for line, body volume, spread ellipse time intersections ($P_{\tau,L}(i_k, c_{i,j,k}) \neq P_{\tau,B}(i_k, c_{i,j,k}) \neq P_{\tau,B}(i_k, c_{i,j,k})$ for one intruder i_k).

$$P_{O_I}(i_k, c_{i,j,k}, l, b, s, \tau) = \begin{cases} \tau = 0 & : \max \left\{ \begin{matrix} P_L(i_k, c_{i,j,k}).l \\ P_B(i_k, c_{i,j,k}).b \\ P_S(i_k, c_{i,j,k}).s \end{matrix} \right\} \\ \tau = 1 & : \max \left\{ \begin{matrix} P_{\tau,L}(i_k, c_{i,j,k}).P_L(i_k, c_{i,j,k}).l \\ P_{\tau,B}(i_k, c_{i,j,k}).P_B(i_k, c_{i,j,k}).b \\ P_{\tau,S}(i_k, c_{i,j,k}).P_S(i_k, c_{i,j,k}).s \end{matrix} \right\} \end{cases} \tag{6.151}$$

6.6.5 Moving Constraints

Idea: The basic ideas is the same as in case *static constraints* (sec. 6.5.3). There is horizontal constraint and altitude constraint outlining the constrained space. The only additional concept is moving of *constraint* on horizontal plane in global coordinate system.

The constraint intersection with *avoidance grid* is done in *fixed decision Time*, for cell in *fixed cell leave time* (eq. 6.118), which means concept from static obstacles can be fully reused.

Definition: The *moving constraint definition* (eq. 6.152) covers minimal data scope for moving constraint, assuming linear constraint movement.

The original definition (eq. 6.109) 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.

$$constraint = \{position, boundary, \ldots \\ \ldots, velocity, detectionTime, \ldots \\ \ldots altitude_{start}, altitude_{end}, safetyMargin\} \quad (6.152)$$

Cell Intersection: The *intersection algorithm* follows (eq. 6.113), only shift of the *center and boundary points* is required.

First let us introduce $\Delta time$ (eq. 6.153), which represents difference between the constraint detection time and expected cell leave time (eq. 6.118).

$$\Delta time = UAS_{leave}(cell_{i,j,k}) - detectionTime \quad (6.153)$$

The constraint boundary is shifted to:

$$shiftedBoundary(constraint) = \{newPoint = point + velocity \times \Delta time : \ldots \\ \ldots \forall point \in constraint.boundary\} \quad (6.154)$$

The constraint center is shifted to:

$$shiftedCenter(constraint) = constraint.center + velocity \quad (6.155)$$

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 [32], the selected algorithm is *Shamos-Hoey* [33].

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

6.7 Avoidance Concept

This section introduces *Platform Independent Avoidance Concept* core functionality (fig. 6.2) modules responsible for *path finding* and *navigation* including *data fusion* interface. The sections are organized like follow:

1. *Data Fusion* (sec. 6.7.1) - implementation details of *input interface* responsible for *processing partial known world data* into final visibility, obstacle, intruder, and, constraints ratings.

2. *Avoidance Grid Run* (sec.6.7.2) (inner avoidance run) - the *best path finding* in one *Avoidance Grid* with *situation assessment* done.

3. *Mission Control Run* (sec . 6.7.3) (outer navigation run) - main navigation and decision making algorithm for *non-cooperative obstacle avoidance*.

4. *Computation Complexity* (sec. 6.7.4) - the *computational feasibility study* and *weak point identification* of our approach.

5. *Safety Margin Calculation* (sec. 6.7.5) - the boundaries of *Safety Margin* and identified *impact factors*.

6.7.1 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 Ratings* commutation. Then these ratings will be discredited into Boolean values as properties of *Avoidance Grid/Trajectory*. Then these Boolean values will be used for further classification of space into *Free(t), Occupied(t), Restricted(t)* and *Uncertain(t)*.

All mentioned ratings are result of *Filtered Sensor Readings* from *Sensor Field* and *Information Sources* with prior processing. This section will focus on *final fuzzy value calculation* and *discretization*.

Note. All rating values are in *range:* $[0, 1]$ and they were introduced in previous sections.

Visibility: The *sensor reading* of *sensor* if *Sensor field* returns a value of *visibility* for cell space in time of decision t_i.

The *visibility* for cell is given in (eq. 6.156) as minimal visibility calculated from all capable sensors in *Sensor Field*.

$$visibility(cell_{i,j,k}) = \min \left\{ \begin{matrix} visibility(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \tag{6.156}$$

The example of *visibility* calculation for *LiDAR* sensor is given in (sec. 6.5.2).

Note. Sensor reliability for *visibility* is already accounted prior *data fusion*. If not *weighted average* should be used instead.

Detected Obstacle: The *physical obstacles* are detected by *sensors* is *Sensor Field*. Each *sensor* returns *detected obstacle rating* in range $[0, 1]$ reflecting the probability of obstacle occurrence in given cell.

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

$$obstacle(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \times \dots \\ \dots \times visibility(cell_{i,j,k}) \quad (6.157)$$

The example of *detected obstacle rating* calculation for *LiDAR* sensor is given in (sec. 6.5.1).

Map Obstacle: The *Information Sources* are feeding *Avoidance Grid* with partial information of *Map obstacle rating*. *Map Obstacle Rating* shows the certainty that *charted obstacle* is in given cell. This property is bound to *Information Source* and it has *range* in $[0, 1]$.

The *Map Obstacle Rating* for cell (eq. 6.158) is calculated as product of maximal *Map Obstacle Rating* and *inverse visibility*. This gives *visibility biased* certainty of *Map Obstacle*.

$$map(cell_{i,j,k}) = \max \left\{ \begin{matrix} map(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \times \dots \\ \dots \times (1 - visibility(cell_{i,j,k})) \quad (6.158)$$

The example of *Map Obstacle Rating* calculation is given in (sec. 6.5.2).

Intruder: There is a set of *Active Intruders*, each intruder is using its own *parametric intersection model*. This parametric *intersection* model calculates *partial intersection ratings* representing *intersection certainty* ranging in $[0, 1]$. The more *partial intersection rating* is closer to 1 the higher is the probability of aerial collision with that intruder in that cell.

The *geometrical bias* is used for cumulative of multiple intruders, the *intruders are not cooperative*, therefore their occurrence can not be addressed by simple *maximum*. The proposed formula (eq. 6.159) is simply bypassing the intruder rating if there is one intruder. If there is more intruders the geometrical bias is applied.

$$intruder(cell_{i,j,k}) = 1 - \prod_{\forall intruder_i \in Intruders} \left(1 - intersection \begin{pmatrix} cell_{i,j,k}, \\ intruder_i \end{pmatrix}\right) \quad (6.159)$$

The *intruder intersection models* are outlined in (sec. 6.6.1).

Constraint: The *constraints* are coming from various *Information Sources*, the *hierarchical constraint application* is resolved by higher level logic. All *constraints* in this context are considered as *hard*.

The *Constraints rating* (eq. 6.160) is in *range* $[0, 1]$ reflecting certainty of constraint application in cell (usually 1).

$$constraint(cell_{i,j,k}) = \max \left\{ \begin{matrix} constraint(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \quad (6.160)$$

The *Constraint Rating* calculation example for *static* constraints is given in (sec. 6.5.3), the example for *moving* constraints is given in (sec. 6.6.5).

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

Threat: The concept of threat is *rating of expected harm* to receive in given segment of space. The threat can be time-bound to *decision time* t_i (time sensitive *intruder intersection models*).

The *harm prioritization* is addressed by higher navigation logic (fig. 6.26). All *sources of harm* are considered as equal. Threat is formalized in *following definition*:

Definition 19. *Threat is considered as any source of harm. The threat is* maximal aggregation *of various harm ratings. Our* threat *for specific cell is defined by (eq. 6.161).*

$$threat(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}), map(cell_{i,j,k}), \\ intruder(cell_{i,j,k}), constraint(cell_{i,j,k}) \end{matrix} \right\} \quad (6.161)$$

Reachibility: The *Reachibility* for trajectory reflects how safe is *path along*. The *Threat* (def. 19) for each cell has been already assessed. The set of *Passing Cells* is defined in *Trajectory Footprint* (eq. 6.70).
The *Trajectory Reachibility* is given as product of *Threats* along the trajectory (eq. 6.162). The *Trajectory Reachibility* can be calculated for each *trajectory segment* given as $\{movement_1, \ldots, movement_i\} \subset Buffer$ originating from $state_0$.

$$reachibility(Trajectory) = \prod_{PassingCells}^{\forall cell_{i,j,k} \in} (1 - threat(c_{i,j,k})) \quad (6.162)$$

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

The *Reachibility* of *cell* is given by best trajectory segment passing 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 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. 6.163).

$$reachibility(cell_{i,j,k}) = \max\{Trajectory.Segment(cell_{i,j,k}).Reachibility : \forall Trajectory \in PassingTrajectories(celli, j, k)\} \quad (6.163)$$

Note. Function Trajectory.Segment($cell_{i,j,k}$).Reachibility gives same results for any segment in $cell_{i,j,k}$, because (eq. 6.162) 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* [35] demands the fail rate below 10^{-7}, because there is no definition for *UAS* the *minimal fail rate* is expected to be at similar level.

The *fuzzy values* $[0, 1]$ are projected to *Boolean* properties of *cell* and *Trajectory* in following manner (tab. 6.5).

The high values of *Visibility* (eq. 6.156) and *Reachability* (eq. 6.163, 6.162) are expected. The low *threshold* for *threaths* values is expected. The error margin is solved by *Sensor Fusion* therefore initial *false positive* cases have low rate. The *Detected Obstacle Rate* (eq. 6.157), *Map Obstacle Rate* (eq. 6.158), *Intruder Rate* (eq. 6.159), and *Constraint Rate* (eq. 6.160) 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	$reachibility(trajectory)$	$\geq$	$(1 - threshold)$
Reachable Cell	$reachibility(cell_{i,j,k})$	$\geq$	$(1 - threshold)$

Table 6.5: Changing ratings from fuzzy to Boolean parameters.

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

The *Uncertain* space for decision time t_i is a portion of *Avoidance Grid* which *UAS* can not *read* with *Sensor Field*. The *cells* with $\neg Visible$ property. The *Uncertain* space is given by (eq. 6.164).

$$Uncertain(t_i) = \{cell_{i,j,k} : cell_{i,j,k} \in AvoidanceGrid(t_i), cell_{i,j,k}.\neg Visible\} \tag{6.164}$$

The *Occupied* space for decision time t_i is a set of cell which are classified as *Detected Obstacles*. The *Visibility* is not an issue, due the initial damping in (eq. 6.157). The formal definition is the space portion where it is possible to detect *obstacle bodies* or their portions (eq. 6.165).

$$Occupied(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.DetectedObstacle \end{array} \right\} \tag{6.165}$$

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. 6.166).

$$Constrained(t_i) = \left\{ \begin{array}{l} \\ cell_{i,j,k} : cell_{i,j,k} \in AvoidanceGrid(t_i), \\ \\ \end{array} \right. \left. \begin{array}{l} \\ cell_{i,j,k}.Visible, \\ cell_{i,j,k}.Constraint \vee cell_{i,j,k}.Intruder \end{array} \right\} \tag{6.166}$$

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. 6.167).

$$\begin{aligned} Free(t_i) = AvoidanceGrid(t_i) - \ldots \\ \cdots - (Uncertain(t_i) \cup Occupied(t_i) \cup Constrained(t_i)) \end{aligned} \tag{6.167}$$

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. 6.168).

$$Reachable(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Reachable \end{array} \right\} \tag{6.168}$$

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

$$|Reachable(t_i)| > 0, \quad Reachable(t_i) \subset Free(t) \tag{6.169}$$

6.7.2 Avoidance Grid Run

Main Goal: The main goal of this section is to introduce the trajectory selection process, based on a *situation assessment*, originating from *Data Fusion Procedure* (sec. 6.7.1).

Note. The *rating calculation* is outlined in (sec. 6.7.1). Low cost sensor fusion example usable to feed our data fusion procedure is given in [36]. Semi-optimal concatenation trajectory search like ours can be found in [37].

Note. The *Sensor Fusion Procedure* is solving all following steps (sec. 6.7.1). The *main purpose* of *Avoidance Run* is finding best path under certain conditions.

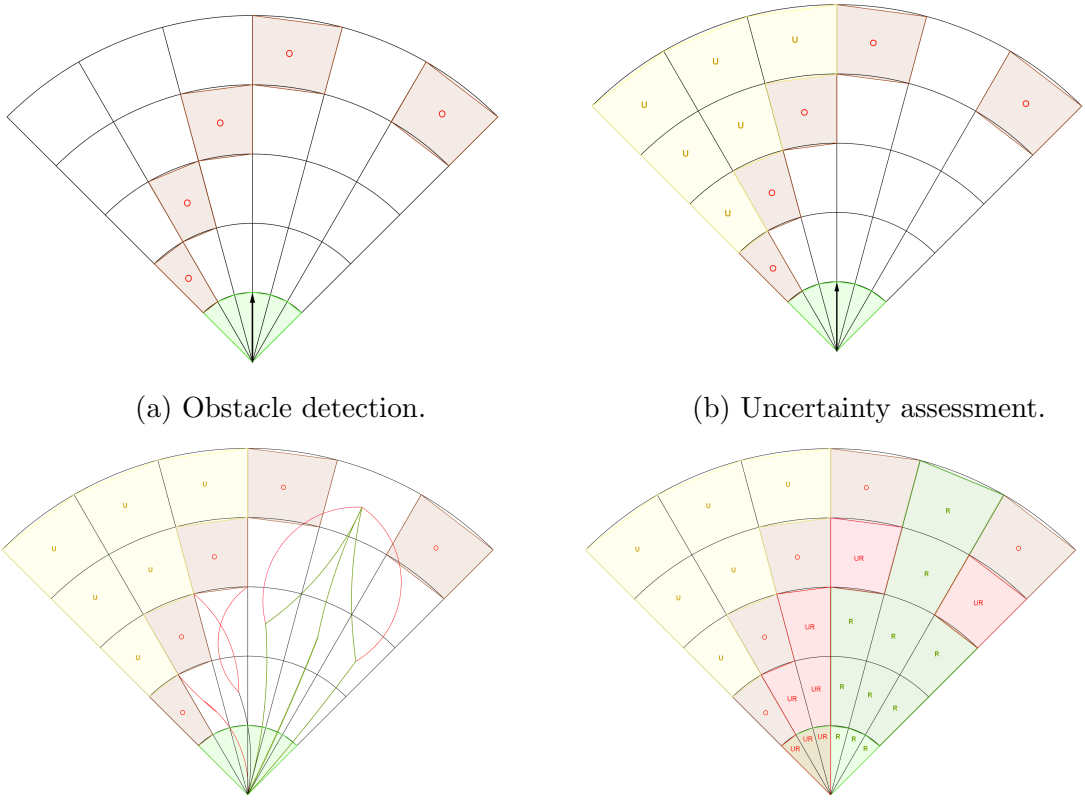
Space Assessment Principle: The *Avoidance Grid* is fed through *Data Fusion* (sec. 6.7.1). The process of *ratings assessment* (tab. 6.5) is given in (fig. 6.20):

1. *Obstacle detection* (fig. 6.20a) - assessment of *detected obstacles* (eq. 6.157). The red (O) *cells* have Detected obstacle set as *true*. The other threats: *map obstacles* (eq. 6.158), *intruders* (eq. 6.159), *constraints* (eq. 6.160) are false. The red (0) *cells* are representing $Occupied(t_i)$ (eq. 6.165) space in *Avoidance Grid* at decision time t_i.

2. *Uncertainty assessment* (fig. 6.20b) - the uncertain cells are cells which status can not be *assessed*. The *Visibility* (eq. 6.156) is low. The *Uncertain* cells (yellow (U) mark) are equal to $Uncertain(t_i)$ (eq. 6.164) in *Avoidance Grid* in *decision time* t_i. The $Constrained(t_i)$ (eq. 6.165) space is equal to $\varnothing$ in this example.

3. *Trajectory reachibility evaluation* (fig. 6.20c) - the *Reach Set* given as *Trajectory Set* (eq. 6.66). is then projected through *Avoidance Grid* and pruned according to (def. 14). *Reachable Trajectories* (eq. 6.162) are only those contained in $Free(t_i)$ space (eq. 6.167). The *Reachable Trajectories* are denoted as *green lines*. The *Unreachable* trajectory segments are denoted as *red lines*.

4. *Cell reachibility evaluation* (fig. 6.20d) - the evaluation of *cells* reachibility is going according to (eq. 6.163). The *Reachable cells* are those which *contains* at least one *Reachable Trajectory Segment*.



(a) Obstacle detection.

(b) Uncertainty assessment.

(c) Trajectories reachibility evaluation.

(d) Cell reachibility evaluation.

Figure 6.20: 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. 6.5). The following properties are know prior the *trajectory* selection:

1. *Reachibility* for each $cell_{i,j,k}$ (eq. 6.163).
2. *Reachibility* for each $Trajectory(\circ)$ (eq. 6.162).
3. *Free Space* as non empty set of *cells* in *Avoidance Grid* (eq. 6.167), with *Reachable Space* (eq. 6.168).
4. *Goal Waypoint* $\mathcal{WP}_G$ from *Mission Control Run* (sec. 6.7.3).

The *Algorithm* (alg. 6.6) is based on *shortest path* search. Navigation is trying to reach *goal waypoint*, therefore it tries to shorter distance between *trajectory final cell* and *goal waypoint*. If there is *reachable space* two situations can occur:

1. *Goal waypoint is inside the Avoidance Grid* - the *avoidance cell* is $\text{cell}_{i,j,k}$ containing *goal waypoint* if reachable.
2. *Goal waypoint is outside the Avoidance Grid* - the *avoidance cell* is closest cell considered as *outer cell* to *goal waypoint*.

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

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

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

Input : Cell[] reachable (eq. 6.168), Waypoint goal, AvoidanceGrid(t_i) grid
Output: Trajectory avoidancePath, Error message

```
# Initialization & Reachibility 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]
```

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 following example of *space assessment* and *Best Path Selection*.

The *UAS* (blue plane) is following *mission plan* in open space. Then there is a detection of an *collision situation* (fig. 6.21). The *Obstacle* is detected in *top-right* Avoidance Grid corner.

The *LiDAR hits* are denoted as red filled circles. The *Avoidance Grid* space is constrained by black dashed line. The *Avoidance Grid* is separated into 5 layers going from top to *bottom*. The *Reach Set* is projected as a set of *Trajectories* with colorization.

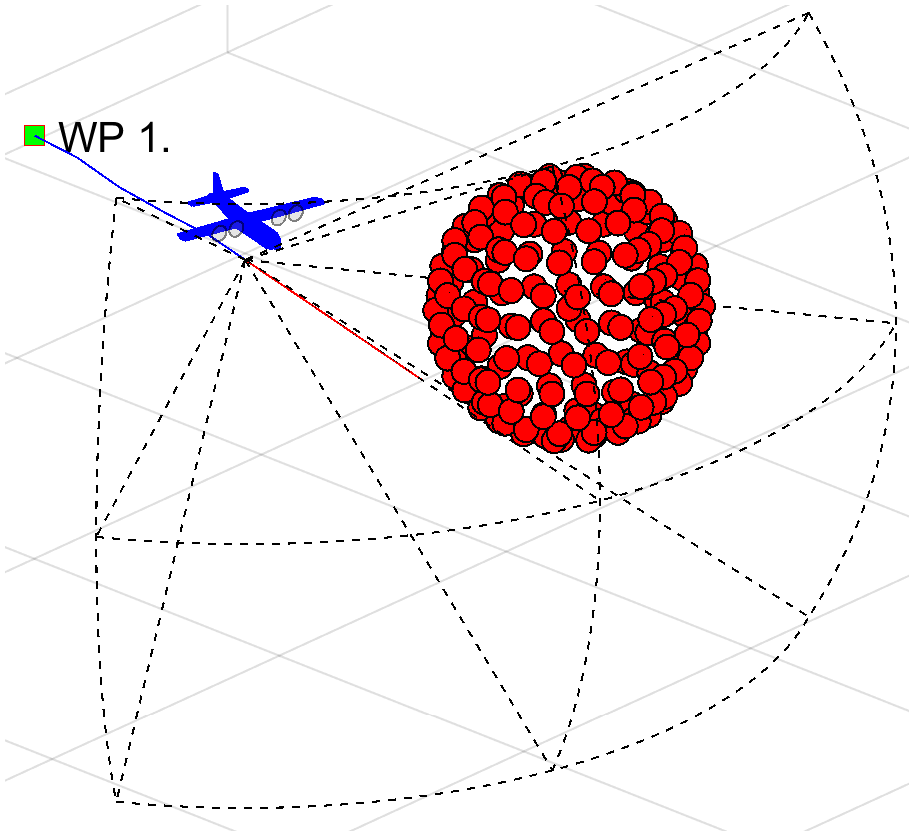


Figure 6.21: Example: The situation to be evaluated by *Avoidance Run*.

Visibility Assessment: The visibility assessment (fig. 6.22) 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.

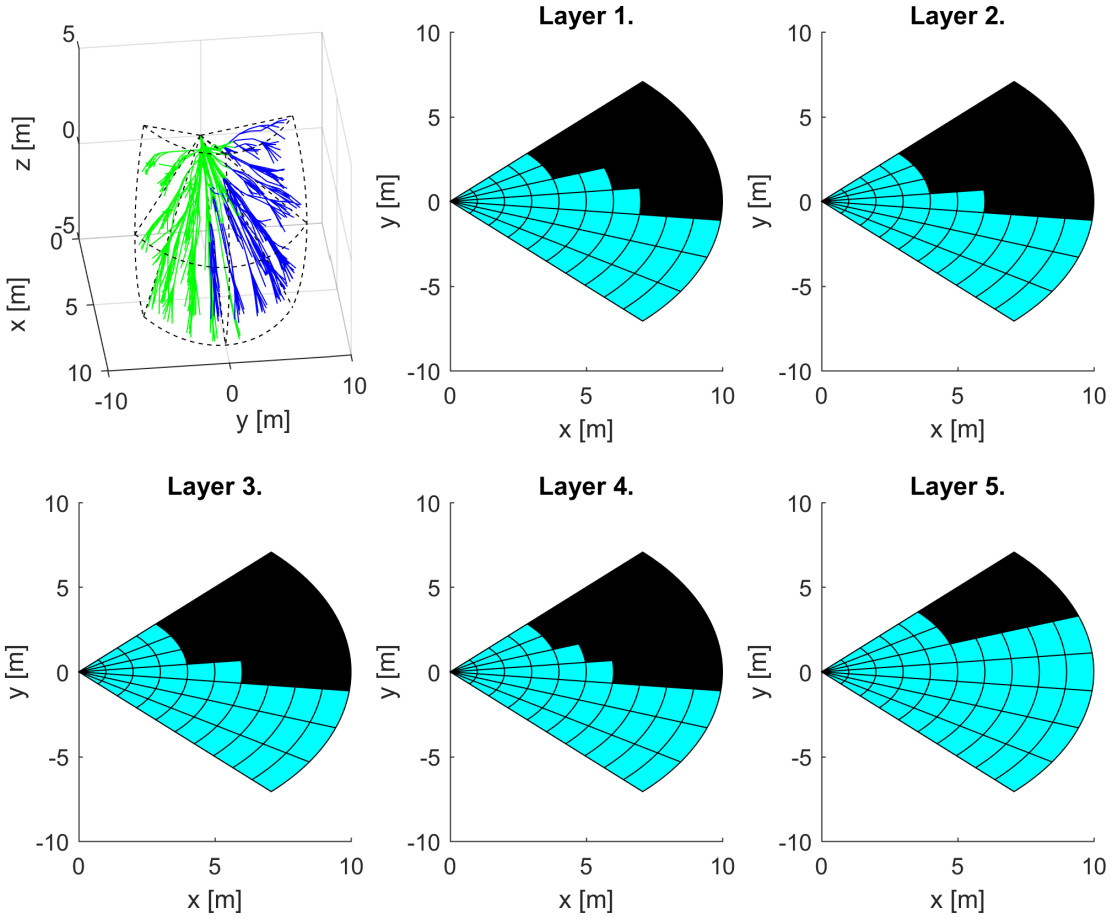


Figure 6.22: Example: The *Visibility* evaluation by *Avoidance Run*.

Reachibility Assessment: For Each trajectory the *Reachibility* is assessed (fig. 6.23). The *Obstacle Space* and *Uncertain Space* are rendering *reachibility*, effectively separating *trajectories* into two categories:

1. *Unreachable Trajectories* (red lines) - there is at least one trajectory segment leading 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 similar matter, depending on 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*.

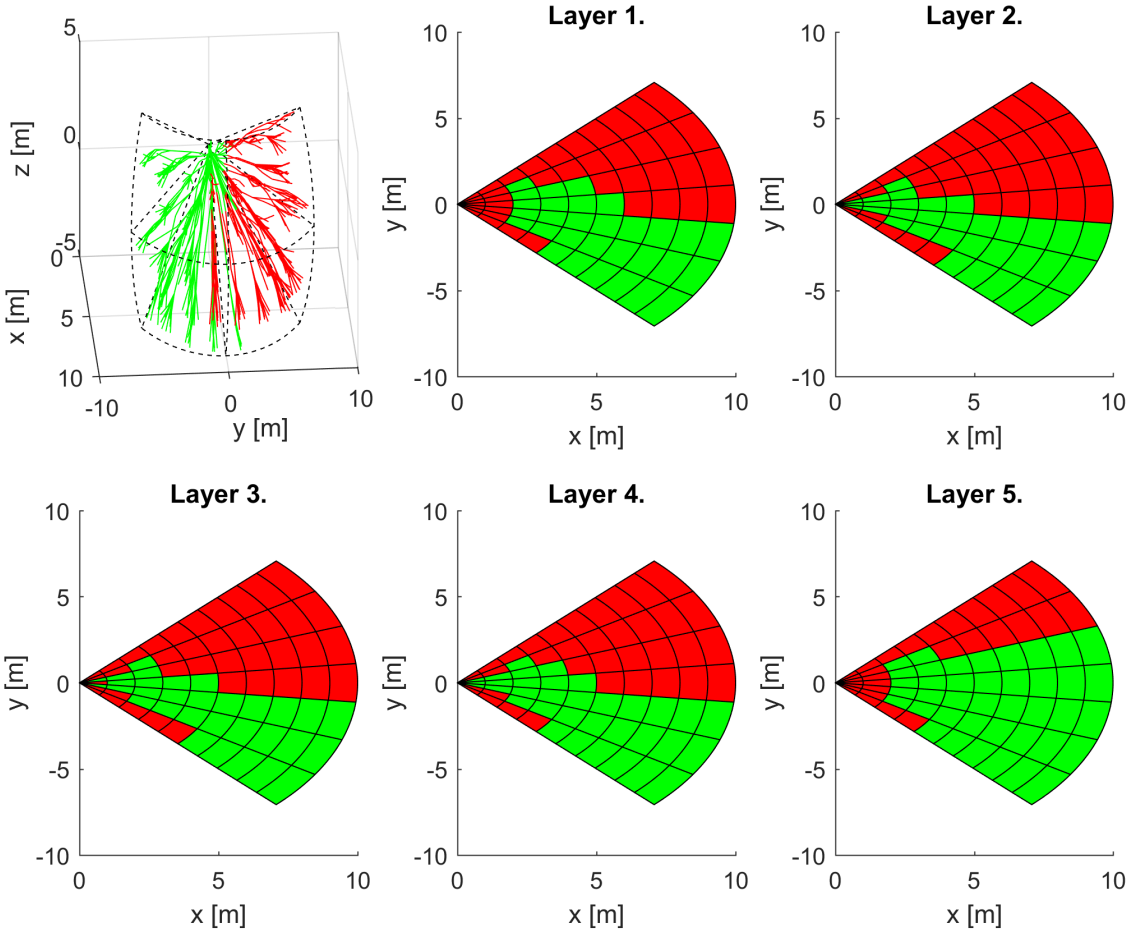


Figure 6.23: Example: The *Reachibility* evaluation by *Avoidance Run*.

Note. The *best avoidance path* is selected form *reachable outer cells* (green fill in fig. 6.23), depending on *goal waypoint* according to (alg. 6.6).

6.7.3 Mission Control Run

Introduction and Motivation: This section will introduce *Navigation Concept* using *Reach Set Approximation*. The *Avoidance Framework Concept* (fig. 6.2) defines *Navigation Module* as *sub-system* for long term *trajectory tracking*. The *Avoidance Grid Run* (sec. 6.7.2) is solving the *Path Search* problem inside operation space constrained by *Avoidance Grid* for time t_i.

There is a need to build a trajectory between *Waypoints* which are further away than *distance* of one *Avoidance Grid*. The *UAS* is controlled via *Movement Automaton*. The *Movements* which are in *Movement Buffer* can be replaced with another movements. This feature of *Movement Automaton* is called *Movement Chaining* (eq. 6.8).

To join the multiple *Avoidance Grids* paths following terminology needs to be established (fig. 6.24a):

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

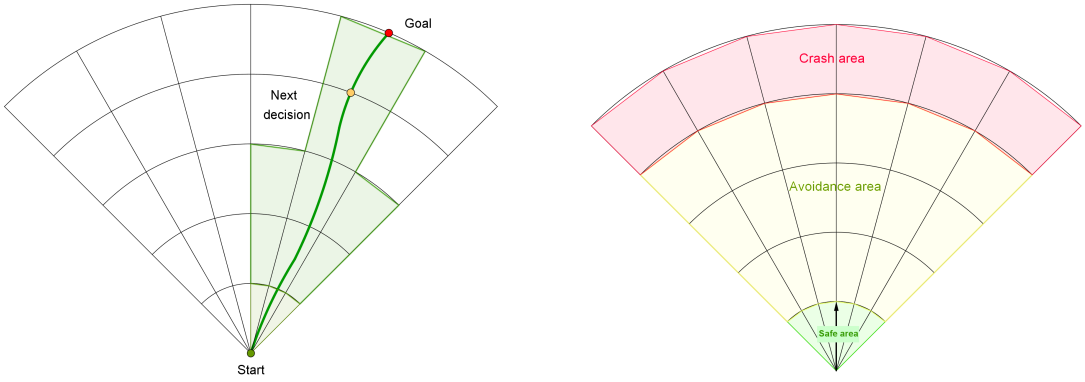
2. *Next Decision* - the point when the next *Avoidance Grid Run* is applied. The outline of events and triggers is required. The *decision* will be made in *next decision time* t_{i+1}.

The *Avoidance Grid* from *UAS* viewpoint can be separated into following zones (fig. 6.24b):

1. *Crash Area* (last layers) - there is no place for safe return and the *border* of *Avoidance Grid* is near. The *Decision Point* needs to lie before this zone.

2. *Avoidance Area* (middle layers) - the area of *Active Avoidance Maneuvering*. The *Reach Set Approximation* performance (sec. 6.4.1) is important in this area.

3. *Safe Zone* (first layers) - there is space for safe return or damage mitigation.



(a) Mission control run example. (b) Grid Zones.

Figure 6.24: Definitions for *Mission Control Run* (outer loop).

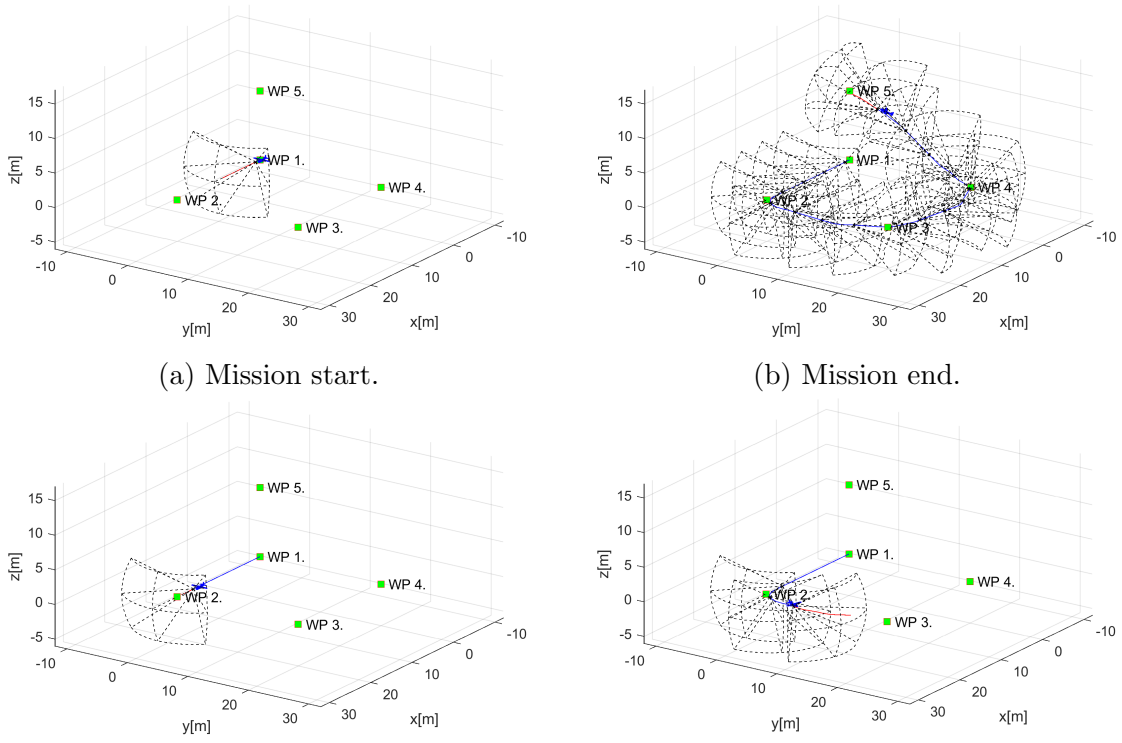
Joining *Avoidance Grid Runs* (fig. 6.25) example portrays *Avoidance Grid Runs* invoked on various *Decision Points* to achieve *Navigation* functionality. The UAS (blue plane) is flying Mission (green numbered waypoints). The *Avoidance Grid* boundary (black dashed line) for each *Decision Point* (UAS position at time t_i). Following example of *Navigation* (fig. 6.26) run is shown:

1. *Mission Start* (fig. 6.25a) - 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. 6.25b) - UAS have reached *last waypoint*. All *Avoidance Grid* boundaries (black dashed line) for all *runs* are drawn along flown trajectory.

3. *Waypoint Reach* (fig. 6.25c) - 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. 6.25d) - 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 6.25: Joining multiple *Avoidance Grid Runs* for achieve Navigation.

General Concept: [3] The *General Concept* is taken from [38, 39], consisting from following main modules:

1. *Navigation Loop* - module responsible for *Navigation* providing *Goal Waypoint*.
2. *Data Fusion* (background in sec. 6.7.1) - module responsible for *Surveillance Data Feed*.
3. *Situation Assessment* - module responsible for *UAS Safety Evaluation*.
4. *Avoidance Run* (background in sec. 6.7.2) responsible for *Avoidance Path* selection.

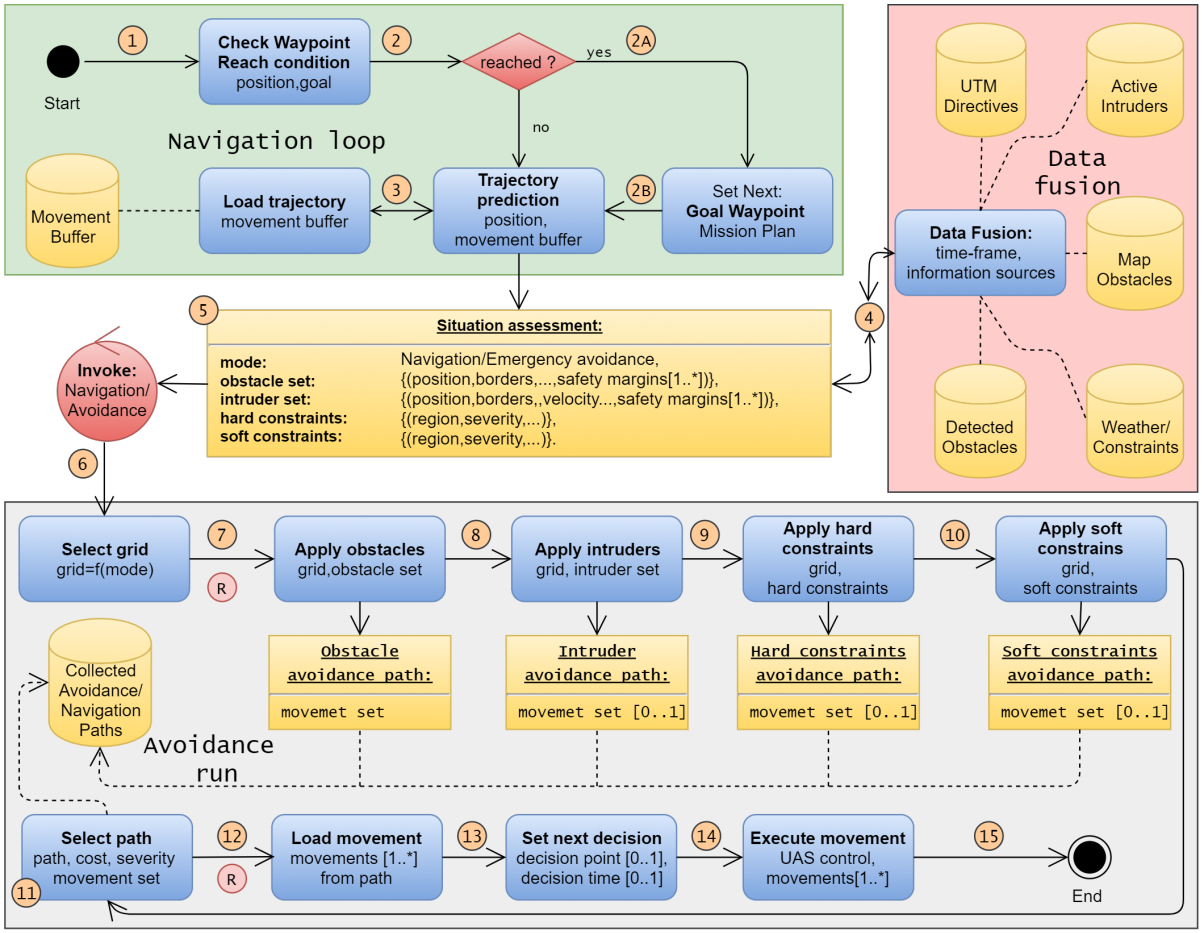


Figure 6.26: Mission control run activity diagram.

The main changes to *Navigation architecture* are given in *Mission Control Run* activity diagram (fig. 6.26):

1. *Situation Assessment* - added event-based mode switching control.
2. *Avoidance Run* - added hierarchical evaluation for *Avoidance Path* selection. Prioritizing threat avoidance according to a type.

[3] Mission Control Run Function Implementation: `RuleEngine/MissionControl/MissionControl.m::runOnce(.)`

The *Operation Mode* is introduced, based on *Situation assessment* and *Triggering Events* one of following modes are selected in *Avoidance Run*:

1. *Navigation Mode* - the *UAS* is navigating through *Airspace* following *cost effective patterns* and obeying *Airspace Authority* (UTM). The *Navigation Grid* is a instance of *Avoidance Grid* (sec. 6.3) with initialized *Navigation Reach Set* (ex. *Harmonic Reach Set Approximation* (sec. 6.4.4)).

2. *Emergency Avoidance Mode* - the *UAS* is *threatened* by obstacle, intruder, hard constraint or *soft constraint*, the *UAS* is navigating through *Airspace* following *safe avoidance patterns* and *minimizing the impact* of possible damages. The *Avoidance Grid* is term used for *Emergency Avoidance Mode*. The *Avoidance Reach Set Approximation* is initialized in *Avoidance Grid* (ex. *Chaotic Reach Set Approximation* (sec. 6.4.3))

Note. Depending on *Operation Mode* the pair of *Avoidance Grid* and *Reach Set* is selected in *Avoidance Run* part.

The *Navigation Grid* and *Avoidance Grid* shares the space segmentation pattern, therefore the *Data Fusion* (sec. 6.7.1) needs to be evaluated only once for both grids.

Decision Time Frame ($[t_i, t_{i+1}[$): The *Mission Control Run* is executed for *Decision Time Frame* bounded to the *period* of the *UAS executed movement* (fig. 6.2).

The *UAS System* (sec. 6.2.4) controlled by *Movement Automaton Implementation* (sec. 6.2.5) *Planned Movements* can be changed at any time. The real impact on control is shown after the *actual movement* is executed.

Note. For our *Movement Automaton Implementation* movements the average *movement duration* is *1/velocity second* (tab. 6.1, 6.2).

The *Decisions* are made based on *system* state in *current* time-frame started at t_i for *next* time frame starting at t_{i+1}.

Note. Because the *Decision Delay* is crucial in *Avoidance System* it is beneficial to have *short time movements*. On the other hands, the *length and duration of movements* is impacting *Reach Set Complexity*. The proper construction of movement automaton is greatly impacting overall *approach performance*.

Initialization: The *UAS* is going to solve a problem for *Rules of the Air* (eq. **??**). Using control scheme (fig. 6.2) with given *Sensors*:

$$Sensors = \{LiDAR, ADS - B\} \tag{6.170}$$

The sensors obstacle assessment into avoidance grid is outlined for static obstacles in (sec. 6.5) and for moving obstacles in (sec. 6.6.)

The *Data Fusion Procedure* is given as follow:

$$DataFusion = \{RatingBasedDataFusion \quad (sec.6.7.1)\} \tag{6.171}$$

Then the *UAS system* (sec. 6.2.4) with *Movement Automaton Implementation* (sec. 6.2.5) with empty movement buffer:

$$MovementBuffer = \{\} \tag{6.172}$$

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:

$$\begin{aligned} AvoidanceGrid(0) &= \{UAS.position(0), AvoidanceReachSet(UAS.ReachSet)\} \\ NavigationGrid(0) &= \{UAS.position(0), NavigationReachSet(UAS.ReachSet)\} \\ OperationMode &= Navigation \end{aligned} \tag{6.173}$$

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

$$\begin{aligned} Mission &= \{Waypoint_1 \dots Waypoint_n\} \\ GoalWaypoint &= Mission.waypoint_1 \\ LastWaypoint &= Mission.waypoint_n \end{aligned} \tag{6.174}$$

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

$$obstacles = \{\}, intruders = \{\}, hardConstraints = \{\}, softConstraints = \{\} \tag{6.175}$$

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 definition of *waypoint reach condition* (def. 20) and *Unreachable waypoint* (def. 21).

Definition 20. *Waypoint Reach Condition for* current *decision time t_i for* UAS *position and current* Goal Waypoint *is satisfied only if:*

$$distance(UAS.position(t_i), GoalWaypoint(t_i)) \leq 2 \times \max\{length(movement) : \forall movement \in MovementSet\} \quad (6.176)$$

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

Definition 21. *Unreachable Waypoint. The* Goal Waypoint *is evaluated as unreachable in decision time t_i when* Avoidance Grid Run *(alg. 6.6) can not find the* navigation/avoidance path *leading to it.*
Formally: The Avoidance/Navigation Grid *has range defined as* final layer distance*. When the* Goal Waypoint *is in* range *of* Grid*:*

$$Grid(t_i).range \geq distance(UAS.position(t_i), GoalWaypoint(t_i)) \quad (6.177)$$

and following condition is satisfied:

$$\forall cell_{i,j,k} \in Grid(t_i) \; \nexists cell_{i,j,k}.Reachable == true \wedge \ldots \cdots \wedge distance(cell_{i,j,k}, GoalWaypoint(t_i)) \leq \ldots \cdots \leq 2 \times \max\{length(movement) : \forall movement \in MovementSet\} \quad (6.178)$$

The Goal Waypoint *is unreachable.*

Then the *Navigation Loop* is invoked every *decision time t_i*, *Mission Control Run* (fig. 6.26), it is described as sequence of following steps:

1st Check Waypoint Reach Condition - the *UAS position* for given *time frame t_i* is checked under condition (eq. 6.176). If condition is met continue with 2nd step otherwise continue with 3rd step.

2nd Set Next Waypoint - until following condition is met:

$$GoalWaypoint == LastWaypoint$$

Set next goal waypoint like follow:

$$GoalWaypoint = Mission.getNextWaypoint()$$

Otherwise enforce *Landing sequence* (Out of Scope).

3rd Trajectory Prediction - the *Movement Buffer* is loaded with planned movements from *Movement Automaton*. The *future trajectory* is predicted according to (eq. 6.37):

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

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

Data Fusion (4th step) The *Data Fusion* (sec. 6.7.1) in this context is *Threat Sets* preparation for *Avoidance Run*. It is depending on values of *Boolean values* defined in (tab. 6.5) for *threat* classification.

Note. Avoidance Grid's Data fusion (sec. 6.7.1) is run in 7th- 10th step (fig. 6.26).

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

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

The *intruders* source are valid *active intruders notifications* received from ADS-B In positioned to *future expected positions* at *decision time* t_{i+1}:

$$intruders = ADS - B.getActiveIntruders(t_{i+1})$$

Note. The *Intruders* needs to be predicted for the next decision time-frame starting at time t_{i+1} Due their mobility.

The *hard/soft constraints* are obtained from *Information Sources* and the area of next decision time t_{i+1} *Avoidance Frame* is used as space parameter in search. The sets of hard and soft constraints are obtained in following manner:

$$hardConstraints = InformationSources.fuse(AvoidanceGrid(t_{i+1}))$$

$$softConstraints = InformationSources.fuse(AvoidanceGrid(t_{i+1}))$$

The results of *Data Fusion* threats set preparation are used in next step.

Invoke Navigation/Avoidance based on Situation Assessment (5th-6th step): The *deciding events* depending on *Trajectory Prediction* (3^{rd} step) and *Data Fusion* (4^{th} step) (fig. 6.26) are following:

1. *General Events* are *triggered* regardless *Operation Mode*. They are considered after *specific mode events* are handled and *Navigation/Avoidance Grid* is selected:

a. *Empty Movement Buffer* ($MovementBuffer = \varnothing$) - if there is no movement in *Movement buffer* to be executed (from 3rd step: Load Trajectory), the *Avoidance Run* is enforced to run with *Navigation/Avoidance Reach Set Approximation* to generate new path.

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

c. *Waypoint Unreachable* - this type of event is very situations based. The *Waypoint Reachibility* (assumption. **??**) has not been relaxed, therefore this event is not properly handled in approach. The *implementation* considers *selecting next waypoint in mission* as a goal waypoint of *first waypoint* if *unreached/unreachable waypoints* are exhausted.

2. *Navigation Mode Events* are triggered if *Operation Mode* is set as *Navigation*:

a. *Empty Navigation Grid* ($|threats| = 0$) - if *movement buffer* contains at least one *movement*, the *Avoidance Run* is omitted. The *Operation Mode* stays in *Navigation Mode*.

b. *Collision Case Resolution* ($|ActiveCollisionCases| > 0$) - there is new/active *Collision Case* (sec. 6.8.8), the *Navigation Reach Set Approximation* trajectories will be constrained according to active *Collision Case(s)* requirements. If there exists at least one *Reachable* avoidance path, the *Operation Mode* will remain *Navigation*. If there is no *Reachable* avoidance path, the *Operation Mode* switches to *Emergency Avoidance*.

c. *Static Obstacle Detection* ($LiDAR.Hits > threshold$) - if *static obstacle set* contains at least one *detected obstacle* (sec. 6.5.1) intersecting with *Navigation grid* the *Operation Mode* will be *switched* to *Emergency Avoidance Mode*.

d. *Intruder Detection* ($intruders > 0$) - if *active intruders set* contains at least one *intruder* which expected impact area (intersection models (sec. 6.6.1)) *Navigation grid* the *Operation Mode* will be *switched* to *Emergency Avoidance Mode*.

e. *Hard or Soft Constraint Occurrence* ($|hardConstraints| > 0 \vee |softConstraints| > 0$) - if *hard/soft constraint set* contains at least one *constraints* which intersects (static constraints (sec. 6.5.3), moving constraints (sec. 6.6.5)) *Navigation grid* the *Operation Mode* will be *switched* to *Emergency Avoidance Mode*.

3. *Emergency Avoidance Events* are triggered if *Operation Mode* is set as *Emergency Avoidance*:

a. *Empty Avoidance Grid* ($|threats| = 0$) - if there is no *detectable* threat, the remainder of *avoidance path* is removed from *Movement Buffer*. The *Operation Mode* is switched to *Navigation* and new *navigation path* is selected.

5th Situation Assessment - if there is any flag raised by *Event Triggers*, there is an *avoidance situation.*

The *Event Triggers* describe complex *Operation Mode* switching. The simplified principle is following: *If UAS is in Emergency Avoidance Mode Always Invoke Avoidance Run. If UAS is in Navigation Mode Invoke Only if Necessary.*

If there was event trigger continue with 7th step, otherwise wait for *next decision time* t_{i+1}, execute movement and continue with 1st step.

6th Invoke Navigation/Avoidance depending on the *Operation Mode* the *Reach Set/Grid* pair is selected. The future $state(t_{i+1})$ in next decision frame t_{i+1} is necessary for Grid/Reach Set initialization. The *next decision frame initial state* is obtained by *prediction*:

$$state(t_{i+1}) = Trajectory(state(t_i), currentMovement)$$

The *Reach Set Approximation* is loaded based on *mode* and $state(t_{i+1})$. The *Grid* is initialized as $Free(t_{i+1})$ (eq. 6.167) for all cells.

Avoidance Run (7th-15th step): The *Avoidance Run* goal is to obtain *Path* represented as *Trajectory*(state(t_{+1},MovementBuffer)) (eq. 6.37) from *Navigation/Avoidance Grid* and associated *Navigation/Avoidance Reach Set Approximation.*

If the *Operation Mode* is set as *Navigation Mode* the algorithm continues with 11th step. Otherwise the *Avoidance Grid Space Assessment* is run multiple times to obtain $Reachable(t_{i+1})$ (eq. 6.168). The *Threat Data* obtained from 4th step are used.

7th Apply Obstacles - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated with following threat modification:

$$intruders = \varnothing, softConstraints = \varnothing, hardConstraints = \varnothing$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Obstacle Avoidance Path.*

8th Apply Intruders - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated with following threat modification:

$$softConstraints = \varnothing, hardConstraints = \varnothing$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Intruders Avoidance Path.*

9th Apply Hard Constraints - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated with following threat modification:

$$hardConstraints = \varnothing$$

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Hard Constraint Avoidance Path*.

10th Apply Soft Constraints - The *Space assessment* (tab. 6.5) for *Avoidance Grid* is calculated without any modification.

The *Find Best Path* (alg. 6.6) is applied, the resulting *avoidance path* is labeled as *Soft Constraints Avoidance Path*.

Note. The 7th to 10th steps are code-optimized for efficient calculation.

11th Select Path - based on *Operation Mode* the *Navigation/Avoidance Path* is selected.

The *Navigation Path* for *Navigation Mode* is selected by standard *Find Best Path* (alg. 6.6) procedure. The *Navigation Reach Set Approximation* can be constrained by *Rule Engine* (fig. 6.35).

The *Avoidance Path* for *Emergency Avoidance Mode* is selected from *Collected Avoidance Paths* with following priority:

1. *Soft Constraints Avoidance Path* - if exists continue with 12th step, if does not exist try to select:
2. *Hard Constraints Avoidance Path* - if exists continue with 12th step, if does not exist try to select:
3. *Intruders Avoidance Path* - if exists continue with 12th step, if does not exist try to select:
4. *Obstacle Avoidance Path* - continue with 12th step.

Note. The *Waypoint Reachibility* (assumption **??**) is weakened to the point that it is necessary for waypoint to be *Reachable* only in static obstacle environment. The *Constrained* and *Occupied* spaces are shrunk in following matter to increase UAS survival chances. There are following relaxations with their conditions:

1. *Soft Constraint Relaxation* - they are breakable by default. This kind of situation is allowed to happen under any circumstances.
2. *Hard Constraints Relaxation* - they can be broken in case of emergency (airspace constraints) or UAS robust build (Weather Constraints). This kind of situation is allowed under very specific conditions depending on *broken constraint* severity.

3. *Intruder Occupied Space Relaxation* - this can be broken if and only if there is guarantee the Intruder dynamic and navigation algorithm allows to avoid *Collision* with UAS. This relaxation should be used as *the last resort*.

12th Load Movements - the *Movement Buffer* is flushed for *future decision times* $t_{i+1}, \ldots, t_{i+k}$. The *Navigation/Avoidance Path* movements are pushed into *Movement Buffer* instead. The *executed movement* for *decision time* t_i remains (because its executed at this time point).

13th Set Next Decision - the *next decision point* is set depending on circumstances:

1. Navigation Mode (no active collision cases) - *Decision Point* is set as point before *UAS* enters into *Crash Zone* (fig. 6.24b) in *Navigation Grid*.
2. *Navigation Mode (at least one active collision case)* - *Decision Point* is set after *next movement execution*. Current decision point $UAS.Position(t_i)$, next decision point $UAS.Position(t_{i+1})$.
3. *Emergency Avoidance Mode (any circumstances)* - *Decision Point* is set after *next movement execution*. Current decision point $UAS.Position(t_i)$, next decision point $UAS.Position(t_{i+1})$.

14th Execute Movement - the *First Movement* from *Movement Buffer* is loaded to be executed in decision time frame $[t_{i+1}, t_{i+2}[$.

15th Finish Avoidance Run - if the *UAS* is flying, continue with 1st step.

Decision Frame: The *mission control run* (fig. 6.26) describes overall process in *sequence*. The *orchestration overview* is given in (fig. 6.27).

The key idea is to explain what happen in one *decision frame*. The *mission control run* is implemented as multi-thread application which sends the signals between threads. Each thread is 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 partial known world assessment (sec. **??**). An *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 *operational space context* (controlled/non-controlled airspace). The output of *data fusion* is full *known world assessment* (sec. **??**, 6.7.1). The *UTM-related* and *constraint related* events can arise from *data fusion*.

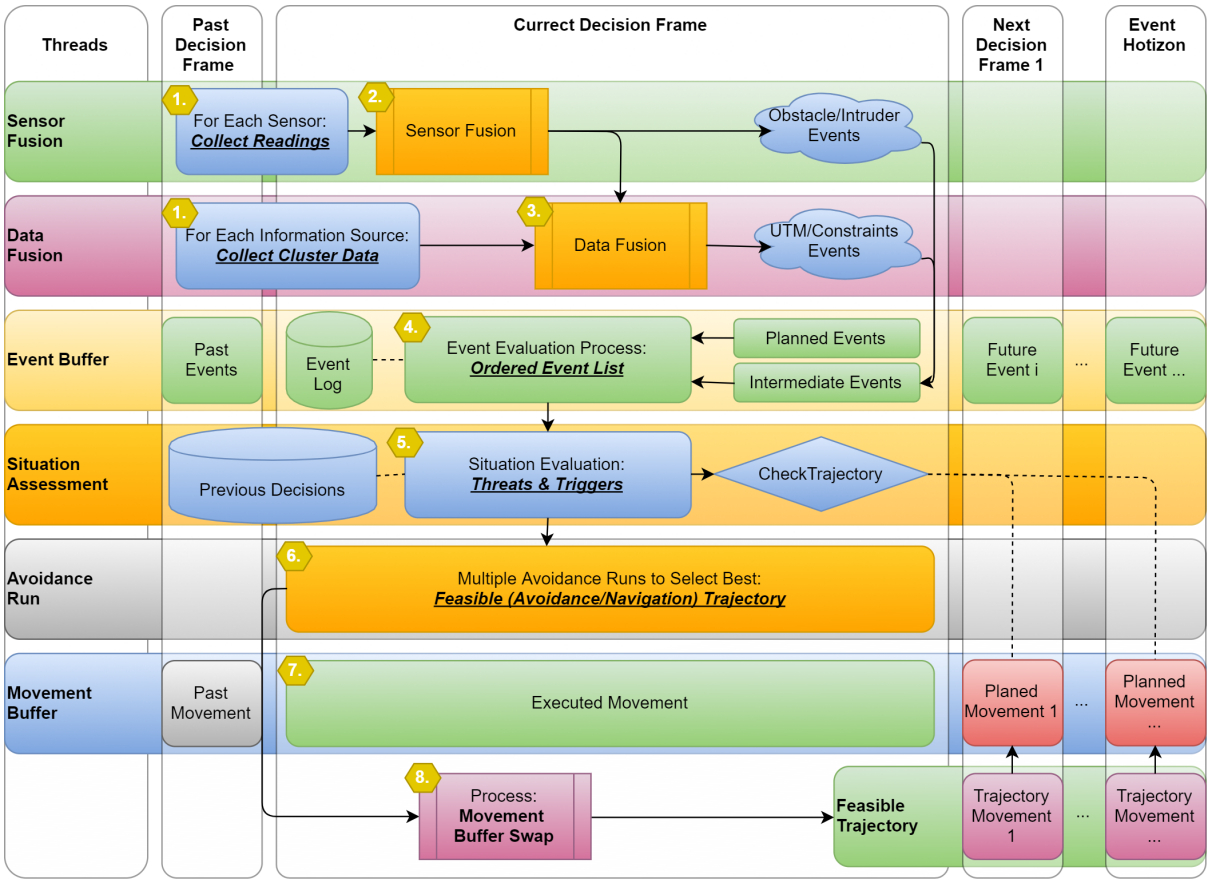


Figure 6.27: Mission control orchestration diagram.

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

   The *implemented events* are listed in 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 *event log*. The trace is useful for process and rule fine-tuning.

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

   c. *Handling* - the events are handled by invoking the *situation assessment* or by rule engine invocation (sec. 6.9.1).

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

4. *Situation Assessment* - invoked by *event buffer* o assess situation, responsible for proper *avoidance run* (sec. 6.7.2) 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 7^{th}-11^{th} step of *mission control run*.

6. *Movement Buffer* - represents *movement automaton implementation* (sec. 6.2.5). 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 current decision frame, this movement can not be changed.

   c. *Future movements* - future planned movements to be executed after *current decision frame* expires. These movements outlines planned trajectory (predictor mode sec. 6.2.7).

7. *Feasible Trajectory* - consists of *future planned movements* taking place directly after *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. 6.26). The numbered steps in (fig. 6.27) shows the threads orchestration in 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, 6.6).

3. *Data fusion* - the information sources are preprocessed according to (sec. 6.5, 6.6).

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 the most of the threats is selected.

7. *Movement execution* - the movement for *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*.

6.7.4 Computation Complexity

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

Navigation Loop: I the navigation loop, the *waypoint reach condition* (eq. 6.176) is checked, this is 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* is depending on clustering. The *Airspace Clustering* is considered as static for our setup. Therefore the *count of active airspace clusters* has main impact on complexity. The *count of information sources* is static and not changing over mission time. Information sources usually implement *Hash search function* with complexity $\mathcal{O} \ln |searchedItemSet|$.
The *computation complexity* boundaries for *Data fusion* in our setup are following:

$$\begin{aligned} \texttt{Select Active Collision Cases:} &\quad \mathcal{O}(|collisionCases|) \\ \texttt{Threshold Detected Obstacles:} &\quad \mathcal{O}(|cells|) \\ \texttt{Load Map Obstacles:} &\quad \mathcal{O}(\ln |activeClusters| \times |informationSources|) \\ \texttt{Load Hard Constraints:} &\quad \mathcal{O}(\ln |activeClusters| \times |informationSources|) \\ \texttt{Load Soft Constraints:} &\quad \mathcal{O}(\ln |activeClusters| \times |informationSources|) \end{aligned}$$

Note. The *real-time clustering* is *hard non-polynomial problem* [40]. Usually all information sources and sensor have *polynomial complexity* of processing. The *controlled airspace clusters* are usually set for very long period of time. Therefore *Obstacle Map*, *Airspace Constraints*, and, *Weather Constraints* can be considered as preprocessed

Situation Assessment: The *Situation Assessment* is evaluating triggering events. The *evaluation* is usually simple existence question without further calculations. The *complexity* of *event evaluation* for our case is $\mathcal{O}(1)$. There are 8 triggers. The count of *triggers* needs to be accounted in complexity boundary:

$$\mathcal{O}(|triggers| \times eventEvaluationComplexity)$$

Note. The *trigger calculation complexity* needs to stay low, because the *triggers* are verified every *Mission Control Run*. The *Avoidance Run* trigger frequency should be very low under normal conditions.

Avoidance Run: The *Avoidance run* is most critical part of *Mission Control Run*, because *Avoidance Path* calculation. The *Navigation Path* calculation is less complex (Rule engine is not accounted), therefore *Emergency Avoidance Mode* is assumed.

The *threat insertion* is realized in 7^{th} to 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* (sec. 6.6.2) is going through maximum of *layers count* cells.

The *body volume intersection model* (sec. 6.6.3) can check the *simple intersection condition* over all *Avoidance Grid* in worst case, therefore complexity for this check is bounded by *count of cells*.

The *Maneuverability Uncertainty Intersection* (sec. 6.6.4) can hit all cells in *Avoidance Grid*. The calculation complexity boundary is exponential depending on *horizontal/vertical* spread in $[rad]$. The *intersection* implementation was done *ad-hoc*. The impact of *intersection application* is visible only when there is more than *4* concurrence intruders (fig. **??**).
The *complexity boundary for* intruder insertion is given like follow:

$$\texttt{Insert Intruders:} \quad \mathcal{O}\left(\sum \begin{bmatrix} |linearIntersections| \times |layers| \\ |bodyvolumeIntersections| \times |cells| \\ |cells|^{horizontalSpread \times verticalSpread} \end{bmatrix}\right)$$

Note. The *intruder intersection* is critical in *non-controlled airspace*. The main complexity gain in *controlled airspace* is from *rule application*. Our *rule complexity* is in worst case depending on *Reach Set* node count and *Active Collision Cases* count.

$$\texttt{Apply Our Rules:} \quad \mathcal{O}(|activeCollisionCases| \times |nodes|)$$

For *Hard/Soft Constraints* The algorithm used for intersection polygons was selected based on study [32], the selected algorithm *Shamos-Hoey* [33]. 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 10^{th} step. All ratings (tab. 6.5) expect $Reachibility(cell_{ij,k})$ and $Reachibility(Trajectory)$ are calculated. The *calculation complexity* boundary for one *reachibility rating* is $\mathcal{O}(1)$. (eq. 6.162, 6.163). The *Recalculate Reachibility* operation applied $4\times$ have maximal *complexity* boundary given as follow:

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

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

6.7.5 Safety Margin Calculation

Safety Margin Determination: To determine *safety Margin* the *Rule of Thumb* is used:

$$maximalBodyRadius \leq safetyMargin \leq 2 \times turningRadius \tag{6.179}$$

The *lower boundary* is given by *UAS* construction. because the *UAS* body is considered as *unit ball* with radius given as *maximal body radius*.

The *upper boundary* is optional, The *double of* turning radius is used by the *conservative approach* [41].

Safety Margin Bloating: The *discretization* of *Reach Set*, *Operation Space* and *Decisions* imposes standard *mixed integer* problem in terms of *safety*. This section covers *non-exhaustive* list of possible *Safety Margin Bloats* in our approach.

Own Position Uncertainty Bloat: The *sensor fusion* is precise, but not *exact* in own UAS position determination. The maximal usual disparity needs to be accounted into *Safety Margin*.

Intruder Position Uncertainty Bloat: The *sensor fusion* of Intruder is precise, but not *exact* in own UAS position determination. The maximal usual disparity needs to be accounted into *Safety Margin*.

Weather bloat: The *Weather* impact type may result to increased *safety margin*. Example: UAS is not humidity resistant, the clouds will be avoided from greater distance.

Airspace bloat: The *Airspace* depending on cluster or *country* may require greater separation distances, depending on circumstances. The example can be UAS directive to keep minimal separation from obstacles. The *Safety Margin* is usually overridden by UTM directive value.

UTM Synchronization Bloat: *Both UAS* decision times were *synchronized*. The *intruder* can be offset for *full decision frame*. This is not an assumption, but it shows critical performance. Usually safety margin is bloated for (worst case offset):

$$safetyMarginBloat = \begin{pmatrix} intruderVelocity \times \dots \\ intruderDecisionFrame \end{pmatrix} [m, ms^{-1}, s] \tag{6.180}$$

6.8 UAS Traffic Management

The *Traffic Management* for UAS is based on existing Air Traffic Management System for manned aviation [35]. The controlled airspace segments are *static* and have one *authority for one zone* principle. The dynamic zones have been proposed in [42]. But it will be omitted for *simplification purpose*. The necessity for *UAS integration* into *National Airspace* have been outlined in [43].

The latest *Airbus blueprint* [44] outlines some functionality. The main purpose of this section is to show *Reach Set based Approach* capability to follow *Usual Air Traffic Management* commands.

The *section* is organized to introduce:

1. *UTM Architecture* (sec. 6.8.1) - centralized ATM like authority over airspace cluster.
2. *Cooperative Conflict Resolution* (sec. 6.8.2) - the model used for conflict resolution in *controlled airspace*.
3. *Non-Cooperative Conflict Resolution* (sec. 6.8.3) - the model used for conflict resolution in *non-controlled* airspace and in *emergency avoidance*.
4. *Handling Standard Collision Situations* - head on approach (sec. 6.8.4), converging situation (sec. 6.8.5), overtake (sec. 6.8.6).
5. *Position Notification* (sec. 6.8.7) - position notification design.
6. *Collision Case* (sec. 6.8.8) - calculation and handling of *collision situations*.
7. *Weather Case* (sec. 6.8.9) - definition and handling of *weather hazards*.

6.8.1 Architecture

UTM Concept is based on *asynchronous event-based control* [45]. *Event* in *controlled airspace* is handled in form of *cases* [46]. There are following *event sources*:

1. *Weather Information Service* (from [47]) - used to create *weather case* (tab. 6.9).
2. *Position Notification from UAS systems* (tab. 6.6) - used to create *collision cases* (new functionality) (tab. 6.8).

Decision Frame (eq. 6.181). The *UTM* is operating in discrete decision frames which are starting on current *decision time* and ending at next *decision time*:

$$decisionFrame_i = [decisionTime_i, decisionTime_{i+1}[, \quad i \in 1, \ldots, k, k \in \mathbb{N}^+ \qquad (6.181)$$

Event-based Airspace Control is collecting events in previous $decisionFrame_{i-1}$ and issuing commands in current $decisionFrame_i$.There are following phases during the *UTM frame* cycle:

1. *Planning* - the detection phase, when the hazardous situations are assessed.
2. *Fulfillment* - the monitoring phase, when the state of affairs for directives and mandates is full filled by controlled UAS systems.
3. *Acknowledgement* - the closing phase, when UTM assess and acknowledges the performance of controlled UAS systems.

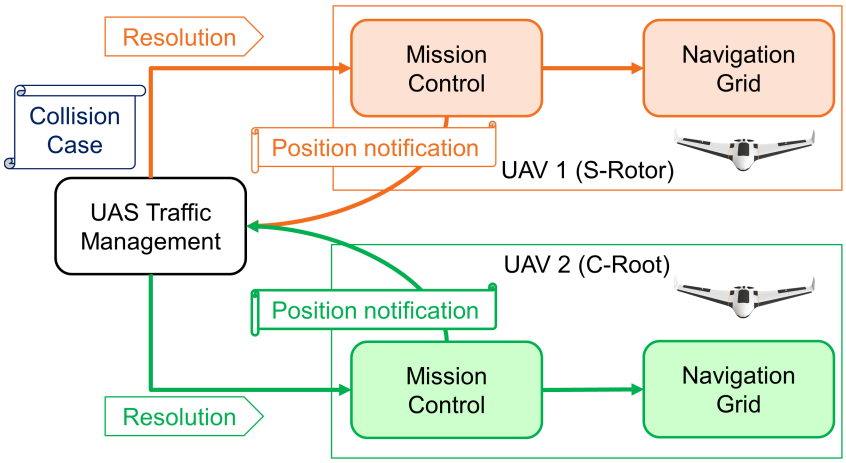


Figure 6.28: UAS Traffic Management (UTM) architecture overview.

Architecture (fig. 6.28). There are multiple UAS systems equipped with standard *Mission Control* and *Navigation* procedures.

Depending on the *airspace cluster* decision time frame they are sending *periodical position notifications* (tab. 6.6).

The *UAS Traffic Management* (UTM) collects the event data from *Weather Information Service* and *Position Notifications* calculating respective *cases*.

If there is an *active collision/weather case* the *UTM* will send *resolutions* to respective airspace attendants.

6.8.2 Cooperative Conflict Resolution

Idea: There is a *final decision maker* (absolute authority) in conflict resolution. This authority is *UTM* or *air traffic attendant* with higher priority. The future *UTM system* is such authority. The approach to mixed conflict resolution is mentioned in [48], based on navigation [49]. This is similar to our approach.

Note. Open Issue: Decentralized model with UTM as approver of directives is possible, but that is topic for own research.

Goal: UAS is obligated to follow up committed mission plan with given precision. There is one to five percent allowed deviations for ATM mission plans. Similar rates are achievable according to [48]. This requirement is given by [35] ICAO 4444 document for ATM operations.

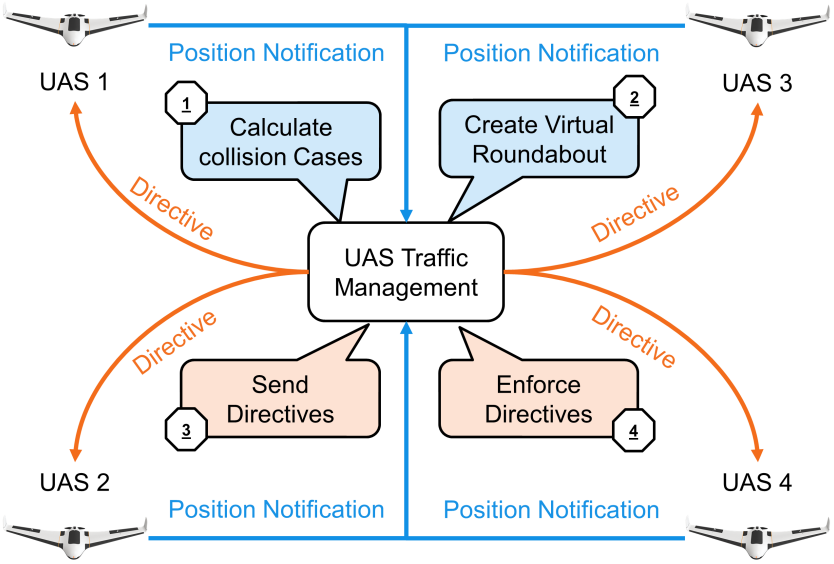


Figure 6.29: Cooperative conflict resolution via UTM authority.

Cooperative Conflict Resolution (fig. 6.29) shows functional diagram of one *UTM time-frame* there are following actors:

1. *Unmanned Autonomous System* (UAS) equipped with necessary navigation and communication modules, providing the unique *identification number*.
2. *UAS Traffic Management* (UTM) posing as central authority for given *airspace cluster*.

The following steps are executed during *Cooperative conflict resolution*:

1. $UAS_* \rightarrow UTM$ *Send position notification* - each *UAS* is notifying the authority (UTM)
2. $\circlearrowleft$ *UTM Calculate collision Cases* - UTM gathers data and predicts possible collisions then it tries to link them and manage the situation.
3. $\circlearrowleft$ *UTM Create virtual Roundabout* - active collision cases are aggregated into virtual roundabout.
4. $UTM \rightarrow UAS_*$ *Send directives* - UTM sends commands to UAS systems whom needs to change their planned trajectories.
5. $UTM \rightarrow UAS_*$ *Enforce directives* - UTM is periodically checking constraints imposed in previous *decision frames*.

6.8.3 Non-Cooperative Conflict Resolution

Idea: There is *main UAS(1)* which is flying in open *non-controlled* airspace. There are other *UAS* operating in its vicinity. It is expected that they are claiming their *planned trajectories*. The *Main UAS(1)* detects the collision with other *UAS*(2-4).

There is no *final decision maker* nor *supervising authority*, all communication participants have similar level of rights.

Note. There is assumption that other airspace users are behaving like intruders, without intent to destroy or harm. The *adversarial behaviour* is not accounted. The response from *intruder* is not mandatory in *non-controlled* airspace.

Goal: Provide *mutual avoidance mechanism* in *non-controlled* airspace. Considering the equal standpoint of all airspace attendants.

Conflict Resolution: The conflict resolution depends on current mode and *handshake* between airspace attendants. The non-cooperative behaviour have been implemented like follows:

1. *Navigation mode* - every *airspace attendant* is calculating own *collision cases* and checking the behaviour of the other (virtual UTM).
2. *Emergency avoidance mode* - is depending on communication mode:
   a *Response mode* - claiming separation methods and using avoidance mechanism (Avoidance grid with intruder model in our case).
   b *Blind mode* - every conflict side picks own strategy respecting given *rules of the air*.

Note. Intruder Intersection model selection: UAS based on Event detects possible collision for some reason UTM directive is out of question, then try to claim separation (body volume intruder model (sec. 6.6.3)), If separation fails, go full survival mode (uncertain intruder model (sec. 6.6.3)).

Special Cases in Manned Aviation: There are IFALPA reports which can give us overview of *enforced non-cooperative* mode causes in *controlled airspace*:

1. *VFR disabled* - flying in fog or thick clouds can render pilot vision, similar to UAS cameras/LiDAR.
2. *IFR equipment broke* - the sensor malfunction is more likely to happen due the lesser redundancy in UAS systems.
3. *C2C Link disabled* - communication loss is more likely to happen, due the lesser redundancy.
4. *ATM failure* - the ground control module of UTM can also fail.

Note. Traffic management related fails are lesser than 0.001 cases per one flight (according to IFALPA [50]).

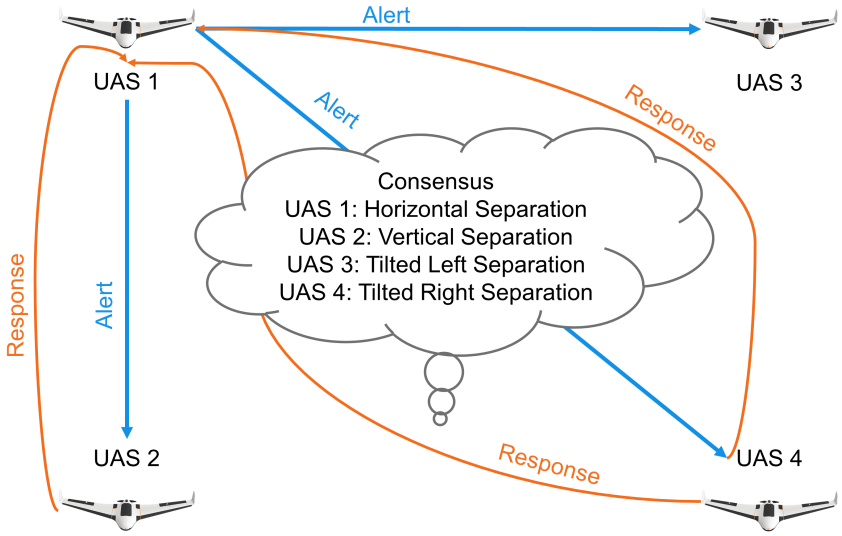


Figure 6.30: Non-cooperative conflict resolution via UAS claims.

Response mode scenario example: The *main UAS(1)* is going to collide with other *UAS*(2-4):

1. $UAS(1) \rightarrow UAS(2-4)$ sends position and heading notification.
2. $\circlearrowleft UAS(2-4)$ calculates possible collisions.
3. $UAS(2-4) \rightarrow UAS(1)$ sends response to the *main UAS(1)* with claimed separation mode.
4. $\circlearrowleft UAS(1)$ acknowledges proposed *separation modes*.
5. $\circlearrowleft UAS(1-4)$ avoids each other using claimed separation mode, because every *UAS* achieved *consensus*.

Note. The mutual consensus is not usually achieved via C2 communication. The most common case is *assuming separation mode*. This case is shown in (sec. **??**)

6.8.4 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 [51] 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".

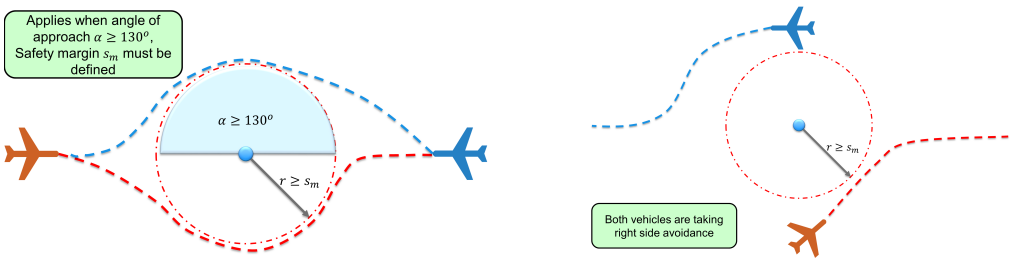
IFR: The *Instrument Flight Rules* in annex 2. [51] and 11. [52] are defining the boundaries and events for success full *Head on resolution* in larger detail.

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

1. *Angle of approach* $\geq 130°$ - the minimal planar angle between aircraft positions and expected collision point is in interval $[130°, 180°]$.

2. *Minimal detection range* - the minimal detection range of head on collision is $2 \times turningRadius + safetyMargin$.

3. *Safety margin* - during avoidance all aircraft keeps mutual distance at least at value of safety margin.



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

Figure 6.31: Head on approach detection/resolution/Closing

Triggering Events: The *head on approach* (fig. 6.31) *triggering events* are following:

1. *Detection* (fig. 6.31a) - the *collision case* is open, when *collision point* with respective angle of approach is detected. This must happen until *point of no return* is achieved.

2. *Resolution* (fig. 6.31b) - the *virtual* roundabout is enforced until the closing condition is met.

3. *Closing* (fig. 6.31b) - based on condition that all vehicles are heading away from *collision point* and heir mutual heading is neutral or opposite.

Virtual roundabout: The *flight levels* can be abstracted as *virtual 2D surface*. The *airspace attendants* are moving on virtual routes which can cross each other. The idea is to create virtual roundabout with enforced velocity to enable smooth collision avoidance.

1. *Center* - the center defined in *airspace cluster* local coordinate system (flight level defining the horizontal placement).
2. *Diameter* - the minimal distance to *center*, accounting the *wake turbulence* and other phenomenons.
3. *Enforced velocity* - all attendants at *virtual roundabout* keeps same velocity. It helps to keep constant mutual distances.

6.8.5 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 [51]. The rule is different from *Head on Approach* (sec. 6.8.4), because there are multiple roles depending on relative aircraft position:

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

The *avoiding aircraft* should take *right of the way aircraft* from behind, with sufficient *safety margin*, and return to original *heading* afterward. The *magnitude* of *avoidance curve* must consider *wake turbulence* and other impacts of *avionic properties*.

Note. This rule is applied only when the both *aircraft* belong to same *maneuverability class* [51].

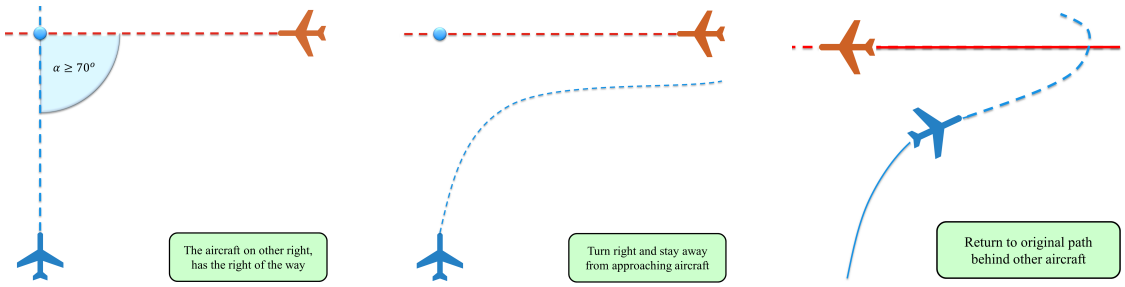
IFR: The *Instrument Flight Rules* in annex 2. [51] and 11. [52] are defining *converging maneuver* in detail.

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

1. $70° \leq$ *Angle of Approach* $< 130°$ - the minimal planar angle between aircraft position and expected collision point is in interval $[70°, 130°[$.
2. *Minimal detection range* - given as $turningRadius + safetyMargin$, while *safety margin* is accounting all impact factors.
3. *Safety margin* - during avoidance all aircraft keeps mutual distance at least on value of *Safety Margin*.

Note. Lesser *angle of approach* induces stronger wake turbulence impact on avoiding aircraft. This results into increase of *safety margin*.

The *wake turbulence* is represented as droplet at the back of the plane. *Wake turbulence range* can be calculated based on wake turbulence cone.



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

Figure 6.32: Converging maneuver Detection/Resolution/Closing

Triggering Events: The *converging maneuver* (fig. 6.32) *triggering events* are following:

1. *Detection* (fig. 6.32a) - The *avoiding airplane* (blue) detects *collision point* (blue circle) which satisfy the *converging maneuver conditions*. The distance between *aircraft position* and *collision point* is lesser than *detection range*.

2. *Resolution* (fig. 6.32b) - the *Right Of the Way aircraft* (red) stays at the original course. The *avoiding aircraft* (blue) follows the *parallel* to other *plane*. The distance of *avoiding plane* to *other plane trajectory* is greater or equal to *safety margin*.

3. *Closing* (fig. 6.32c) - when both planes have opposite heading and they miss each other the converging maneuver can be closed. The *avoiding airplane* will return to *original trajectory*, while keeping the distance from *other plane* (red) at greater or equal to *safety margin*.

6.8.6 Handling Overtake Maneuver

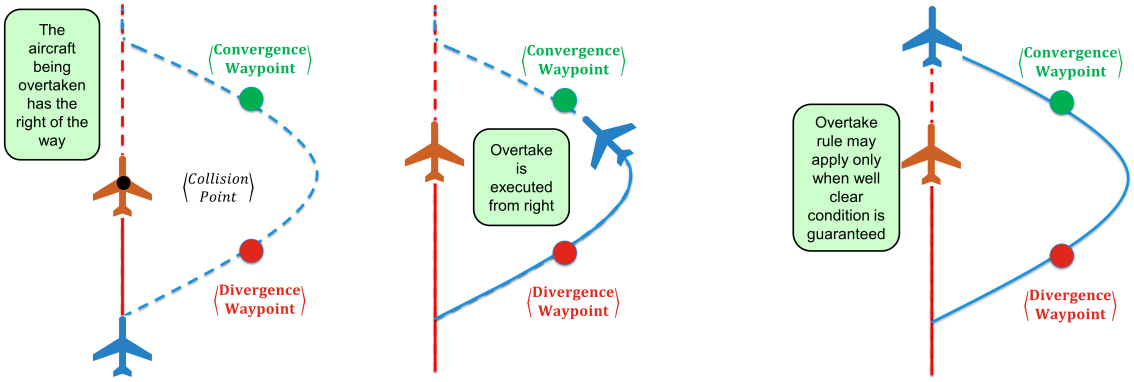
Goal: Identify *required parameters* sufficient for automatic solution of *Overtake Maneuver*

VFR: The *Visual Flight Rules* (VFR) are specified in annex 2 [51]. The rule states that faster air traffic attendant may overtake slower one, from right side keeping sufficient distance (*safety margin*). There are two forced roles:

1. *Overtaking* - faster aircraft with similar heading cruising in similar altitude than *overtaken* (blue). It is expected that *faster aircraft* has maneuvering capability to avoid slower aircraft.

2. *Overtaken* - slower aircraft which keeps the *Right of the way*

Note. This rule is applied only when both aircraft have same maneuverability class [51]. 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 6.33: Overtake maneuver Detection/Resolution/Closing

IFR: The *Instrument Fight Rules* in annex 2. [51] and 11. [52] are defining the converging manual in detail:

1. $0° \leq$ *Angle of Approach* $< 130°$ - the minimal planar angle between aircraft position and expected collision point is in interval $[0°, 70°[$

2. *Minimal Detection Range* - given as $2 \times reactionTime \times speedDifference$.

3. *Safety Margin* - during avoidance the overtaking aircraft keeps the minimal distance of *wake turbulence* of overtaken aircraft in own flight altitude.

Note. The *Safety Margin* is sufficiently small, because speed difference is usually much lesser than in case of *Head on approach*. The *Wake turbulence* can be avoided completely by taking the higher altitude level than overtaken aircraft.

Triggering events:

1. *Detection* (fig. 6.33a) - occurs when the distance between *overtaking* (blue) and overtaken (red) is approaching *minimal detection range* or double of *safety margin*. If the performance of *overtaking aircraft* (blue) allows to take *sharp right side overtake* the *Maneuver starts*, otherwise *overtaking aircraft* (blue slows down) and keeps at least *safety margin distance* to avoid *wake turbulence*.

2. *Resolution* (fig. 6.33b) - *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. 6.33c) - the *closing* of *Overtake* starts when *overtaking* aircraft (blue) have sufficient lead over *overtaken* aircraft (red). The *overtaking* aircraft (blue) can safely change the heading to original waypoint.

Constant Cruising Speed: Most of traffic attendants at same flight level have similar (close to constant) cruising speed. Lower flight levels are for slower turbo-prop planes and higher altitudes are for jet planes. It is stated that this principle will persist even when UAS will be integrated [53, 54, 55] in multiple air-traffic models.

6.8.7 Position Notification

Motivation: The *position notification* (tab. 6.6) is designed for further *collision case resolution* (sec. 6.8.8). It is similar to ADS-B[4] message information.

The main purpose is to broadcast the *position notification* in *controlled aerospace*. The broadcast for *non-controlled* airspace needs to contain *intruder properties*, *preferred separation mode* and *near miss margin*.

Position: The position is defined in *Global Coordinate System* using GPS for latitude and longitude. The barometric altitude is required for controlled airspace, preferred for non-controlled airspace.

Heading: The *Linear Velocity* combined with heading in standard *North-East* coordinate frame is used.

Flight Levels: The *flight level* is notified to UTM for *collision detection* purposes. There is *main flight level* where *aircraft* belong physically. There is *passing flight level* form which/to which is aircraft emerging [35].

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

2. *Balloon* (manned) [51] - the aircraft with *altitude* control and very slow dynamics implying very low maneuverability.

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

3. *Glider* (manned) [51] - the aircraft with *full control* but without own *propulsion*. The overall *maneuverability* is good, but the *velocity* changes are impossible with sufficient flexibility.

4. *Aerial towing* (manned) [51] - 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) [51] - the airship have *own propulsion* and full maneuverability, the constraint is low acceleration/deceleration and huge turning radius.

6. *Other manned aviation* [51] - containing all vehicles with required level of *airworthiness* for given operational *altitude*. They usually have required maneuverability.

7. *UAS Autonomous* (proposed) [56] - containing all autonomous UAS, the lower flexibility is expected at beginning of integration.

8. *Remotely Piloted Aerial System (RPAS)* (proposed) [56] - has lesser priority due the higher response rate of the pilot.

Note. This categorization reflects only Pilot community statement, the general priority rule is broken, because maneuverability and vulnerability should be always considered as key decision factor.

Maneuverability: The maneuverability is the real key factor in priority assessment. The components of maneuverability are *maximal/mean acceleration/deceleration*, *climb/descent rate* and *turning ratio/radius*. The comparison can be done by solving *pursuit problem* using *Reach Sets* [57, 58].
The *Maneuverability categorization* is based on *original aircraft priority categorization* [51] accounting UAS/RPAS as equal to *manned aviation*. The ordered list from the highest to the lowest priority goes as follows:

1. *Impaired control* (Distress aircraft) - any aviation attendant in distress has the priority in case of the conflict occurrence.

2. *Altitude control/No* (Balloon, Hovering aircraft) - the balloon type crafts does not have any type of propulsion and horizontal movements follows the airflow in given altitude.

3. *Full control/No propulsion* (Gliders of any sort) - the gliders can control their horizontal position, but there are limits to altitude control and acceleration/deceleration.

4. *Full control/Linear propulsion* (Any aircraft of plane type) - the *towing aircraft's* and *airplanes* belong there, the difference is the *flexibility* of *maneuvering*.

5. *Full control/VTOL capability* (Any aircraft with VTOL) - the *other aircraft* capable to do on spot turn. The typical representative is *quad-rotor copter*.

There are other aspects like *minimal required* acceleration/deceleration/turn ratio to operate in selected segment of the *airspace*. These should be specified later by *Minimum Operational Performance Standards* (MOPS).

Position	
latitude	based on GPS/IMU sensor fusion.
longitude	based on GPS/IMU sensor fusion.
altitude	barometric altitude *Above Mean Sea Level* (AMSL).
Heading	
orientation	orientation in standard North-East coordinate frame.
velocity	relative UAS velocity.
Flight Levels	
main	flight level, where UAS mass center belongs
passing	flight level, during climb/ascend, or when distance of UAS mass center to flight level boundary $\leq 250ft.$.
Registration	
registration ID	is unique registration number *to be issued* by local aviation authority for UTM communications purposes.
flight code	or mission code is unique identification number for approved mission plan which is going to be flown by UAS.
UAS name	optional UAS identifier to increase human recognition.
Categorization	
craft category	ICAO main category, based on vehicle type.
maneuverability	secondary categorization specifying size class, horizontal/vertical turning radius, minimal and maximal cruising speed.
Safety margins	
universal	minimal safety margin for any avoidance situation
head on	minimal distance from other similar maneuverability class aircraft in case of head on approach.
converging	minimal distance from other similar maneuverability class aircraft in case of head of converging maneuver.
overtake	minimal distance from other similar maneuverability class aircraft in case of overtake maneuver.
wake angle	for wake turbulence cone.
wake radius	for wake turbulence cone.

Table 6.6: Time-stamped *position notification* structure.

Safety Margins: The *Safety Margin* for *Well Clear Condition* value is based on *situation*. There is also an *universal safety margin* which guarantees the minimal safety for encountering intruder.

The most prevalent effect is *Wake turbulence* therefore *wake turbulence cone* angle $[0 \circ -90\circ]$ and radius.

The *safety Margin* for situation-based avoidance is given by list of supported manuevers maneuvers, there is converging (sec. 6.8.5), head on (sec. 6.8.4), overtake (sec. 6.8.6) safety margins

6.8.8 Collision Case

Collision Case Purpose: There is a need for detection and tracking of possible *controlled airspace traffic attendants* collisions. The presented *collision case structure* (tab. 6.8) is minimalist reflection of *ATM* requirements. Following aspects of *collision case* life cycle are explained in this section:

1. *Base terminology* - the definition of *enforcement procedure* and difference between *Resolution* and *Mandate* from UTM authority. The *severity issue* is open.

2. *Calculation of single case for single decision frame* - step by step calculation and threat evaluation. Prequel to *life cycle*.

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

4. *Merge procedure for multiple cases in single cluster* - the naive *merge procedure* to solve *multiple collision cases* via *virtual roundabout*.

Resolution/Mandate Enforcement: *Enforcement procedure* is consisting from *Threat detection phase* and *Mitigation phase*. The *mitigation phase* is a time interval when *UTM* decision is enforced. The decision the UTM is enforcing is delivered in form of *Resolutions* and *Mandates*.

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

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

Threat Severity Evaluation: The threat severity evaluation is omitted partialy, all threats are considered as equal. All commands from *UTM authority* will be considered as *resolutions*.

Calculation procedure: Collision case is calculated for two *Registered UAS systems* in *Unified UTM time-frame*. The *unified UTM time-frame* is a short period of time in future when the anticipated situations are predicted.

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

$$\begin{gathered} adjustedPosition = Position(Trajectory(notifiedState, futureMovements)) \\ adjustedOrientation = Orientation(Trajectory(notifiedState, futureMovements)) \end{gathered} \tag{6.182}$$

For other cases standard linear prediction can be used:

$$\begin{gathered} adjustedPosition = notificationPosition \times notificationVelocity \times timeDifference \\ adjustedOrientation = notificationOrientation \end{gathered} \tag{6.183}$$

2nd The *maneuverability*, *craft category*, *registration ID* are taken from *position notification*.

3rd *Collision case check procedure* goes like follows:

1. *Operation space checks* - the controlled airspace and flight level must match for proceeding.

2. *Maneuverability/Category check* - the maneuverability and UAS category must match. If there is mismatch then right of the way is forced to vehicle with higher priority.

4th *Linear Intersection test* is designed to calculate *closest distance* and *time* of *linear trajectory projections*, First for given *velocity* and *position* for UAS1 and UAS2 the helper variables are calculated:

$$\begin{aligned} A &= \|velocity_1\|^2 \\ B &= 2 * ({velocity_1}^T \times position_1 - {velocity_2}^T \times position2) \\ C &= 2 \times {velocity_1}^T * velocity_2 \\ D &= 2 * ({velocity_2}^T \times position_2 - {velocity_2}' \times position_1); \\ E &= \|velocity_2\|^2 ; \\ F &= \|position_1\|^2 + \|position_2\|^2 ; \end{aligned} \tag{6.184}$$

Then the projection parameters can be calculated:

$$\begin{aligned} time &= \frac{-B - D}{2 \times A - 2 \times C + 2 \times E} \\ destination_i &= position_i + velocity_i \times time, \quad i \in \{1, 2\} \\ collisionPoint &= \frac{destination_1 + destination_2}{2} \\ collisionDistance &= \|destination_1 - destination_2\| \end{aligned} \tag{6.185}$$

If $time < 0$ the trajectories are diverging from each other (because the closest points already occurred). The procedure ends, *collision flag* is not raised.

If $time > timeMargin$ the trajectories will get close to each other, but in further future and changes are anticipated. The procedure ends, *collision flag* is not raised.

If $0 \leq time \leq timeMargin$ the trajectories are converging to each other and distance needs to be checked. If $distance \leq collisionMargin$ then *collision flag* is raised and *collision point* is set.

Note. Collision Margin is some number which is determined based on aircraft category and maneuverability. Our work defines collision margin as follow:

$$collisionMargin = \forall situation : \max \left\{ \begin{matrix} safetyMargin(situation, UAS1) \\ +safetyMargin(situation, UAS2) \end{matrix} \right\} \tag{6.186}$$

Where *safety margin* for every possible situation is evaluated for both *UAS*.

5th The *trajectory* intersection is *Movement Automaton* specific collision detection method. Its based on the assumption that *UTM* have following information from *mission plan*:

1. *UAS state* - not only *position*, *orientation*, and, *velocity* vectors, but other mathematical model parameters mandatory for *movement automaton*.

2. *Movement Automaton* - movement automaton for our UAS system, so UTM can use it in predictor mode.

3. *Future Movements set* - up to reasonable prediction horizon $timeMargin$.

The *Movement Automaton* can be used as trajectory prediction for system initial state and future movements. The prediction function (eq. 6.187).

$$Prediction : UAS \times state \times futureMovements \rightarrow [x, y, z, t] \in \mathbb{R}^4 \tag{6.187}$$

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. 6.187) is given as minimal distance of projected synchronized trajectories for UAS1 and UAS2. In our discrete enviroment the *collision distance* is given as (eq. 6.188).

$$collisionDistance = \min \left\{ \|point_1 - point_2\| : \forall \begin{pmatrix} point_1 \in Prediction_1, \\ point_2, \in Prediction_2, \\ t_1 \sim t_2 \end{pmatrix} \right\} \quad (6.188)$$

If $collisionDistance \leq collisionMargin$ condition is met, *collision flag* is set.

The collision point is then calculated as mean of *UAS positions* in prediction at time when distance is minimal. The final collision point is arithmetic mean of two positions (eq. 6.189).

$$collisionPoint = \frac{point_1 - point_2}{2} : \begin{pmatrix} point_1 \in Prediction_1, \\ point_2, \in Prediction_2, \\ t_1 \sim t_2 \text{ at minimal distance} \end{pmatrix} \quad (6.189)$$

Note. Collision point is overwritten by trajectory intersection (specific) method, the *linear intersection* is considered *general collision detection method*. The collision detection method in future UTM system needs to be determined. The *Trajectory intersection* method presented in this work is one of the possible candidates.

6th *Role determination* phase is invoked if and only if previous conditions are met and *collision flag* with *collision point* exists.

There is *adjusted position* of each UAS used as verticals and *collision point* used as center. First step is normalization of adjusted position around collision point for both UAS:

$$normalized_i = adjustedPosition_i - collisionPoint, \quad i \in \{1, 2\} \quad (6.190)$$

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

$$angleOfApproach = \left| \text{atan2} \begin{pmatrix} normalized_1 \times normalized_2, \\ normalized_1 \circ normalized_2 \end{pmatrix} \right| \quad (6.191)$$

Based on *angle of approach* the *scenario type* is decided like follows:

1. $130° \leq angleOfApproach \leq 180°$ - the scenario type is set as *Head On Approach* (sec.6.8.4)

2. $70° \leq angleOfApproach < 130°$ - the scenario type is set as *Converging Maneuver* (sec.6.8.5)

3. $0° \leq angleOfApproach < 70°$ and *different speed* - - the scenario type is set as *Overtake Maneuver* (sec.6.8.6)

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

1. *Head On Approach* enforces following:
   a. The *avoidance role* us set as *RoundAbounting* for both UAS.
   b. None of the *UAS* does not have the *Right Of the Way.*
2. *Converging Maneuver* enforces following:
   a. *UAS* without free right side have role set as *Converging.*
   b. *UAS* with free right side have the *Right Of the Way.*
3. *Overtake Maneuver* enforces following:
   a. *Slower UAS* has *Overtaken* role with *Right Of the Way.*
   b. *Faster UAS* has *Overtaking* without *Right Of the Way.*
   c. *Faster UAS* mission plan is altered with *divergence and convergence waypoints.*

7th *Safety Margin Calculation* Is invoked when the collision case is *Active.* The *Active Collision Case* in this time-frame means that *Collision Flag* is raised. The *avoidance role* determines *safety margin calculation.*

If *Head On Approach* is case type of *Head collision case* then *safety margin* is calculated as maximum of sum of *default* margins or *head on* margins:

$$safetyMargin = \max \begin{Bmatrix} default(UAS1) + default(UAS2), \\ headOn(UAS_1) + headOn(UAS_2) \end{Bmatrix} \tag{6.192}$$

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

$$safetyMargin = \begin{cases} uas1.role = Converging: & \max \begin{Bmatrix} default(UAS2), \\ converging(UAS1) \end{Bmatrix} \\ uas1.role = Converging: & \max \begin{Bmatrix} default(UAS1), \\ converging(UAS2) \end{Bmatrix} \end{cases} \tag{6.193}$$

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

$$safetyMargin = \max \begin{Bmatrix} default(UAS1), default(UAS2), \\ overtaken(UAS_1), overtaking(UAS_2), \\ overtaking(UAS_1), overtaken(UAS_2) \end{Bmatrix} \tag{6.194}$$

Collision Case Chaining is procedure when multiple active collision cases for different *time-frame* are chained and creates the time ordered series of *collision cases*. There are two notable instances in the *chain*:

1. *Head Collision Case* - Collision case when the first danger was detected. The notable parameters are *collision point* and UAS *avoidance roles*, because these are enforced by *Rule engine* (sec. 6.9). The *head collision case* is first in chain.

2. *Tail Collision Case* - Collision case when the *collision danger* was not detected. The *tail collision case* is last in the chain.

Note. The *Chaining* of *collision cases* is rather primitive and sensitive for errors/noise.

The *Consistency of Avoidance Manuever* is ensured by enforcing *head collision case* parameters.

Data for both attendants	
adjusted position	predicted from previous *position notifications* (6.6) data at time of *UTM decision frame* start.
adjusted orientation	predicted from previous *position notifications* (6.6), *mission plan*, and *expected velocity*.
velocity	proclaimed velocity for given *UTM decision time frame*.
registration ID	is unique registration number issued by local aviation authority
craft category	from *position notifications* (6.6).
maneuverability	from *position notifications* (6.6).
mission plan	is acquired from *allowed mission registers* where it has been registered prior UAS flight
safety margins	list of all safety margins, derived based or craft categorization or overridden by *position notifications* (6.6).
avoidance role	is given based on situation evaluation.
trajectory prediction	simulated based on *position notification* (6.6) and *mission plan*.

Table 6.7: 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* [59]. Example of *airspace clustering* is given it [60].

The main idea is to *encapsulate multiple collision cases* into one virtual roundabout to ease *traffic load* [61]. The potential risk on *turbo roundabouts* have been outlined in [62].

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

period.

The *Collision point* is replaced with *roundabout center* point (eq. 6.195). 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.

$$roundaboutCenter = \frac{\sum_{\in Cluster}^{\forall collisionCase} collisionCase.collisionPoint \times weight}{|collisionCase \in Cluster|} \tag{6.195}$$

Note. The weight in (eq. 6.195) 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 [27, 28] is used to determine safety margin in our approach. The *naive approach* determining *roundabout safety margin* is to take the maximum of all open case *safety margins* including default ones (eq. 6.196).

$$safetyMargin = \max \begin{Bmatrix} case.UAS_i.roundaboutSafetyMargin, \\ case.UAS_i.defaultSafetyMargin \end{Bmatrix},$$
$$\forall case \in Cluster, \quad UAS_i \in \{1, 2\} \tag{6.196}$$

Collision case calculated data	
linear intersection	is predicted on attendants *position*, *heading*,*velocity*, based on *maneuverability* certain thresholds are applied to determine safety properties.
trajectory intersection	is predicted on attendants *position*, *velocity*, *heading*, and *related mission plans*, based on *maneuverability* certain thresholds are applied to determine safety properties.
collision point	is created if there is risk of medium/short time period collision, if head collision case has not been closed, collision point is inherited.
adj. collision point	is created if there exists at least one active collision case in nearby surroundings of this case collision point (cluster).
angle of approach(α)	is calculated based on attendants *velocity* and *position*, range is $[0°, 180°]$, it determines *primary avoidance roles*.
safety margin	is calculated based on *avoidance roles*, *maneuverability*, collision indicators, and *angle of approach*.
margin adjustment	is calculated based on *linked collision cases*, *estimation errors* and *weather*.
linked cases	contains list of collision cases which are active and can have impact on this *collision case*.
head case	is reference to collision case in time frame when it was first opened.
Collision case indicators	
linear intersection	indicates if there was safety breach on linear trajectories estimation with risk of direct collision.
trajectory intersection	indicates if there was breach on trajectory estimation, with risk of direct collision.
well clear breach	indicates if *linear projection* or *trajectory projection* breaches *well clear barrel* in *controlled airspace*.
active case	indicates if case is still open.

Table 6.8: Collision case structure for given decision time-frame.

6.8.9 Weather Case

Motivation: The weather, as defined in (eq. **??**), impacts flight and system dynamics, therefore it impacts the *reach set* is impacted. The *weather impact* can be solved by policy application:

1. *Weather Acceptance* - for bigger *UAS* the normal weather impact does not pose significant risk. The *segmented movement automaton* (def. 8) with *Weather situation* as discrete state is used.

2. *Weather Avoidance* - all *weather* impact zones are considered as hard constraints with protective *soft constraint* around.

3. *Combined approach* - depending on type of impact and declared UAS impact resistance the zones are divined into *soft* and *hard* constraints.

Note. This work handles small *UAS* avoidance, these are very sensitive to any weather impact, therefore *Weather impacted areas* will be considered as *hard constraints with soft constraint protection zone.*

The original *weather impact zone* is considered as obstacle body and enforces the body margin.

The surroundings of *weather impact zone* up to *safety margin* distance is considered as *soft constraint zone* (implemented as bloated polygon).

Purpose: The *weather case* (tab. 6.9) is broadcast ed by *Airspace Authority* to *impacted area*, each *UAS* then change their mission according to *their maneuvering capabilities.* Each trajectory must lead away from *constrained area.* The algorithm used for intersection selected based on [32] the selected algorithm *Shamos-Hoey* [33].

Constrained Area: Constrained area can be defined as *static* (sec. 6.5.3) or dynamic constraint (sec. 6.6.5). The *constraint center* is defined on horizontal plane like follow:

$$ConstraintCenter = center \in [latitude, longitude] \tag{6.197}$$

The *Convex Polygon* boundary is defined on horizontal plane, contains at least 3 vertexes:

$$ConvexPolygon = \{point_i : point_i \in [latitude, longitude], i \geq 3\} \tag{6.198}$$

The *Vertical constraint* is defined as *range of barometric altitude* (Above Mean Sea Level):

$$VerticalConstraint = [startAltitude, endAltitude] \tag{6.199}$$

Additional parameters : Following additional parameters with additional purpose can be attached to *Weather Constraint*.

1. *Type* - defines required resistance - moisture, temperature, wind.
2. *Severity* - defines impact for each *aircraft category*, this is used in soft/hard type assessment.
3. *Duration* - start and end of *constraint* validity, if not defined valid for all *UAS mission time*.
4. *Velocity* - velocity and last position assessment time.

Note. Our implementation does not consider the *type* or *severity*. All *weather impact* is considered as *hard constraint*. The velocity differentiates *static* $(= 0)$/*moving* (> 0) *constraints*.

Avoidance System: Resolve similar to *Converging/Overtake Maneuver* depending on the *angle of approach*. The *virtual roundabout* is utilized for *static constraints*, the *intruder model* is utilized for *dynamic constraints*.

Constrained area	
center position	is given as geometrical *center point of boundary*.
boundary	is represented as *convex polygon* on latitude-longitude plane.
start altitude	is lover boundary barometric altitude given at above mean sea level, where given weather factor have significant impact.
end altitude	is upper boundary barometric altitude given at above mean sea level, where given weather factor have significant impact.
Additional parameters	
type(s)	lists weather events occurring in *constrained area*.
severity list	is recorded for each plane *category*
start	indicates when weather constraint was established.
expected end	of weather constraint.
velocity	indicates if weather phenomenon is moving.
Miscellaneous	
previous	reference to *weather constraint* decision time-frame data.
impacted	list of possibly impacted attendees (planes whom obtained divergence order or warning from UTM).

Table 6.9: Static/Dynamic weather constraint for given decision time-frame.

6.9 Rule Engine

This section is follow up of *UTM functionality definition* (sec. 6.8), outlining realization of *UTM directives* on *UAS* side (sec. 6.9.1, 6.9.2).

Reasoning: The *Avoidance* process and *UTM directives fulfillment* is different in every national airspace. The ICAO issues recommendation [35, 51] which are implemented by every member country, some of procedures are stricter some are implemented differently.

The *UTM* collision case calculation and procedures may be universal, but their realization by *UAS* will be heavily impacted by local legislation and procedures. The *approach* must account the need of *variable parts* of *obstacle avoidance process*. The *dynamic parts* needs to be woven to hard-coded processes.

Note. Please refer to *Template Programming* and *Aspect Oriented Programming* for further explanation.

Inspiration: There was a *Maritime Rules* implementation [63] in form of *Movement Restrictions* and *Waypoint Changes*.

6.9.1 Architecture

Purpose: The *core process* of *Avoidance Grid Run* (sec. 6.7.2) and *Mission Control Run* (sec. 6.7.3) needs to be enhanced based on situation. The architecture is based on *aspect oriented approach* [64]. The key ideas and concepts are taken from rule engine implementation for multiagent navigation system [65].

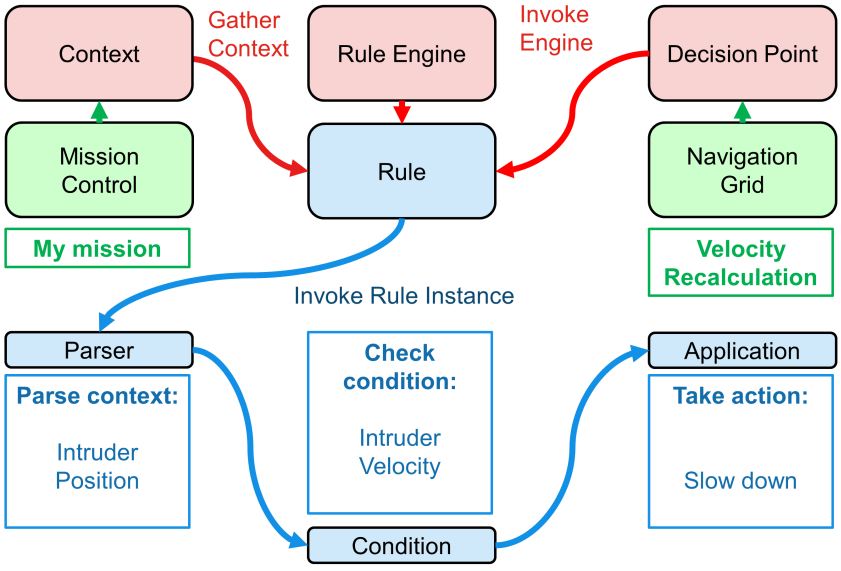


Figure 6.34: Rule engine components overview.

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*.

The rules in context of this work are pieces of code which have semi-static structure consisting from following parts (fig. 6.34):

1. *Decision Point* - hook point in process where the rule can be attached/detached. If more than one rule is hooked the priority of execution needs to be defined.

2. *Context* - the *run time context* in time of *invocation* in our case the *copy* of *Mission Control*, *Avoidance/Navigation Grid* and *Collision Cases*.

3. *Parser Method* - optional helper method to parse interesting data set from *Context*. The *parsed data* have better readability.

4. *Condition Check Method* - implementation of the trigger. If the sufficient condition is met, the rule body is applied.

5. *Rule application* - calculations and data structure changes. Mainly *disabling trajectories* in *Reach Set* in our implementation.

Example: The *UAS* is flying in controlled airspace. The *intruder* shows in front of *UAS*. The *UAS* is faster than *intruder*. The *UAS* tries to obtain permission for *Overtake*. The *UTM* does not allow *overtake*, because of *insufficient UAS maneuverability capability*. The *Rule* (fig. 6.34) with:

1. *Context* - UAS Mission Control, containing the actual mission goal and UAS IMU parameters.

2. *Decision Point* (Joint Point) - Navigation grid, containing projected constraints and reach set approximation.

3. *Rule is invoked:*

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

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

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

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

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

6.9.2 Rule Implementation

Configuration: The *Rule Engine Architecture* (fig. 6.34) is configured to handle *UTM functionality* for *Collision Case Resolution* (sec. 6.8.8). The overview of *Context* (Green), *Decision Points* (red) and *Rules to be Invoked* (cyan) is given in (fig. 6.35).

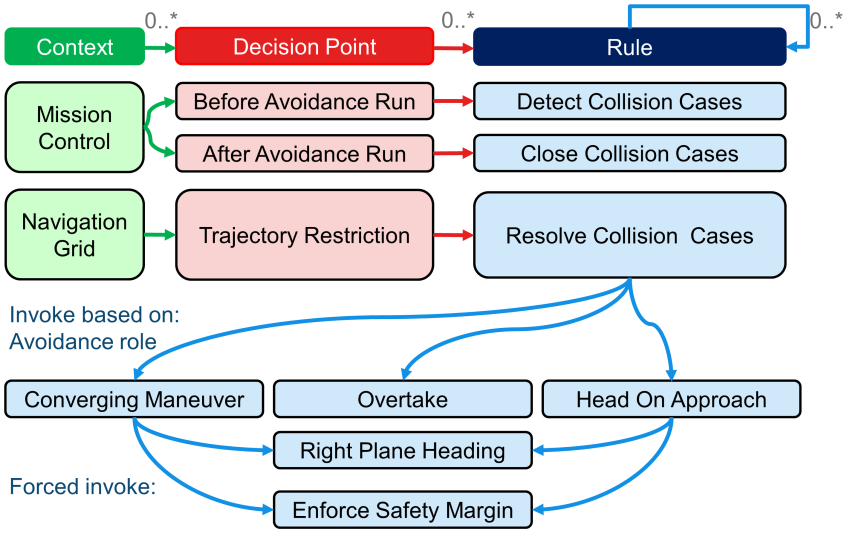


Figure 6.35: Rule engine initialization with Rules of the air.

Decision Points: The *Decisions* are bounded to *Mission Control Run Process* (fig. 6.26) in following manner:

1. *Before Avoidance Run* (before step 7.) - Context: *Mission Control* (Received Collision Cases) - the *UTM* can send directives. It is required to find which ones are impacting our *UAS*.

2. *Trajectory Restrictions* (after step 7.) - Context: *Navigation Grid* (Trajectory Restrictions) - adaptation of *behavior* imposed by *active collision cases*.

3. *After Avoidance Run* (after step 11.) - Context: *Mission Control* (Collision Case Resolutions) - our *UAS* will update the status of *Collision Cases* then it checks the *avoidance conditions*. The *Resolution Notification* resolution notifications are sent to UTM afterwards.

Note. The *Weather Case* (sec. 6.8.9) is handled in similar manner. The mission control loop (fig. 6.26) 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 following categories:

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

   a. *Detect Collision Cases* (sec. 6.9.3) - the detection of active participation in received *collision cases* and generation of *restrictions*.

   b. *Resolve Collision Cases* (sec. 6.9.4) - the enforcement of *active avoidance roles* in *collision cases*. The one *Restriction Rule* is invoked directly.

   c. *Close Collision Cases* (sec. 6.9.5) - impact calculation and *Resolution Notification* to *UTM* authority.

2. *Restriction Rules* - restricting the *Navigation Grid* trajectories or altering *goal waypoint* based on *selected collision cases*:

   a. *Converging Maneuver* (sec. 6.9.7) implementation of *Converging Avoidance* (sec. 6.8.5).

   b. *Head On Approach* (sec. 6.9.6) implementation of *Virtual Roundabout Enforcement* (sec. 6.8.4).

   c. *Overtake* (sec. 6.9.8) implementation of *overtake maneuver* for *Overtaking plane* (sec. 6.8.6).

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

   a. *Right Plane Heading* (sec. 6.9.9) - restrict all trajectories heading to space separated by parametric plane in *Avoidance Grid* which are heading or belonging to plane.

   b. *Enforce Safety Margin* (sec. 6.9.10) - restrict all *Trajectories Segments* which are in proximity of *Collision Point* lesser than *Enforced Safety Margin*.

6.9.3 Rule: Detect Collision Cases

This rule is activated each *UAS avoidance run*. *UTM* sent out all related *collision cases* (6.200) based on our *UAS identifier*. Creation of *collision case* is given in sec. 6.8.8 based on air traffic periodical*position notifications* (sec 6.8.7).

$$UTM \times timeFrame \rightarrow UTMCollisionCases \tag{6.200}$$

If there are available *position notifications* (sec 6.8.7) from surrounding air-traffic, UAS will calculate own *collision cases* (6.201).

$$uasStatus \times positionNotification \times utmTimeFrame \rightarrow UASCollisionCases \tag{6.201}$$

Then UAS merges *own collision cases* with *UTM collision cases*, if there exist following disparities UAS will take action:

1. $distamce(ownCollisionPoint, utmCollisionPoint) \geq threshold$, send UTM notificaiton, use *utmCollisionPoint*

2. $utmMargin \geq ownMargin$, use safety margin from UTM.

3. $utmAvoidanceRole == active, ownAvoidanceRole == inactive$, use UTM avoidance role.

4. $utmCollisionCase == active, ownCollisionCase == uncertain$, use UTM provided collision case, not all *position notifications* are available.

5. $utmCollisionCase == inactive, ownCollisionCase == active$, notify UTM with new collision case, ignore collision case until UTM approves.

Note. Avoidance role is classified as *inactive* if and only if UAS has *right of the way*, it is classified as *active* otherwise.

Safety margin determined by UTM has priority, because not all calculations factors are available for UAS.

Collision Case unknown to UTM are ignored, due to safety reasons (false data spoofing), collison case is activated after UTM confirmation. If there is real intruder not confirmed by UTM it is handled via *non-cooperative* or *emergency* avoidance procedure

Selection process of active *collision cases* is based on UAS *avoidance role* in each *collision case*.

- If the *avoidance roles* are following: *Head On Approach, Converging Maneuver, or Overtake* in all *collision cases* UAS system will stay in cooperative mode.

- If there exists at least one *collision case* with *own avoidance role* or *intruder avoidance role* set as *emergencing*, the UAS will notify UTM and ask for *diversion order*, meanwhile it sets itself into *Emergency avoidance* mode.

- If there exist multiple *Overtake avoidance roles* or combination of *Overtake avoidance role* and *Other active role*, the UAS will decrease its cruising speed like follows:

$$UASSpeed = \max \begin{Bmatrix} minimalUASCruisingSpeed, \\ \min \{intruderSpeed\} \quad \forall activeCollisionCases \end{Bmatrix} \tag{6.202}$$

  During *slow-down* UAS switchs to *emergency avoidance mode* and asks for *divergence order* from UTM.

The rule is summarized in table 6.10.

Invocation: Every *Decision point* in *UAS main loop*

Objective:

1. Fetch UTM *Collision cases* for given decision time frame.
2. Create/update own *own collision cases* based on received *Position notifications* from surrounding *Intruders*.
3. Merge *Collision cases* based on *UTM priority order*.
4. Select active *collision cases* based on following conditions:
   a. *Active participation* in *collision case* where *avoidance role* $\neq$ *Right of the way*.
   b. *Collision point* is int front of UAS.
   c. *Emergency mode detection* there exist at least one non-cooperative participant.
5. Order *collision cases* based on *severity*.
6. If there is at least one *active collision case* enforce rule *Resolve collision case* (tab 6.11) for each *active collision case*.

Context	*Condition*	*Application*
UAS Mission control, Before Avoidance Run, UTM/UAS collision cases	Clean *avoidance grid*, No emergency	Active collision case selection, Prioritization

Table 6.10: Detect collision cases rule definition.

Ordering of collision cases starts if and only if the *UAS* is in *cooperative avoidance mode*. The cases are ordered for processing based on severity rating which is calculated based on:

1. *Safety Margin* - the greater safety margins are prioritized.
2. *Intruder vehicle class* - the more dangerous intruders are prioritized.
3. *Collision point distance* - closer collision points are prioritized.
4. *UAS avoidance role* - *Head on Approach* is favored upon *Converging maneuver*, due to direct collision severity.

Rule engine invocation for each *active collision case* is then applied on *descending severity sorted* list.

6.9.4 Rule: Resolve Collision Case

Active collision cases are processed one by one. All collision cases are applied to *Navigation grid*. *Navigation grid* contains all possible *trajectories* in form of *Reach set*. All *trajectories* are *reachable* at the beginning of UAS *avoidance frame*. Each application of *collision case resolution* rule disables some subset of feasible *trajectories*. For this reason are *active collision cases* sorted by severity.

It is assumed that UAS is in *cooperative avoidance mode*. If previous application of this rule forced UAS into *emergency mode* the rule is not applied to save system resources. *Emergency* mode is invoked if *rule application* disable all *trajectories* in *Navigation grid*. If there is at least one *feasible trajectory* in *avoidance grid* follow-up rule is invoked based on UAS *avoidance role*.
The rule is summarized in table 6.11.

Invocation: This rule is invoked if exists at least one *active collision case* in given *navigation grid time-frame*, moreover *avoidance grid* must be empty and *cooperative avoidance mode* is enforced.

Objective: Based on *active collision case* and *UTM directives* enforce behaviour based on *own avoidance role*:

1. *Head on approach* - rule 6.13.
2. *Converging maneuver* - rule 6.14.
3. *Overtake* - rule 6.15.
4. *Emergency mode* - switch from *active avoidance mode* to *emergency mode*.

Context	*Condition*	*Application*
UAS mission control, Trajectory restriction, Collision cases,	Active merged collision case, Resolution mandate from UTM	Enforce Rules of Air or Enforce emergency

Table 6.11: Resolve collision case rule definition.

6.9.5 Rule: Close Collision Cases

Collection of rule results detected by rule 6.10 and *resolved* by rule 6.11 is done via *context of rule engine*. For each *time-frame* and each $trajectory \in NavigationGrid$, there exists rule engine *context* query (6.203) which returns *trajectory status* and *list of applied rules*

on trajectory.

$$Context(trajectory, timeFrame) \rightarrow \{State : Enabled/Disabled, Rule(s)\} \quad (6.203)$$

Calculation of possible trajectories in *navigation grid* is using *collected rule results* (6.203). If the *trajectory state* and linked *rule reason* is sufficient, the *trajectory* is disabled for given *time frame*. *Standard navigation algorithm* (TBD - reference to section with outer navigaiton loop) is used to select *feasible trajectory*.

Rules of the air and their applicaiton in *General Aviation* cases is consistent. Increasing trafic density can impose new layers of rules, which may cause the *soft deadlock* in *manuverability*. In this case *Navigation grid* will have all *possible trajectories* exhausted. Following procedure is executed:

1. UAS switch into *Non-cooperative avoidance mode* or *Emergency avoidance mode* depending on situation severity (One conflict can be handled with *vertical separation* of conflicting aircrafts).

2. UAS broadcasts *warning message* to all nearby aircrafts, and *separation message(s)* to conflicting aircraft. *Separation message* contains *expected collision point* and *preferred separation type*.Each conflicting aircraft then reacts and sends *action notification* to UTM.

3. If UAS switchs into *emergency mode*, non cooperative avoidance using *avoidance grid* is induced. Each relevant intruder is projected as *timed body volume intruder* (TBD reference to intruder probabilistic model), where *safety margin* is used as *body radius*.

UAS notifies *UTM* with *course change*, *planned avoidance trajectory*, *avoidance mode*. *UTM* approves planned changes or sends *plan corrections* (out of scope). The rule summary is given in table 6.12.

Invocation: There exists at least one *active collision case* which had impact on *Navigation grid*.

Objective: Ensure that multiple *avoidance rules* application gives feasible *avoidance strategy*, enter into *emergency avoidance mode* otherwise. Following steps are executed:

1. *Collect rules* applied on *navigation grid* from *active collision cases*.
2. *Calculate possible trajectories for avoidance*, there may be none.
3. If there is no *feasible route*, for each *intruder* from related *collision cases*:
   a. Issue *warning message* containing *expected collision point* and *preferred separation type*.
   b. Create appropriate *intruder object* for *avoidance grid*.
   c. Calculate *evasive maneuver* based on expected *separation type*.
4. *Notify UTM* with *collision case resolution* for each *active collision case*. *Nofigy UTM* with *planned trajectory* and *avoidance mode*

Context	*Condition*	*Application*
UAS Mission control, After avoidance run, Collision resolutions	At least one trajectory in Navigation grid, Emergency check	Force *Emergency mode* OR Close Collision Case

Table 6.12: Close collision case rule definition.

6.9.6 Rule: Head on Approach

Rule (6.13) is invoked based on *angle of approach* range condition, defined *collision case* section 6.8.8. The handling of *head on* avoidance is given in section 6.8.4.

Virtual round-abound for UAS and intruder is created by UTM. The center of virtual round-abound and *corrections for participants margins* are determined based on:

1. *Collision case center* - contributes to round-abound center median point.
2. *UAS and intruder maneuverability* - determines *attendants avoidance mode* and *maximal avoidance margins*.
3. *Surrounding air-traffic* - contributes to round-abound center median point, determines ideal *ideal avoidance margins* due to *wake turbulence* prevention.

Invocation: When *UAS avoidance role* is *Head on* avoidance and *avoidance grid* is empty.

Objective: Ensure that *UAS body* does not enter into *intruder's well clear zone*.

1. Prevent *left-side leading* maneuvers (rule 6.16).
2. Prevent head on *safety margin* breach(rule 6.17).
3. Return to original course, when *navigation grid* is clear.
4. Prevent *wake turbulence* (by safety margin correction).
5. Enforce *Round-about* behaviour (by clustering collision cases).

Context	*Condition*	*Application*
UAS Navigation Grid, Collision Point, Avoidance role	None	Run rules referenced in objective listing.

Table 6.13: Head on Approach rule definition.

Virtual round-abound center is calculated as *corrected median* (6.204) taking *cluster of collision cases* and calculates median of their collision points corrected by *weather* and *wake turbulence* factor.

$$\begin{aligned} correctedMedian = & \sum_{c_i \in collisionCases} (c_i.center + correction) \, / count(collisionCases) + \\ & + corrrection(Weather) + correction(WakeTurbulence) \end{aligned} \tag{6.204}$$

Corrected margin needs to be calculated for each *participating aircraft*, because of the *virtual roundabout* center correction (6.204). Each *round-abound participant* is ordered based on importance (lowest maneuverability first). Then for each *round-abound participant* obtains *corrected margin* (6.205) calculated from *collision case safety margin*, corrections based on other *more important vehicles*, *weather*, *wake turbulence*.

$$correctedMargin = \min \left[\begin{array}{c} caseMargin + correction \begin{pmatrix} ImportantVehicles, \\ Weather, \\ WakeTurbulence \end{pmatrix} \\ maximalAvoidanceMargin \end{array} \right] \tag{6.205}$$

6.9.7 Rule: Converging Maneuver

Rule is invoked based on *angle of approach* range defined in *collision case calculation* (sec. 6.8.8). Behaviour enforced to this rule is equal to rule 6.13 except the *intruder* stays on his original path. UAS behaviour is described in section 6.8.5. The *rule summary* is given by (tab. 6.14).

Invocation: When *UAS avoidance role* is *Converging* and *avoidance grid* is empty.

Objective: Ensure that *UAS body* does not enter into *intruder's well clear zone*.

1. Prevent *left-side leading* maneuvers (rule 6.16).
2. Prevent head on *safety margin* breach(rule 6.17).
3. Return to original course, when *navigation grid* is clear.
4. Prevent *wake turbulence* encounter (by safety margin correction).

Context	*Condition*	*Application*
UAS Navigation grid, Collision point, Avoidance role	None	Run rules from objective.

Table 6.14: Converging maneuver rule definition.

6.9.8 Rule: Overtake

During overtake maneuver there is our *UAS* and *Intruder* cruising at same *flight level*. *Angle of approach* (α) is lesser than 70°. *UAS* absolute velocity is much greater than *overtaken* absolute velocity.

It is assumed that during *overtake* maneuver *overtaken* intruder will keep constant heading and velocity. If this assumption is broken *UAS* system will invoke *Emergency avoidance* procedure. *UTM* will calculate such *divergence* and *convergence* waypoints that *overtake safety condition* (6.206) is satisfied.

$$distance(uasPosition, overtakenPosition) \geq utmMargin, \forall t \in manueverTime \tag{6.206}$$

Where *utmMargin* is calculated based on *Collision case* resolution. *Main idea*is to calculate *Safe offset for Overtake maneuver*, lets have:

$$velocityDifference = \|uasVelocity - overtakenVelocity\| \quad [ms^{-1}, ms^{-1}, ms^{-1}] \tag{6.207}$$

Decision distance (6.208) is given as distance when *UTM mandate* takes effectiveness, its assumed that *UTM* knows *utm decision frame* [s]:

$$decisionDistance = velocityDifference \times uasDecisionFrame \quad [m, ms^{-1}, s] \quad (6.208)$$

Overtake middle distance(6.209) is length of hypotenuse for triangle where *positional difference* and *utm margin* for overtake are cathetuses:

$$overtakeMiddle = \sqrt{\begin{matrix} \|uasPosition - collisionPoint\|_2 + \\ + safetyMargin^2 \end{matrix}} \quad [m, \vec{m}, \vec{m}, m] \quad (6.209)$$

Safe offset (6.210) is considered as combination of *overtake middle distance* (6.209), *decision distance* and uas *waypoint reach margin.*

$$\begin{matrix} overtakeMiddle \\ safeOffset = +decisionDistance \\ +waypointReachMargin \end{matrix} \quad [m, m, m, m] \quad (6.210)$$

Note. Waypoint reach margin [m] is property of own *UAS navigation algorithm.* It represents maximal distance of vehicle position and waypoint at time when waypoint is considered reached.

Local coordinate frame: UAS and Overtaken are in Local coordinate frame heading in X^+ axis direction (X^+ front of aircrafts, X^- back of vehicles, Y^- right side, Y^+ left side, $flightLevel \rightarrow Z = 0$), Collision Point is considered as $\vec{0}$,

Divergence point (6.211) in local coordinates is given as right offset of *(*UTM margin) and *decision distance*:

$$divergence = \begin{bmatrix} 0 \\ -decisionDistance - utmMargin \\ 0 \end{bmatrix} \quad [\vec{m}, m, m] \quad (6.211)$$

Convergence point (6.212) in local coordinates is given frontal *safe offset* (6.210) and right offset of *UTM margin* and *decision distance*:

$$convergence = \begin{bmatrix} safeOffset \\ -decisionDistance - utmMargin \\ 0 \end{bmatrix} \quad [\vec{m}, m, m] \quad (6.212)$$

Convergence (6.213) and *Divergence* (6.214) waypoint in global coordinate frame is obtained via transformation function R_{XYZ} as follow:

$$\begin{matrix} divergenceWaypoint = collisionPoint \\ + R_{XYZ}(overtakenOrientation, divergence) \end{matrix} \quad (6.213)$$

$$convergenceWaypoint = collisionPoint + R_{XYZ}(overtakenOrientation, convergence) \tag{6.214}$$

Overtake rule is summarized in (tab. 6.15).

Invocation: Invoked by rule *Collision Case Resolution* (rule 6.11)

Divergence Waypoint (6.213): waypoint to diverge from original UAS path to ensure Intruder safety, with unchanged intruder velocity and heading.

Convergence Waypoint (6.214): waypoint when convergence to original UAS path is enabled, within unchanged intruder velocity and heading.

Objective:

1. Calculate *Divergence Waypoint* and *Convergence Waypoint*.
2. Enforce Divergence/Convergence waypoint during avoidance.

Context	*Condition*	*Application*
UAS Navigation Grid, Collision Point, Avoidance Role	$UASVelocity >> IntruderVelocity$	Calculate & Enforce: • Divergence waypoint, •Convergence waypoint

Table 6.15: Overtake rule definition.

6.9.9 Rule: Right Plane Heading

There is need to check if *trajectory* is heading to *right side* from *collision point*. For this purpose we need to define *separation plane in 3D environment*. *Separation plane* will be defined according to Samuelson *hyperplane separation theorem* [66].

Separation plane (6.215) is defined by three points in *global coordination frame*:

1. *UAS Position* which is fixed to given *time-frame*.
2. *Collision point* which is not equal to *uas position* by definition.
3. *Gravitational acceleration* vector fitted to *UAS position* and orthogonal to vector $(uasPosition \rightarrow collisionPoint)$.

The properties of these three points guarantees that $scale.usasPosition \neq scale.collisionPoint \neq scale.gravitationalAcceleration$ for any linear $scale \neq 0$.

$$SeparationPlane = Plane \begin{pmatrix} uasPosition, collisionPoint, \\ loc2glob(uasPosition, gravitationalAcceleration) \end{pmatrix} \tag{6.215}$$

Separation plane (6.215) in *right-hand coordinate frame* where *center* = *uasPosition* X^+ is given by vector $\vec{x^+}$ (*uasPosition*, *collisionPoint*) and Z^- is given by vector $\vec{z^-}$ (*uasPosition*, *gravitationalAcceleration*). Then *right subspace* can be defined as all points where $y \leq 0$ and *left subspace* as all points where $y > 0$.

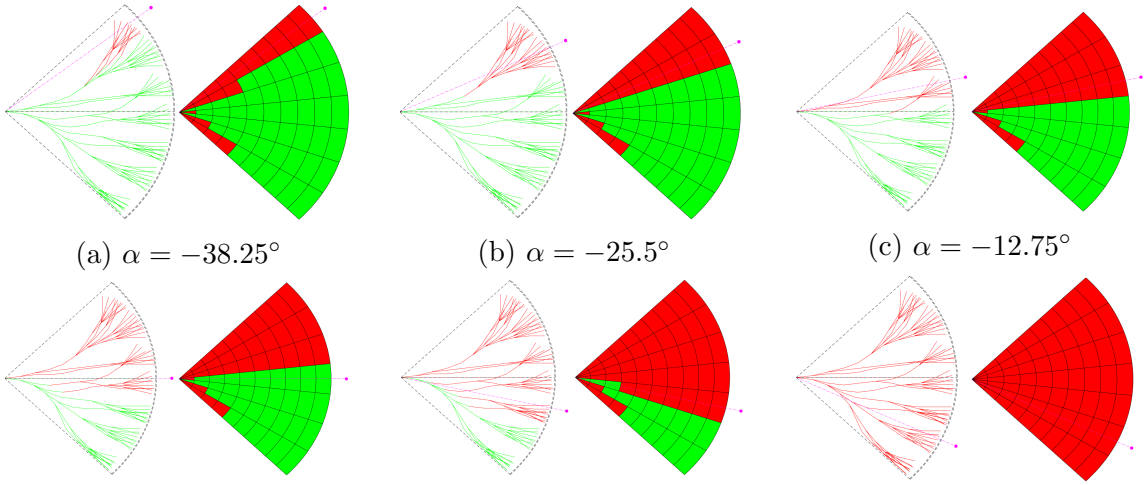
Reach set contains *trajectories*, minimal dataset for trajectory is time-series of *position* and *heading* regardless *underlying nonlinear model*. Let us have *transformation function* which can map UAS *position* and *heading* into *separation plane coordinate frame*.

The *first condition* (6.216) says that each trajectory *point* must lie within *right subspace*.

$$\forall position \in trajectory, \quad position \in rightSubspace \tag{6.216}$$

The *second condition* (6.217) needs to be applied for each *decision point*, when *trajectory* can be re-planned. It must be ensured that in time of reaching *decision point* vehicle is not heading into *left subspace* with given *turning time horizon*. The *minimal information* contains heading (velocity) vector. Checking if linear projection from *position* point with *heading* in given time-frame $[0, horizon]$ is sufficient.

$$\forall t \in [0, horizon], \quad (position + velocity * t) \in rightSubspace \tag{6.217}$$



(a) $\alpha = -38.25°$ (b) $\alpha = -25.5°$ (c) $\alpha = -12.75°$

(d) $\alpha = 0°$ (e) $\alpha = 12,75°$ (f) $\alpha = 25.5°$

Figure 6.36: Right plane heading rule evaluation for various angles of approach α.

Figure 6.36. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left subfigure). These trajectories are divided according to *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. Situation is shown for various *collision point angles of approach* α.

Rule for right plane heading check is summarized in (tab. 6.16).

Invocation: Invoked by other *maneuver rules.* *Objective:* Disable all *trajectories* in *Navigation grid's reach set* which are: 1. *Heading into collision zone* 2. *Leading into collision zone*

Context	*Condition*	*Application*
UAS Navigation Grid, Collision point (LOC)	There are feasible trajectories in Navigation Grid.	Disable trajectories in Navigation Grid.

Table 6.16: Right plane heading rule definition.

6.9.10 Rule: Enforce safety margin

Rule 6.16. checks right plane heading for single mass point along *trajectiories.* Rule needs to account *body mass* of *intruder* and UAS, other factors like safe distance, regulations etc. All mentioned factors are included in *safety margin. Safety margin* is applied as *radius ball* around *collision point.*

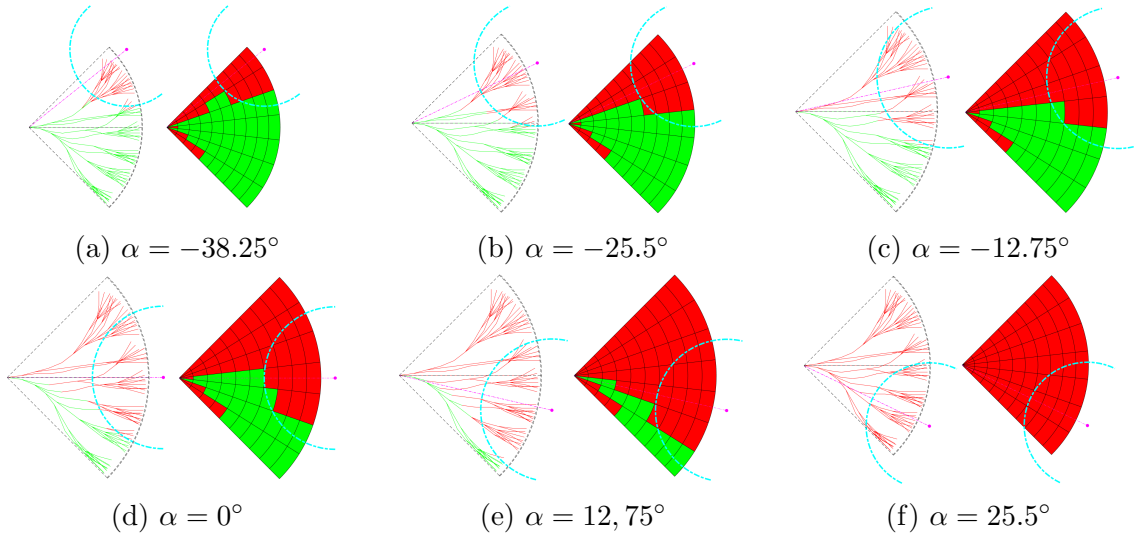
Collision point can be mapped from *global coordinate frame* to *reach set coordinate frame*, based on UAS *position and orientation* in *decision time.* Then comparison of distance between *collision point* and every *trajectory decision point* is trivial.

Trajectory feasibility condition for *non-controlled airspace* (6.218) is given as follow:

$$\forall position \in trajectory, \quad distance(position, collisionpoint) \geq safetyMargin \tag{6.218}$$

Controlled airspace must maintain *well clear condition.* To enforce protective barrel around *collision point* one must compare *global coordinates. Trajectory feasibility condition* for *controlled airspace* (6.219) is given as follow:

$$\begin{aligned} &\forall position \in trajectory, \\ &\qquad XY distance(position, collisionPoint) \geq safetyMargin \\ &\qquad flightLevelStart \geq Z(position) \geq flightLevelEnd \end{aligned} \tag{6.219}$$



(a) $\alpha = -38.25°$ (b) $\alpha = -25.5°$ (c) $\alpha = -12.75°$

(d) $\alpha = 0°$ (e) $\alpha = 12,75°$ (f) $\alpha = 25.5°$

Figure 6.37: Enforce safety margin rule evaluation for various angles of approach α.

Figure 6.37. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left subfigure). These trajectories are divided according to *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). More trajectories are disabled due to *safety margin* (teal dashed line) around the *collision point*. *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. Situation is shown for various *collision point angles of approach* α.

Rule for safety margin check is summarized in (tab. 6.17).

Invocation: Invoked by other *maneuver rules*.

Objective: Based on type of airspace, for given *collision point* and *safety margin* disable trajectories in:

1. Ball radius for *non-controlled* airspace (6.218).
2. Well clear barrel *controlled* airspace (6.219).

Context	*Condition*	*Application*
UAS Navigation Grid Collision point Safety Margin	There are feasible trajectories for condition application.	Disable trajectories in Navigation Grid.

Table 6.17: Enforce safety margin rule definition.

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] Thor I Fossen. Mathematical models for control of aircraft and satellites. *Department of Engineering Cybernetics Norwegian University of Science and Technology*, 2011.

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

[11] 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.

[12] 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.

[13] 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.

[14] Roberto Sabatini, Alessandro Gardi, and Mark A Richardson. Lidar obstacle warning and avoidance system for unmanned aircraft. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(4):702–713, 2014.

[15] Alex A Kurzhanskiy and Pravin Varaiya. Ellipsoidal toolbox (et). In *Decision and Control, 2006 45th IEEE Conference on*, pages 1498–1503. IEEE, 2006.

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

[17] 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.

[18] 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.

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

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

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

[23] 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.

[24] 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.

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

[26] 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.

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

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

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

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

[31] 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.

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

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

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

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] Subramanian Ramasamy, Roberto Sabatini, and Alessandro Gardi. Towards a unified approach to cooperative and non-cooperative rpas detect-and-avoid. In *Fourth Australasian Unmanned Systems Conference*, 2014.

[49] Subramanian Ramasamy, Roberto Sabatini, A Gardi, and Yifang Liu. Novel flight management system for real-time 4-dimensional trajectory based operations. In *proceedings of AIAA Guidance, Navigation, and Control Conference*, 2013.

[50] Branka Subotic, Arnab Majumdar, and Washington Y Ochieng. Recovery from equipment failures in air traffic control (atc): The findings from an international survey of controllers. *Air Traffic Control Quarterly*, 15(2):157–181, 2007.

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

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

[53] 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.

[54] 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.

[55] 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.

[56] 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.

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

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

[59] 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.

[60] 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.

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

[62] 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.

[63] 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.

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

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

[66] Hans Samelson, Robert M Thrall, and Oscar Wesler. A partition theorem for euclidean n-space. *Proceedings of the American Mathematical Society*, 9(5):805–807, 1958.