

## 6.7 Avoidance Concept

This section introduces *Platform Independent Avoidance Concept* core functionality (fig. ??) 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. 6.1) as minimal visibility calculated from all capable sensors in *Sensor Field*.

$$visibility(cell_{i,j,k}) = \min \left\{ \begin{array}{l} visibility(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{array} \right\} \quad (6.1)$$

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

*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*.

$$obstacle(cell_{i,j,k}) = \max \left\{ \begin{array}{l} obstacle(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{array} \right\} \times \dots \dots \times visibility(cell_{i,j,k}) \quad (6.2)$$

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

**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. 6.3) is calculated as product of maximal *Map Obstacle Rating* and *inverse visibility*. This gives *visibility biased* certainty of *Map Obstacle*.

$$map(cell_{i,j,k}) = \max \left\{ \begin{array}{l} map(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{array} \right\} \times \dots \dots \times (1 - visibility(cell_{i,j,k})) \quad (6.3)$$

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

**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. 6.4) is simply bypassing the intruder rating if there is one intruder. If there is more intruders the *geometrical bias* is applied.

$$intruder(cell_{i,j,k}) = 1 - \prod_{\forall intruder_i \in Intruders} \left( 1 - intersection \left( \begin{matrix} cell_{i,j,k}, \\ intruder_i \end{matrix} \right) \right) \quad (6.4)$$

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

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

$$constraint(cell_{i,j,k}) = \max \left\{ \begin{matrix} constraint(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \quad (6.5)$$

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

*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. 6.7). All *sources of harm* are considered as equal. Threat is formalized in *following definition*:

**Definition 1.** *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. 6.6).*

$$threat(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}), map(cell_{i,j,k}), \\ intruder(cell_{i,j,k}), constraint(cell_{i,j,k}) \end{matrix} \right\} \quad (6.6)$$

**Reachability:** The *Reachability* for trajectory reflects how safe is *path along*. The *Threat* (def. 1) for each cell has been already assessed. The set of *Passing Cells* is defined in *Trajectory Footprint* (eq. ??).

The *Trajectory Reachability* is given as product of *Threats* along the trajectory (eq. 6.7). The *Trajectory Reachability* can be calculated for each *trajectory segment* given as  $\{movement_1, \dots, movement_i\} \subset Buffer$  originating from  $state_0$ .

$$reachability(Trajectory) = \prod_{\substack{\forall cell_{i,j,k} \in \\ PassingCells}} (1 - threat(c_{i,j,k})) \quad (6.7)$$

*Note.* The *Reachability* 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 *Reachability* 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 reachability is formally defined in (eq. 6.8).

$$reachability(cell_{i,j,k}) = \max\{Trajectory.Segment(cell_{i,j,k}).Reachability : \forall Trajectory \in PassingTrajectories(celli, j, k)\} \quad (6.8)$$

*Note.* Function  $Trajectory.Segment(cell_{i,j,k}).Reachability$  gives same results for any segment in  $cell_{i,j,k}$ , because (eq. 6.7) 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* [1] 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. 6.1).

The high values of *Visibility* (eq. 6.1) and *Reachability* (eq. 6.8, 6.7) are expected. The low *threshold* for *threats* values is expected. The error margin is solved by *Sensor Fusion* therefore initial *false positive* cases have low rate. The *Detected Obstacle Rate* (eq. 6.2), *Map Obstacle Rate* (eq. 6.3), *Intruder Rate* (eq. 6.4), and *Constraint Rate* (eq. 6.5) thresholds are considered low.

Threshold = $10^{-7}$			
Visibile	$visibility(cell_{i,j,k})$	$\geq$	$(1 - threshold)$
Detected Obstacle	$obstacle(cell_{i,j,k})$	$\geq$	$threshold$
Map Obstacle	$map(cell_{i,j,k})$	$\geq$	$threshold$
Intruder	$intruder(cell_{i,j,k})$	$\geq$	$threshold$
Constraint	$constraint(cell_{i,j,k})$	$\geq$	$threshold$
Reachable Trajectory	$reachability(trajectory)$	$\geq$	$(1 - threshold)$
Reachable Cell	$reachability(cell_{i,j,k})$	$\geq$	$(1 - threshold)$

Table 6.1: 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. 6.9).

$$Uncertain(t_i) = \{cell_{i,j,k} : cell_{i,j,k} \in AvoidanceGrid(t_i), cell_{i,j,k}.\neg Visible\} \quad (6.9)$$

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. 6.2). The formal definition is the space portion where it is possible to detect *obstacle bodies* or their portions (eq. 6.10).

$$Occupied(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.DetectedObstacle \end{array} \right\} \quad (6.10)$$

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. 6.11).

$$Constrained(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Visible, \\ cell_{i,j,k}.Constraint \vee cell_{i,j,k}.Intruder \end{array} \right\} \quad (6.11)$$

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. 6.12).

$$Free(t_i) = AvoidanceGrid(t_i) - \dots \\ \dots - (Uncertain(t_i) \cup Occupied(t_i) \cup Constrained(t_i)) \quad (6.12)$$

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. 6.13).

$$Reachable(t_i) = \left\{ cell_{i,j,k} : \begin{array}{l} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Reachable \end{array} \right\} \quad (6.13)$$

*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:

$$|Reachable(t_i)| > 0, \quad Reachable(t_i) \subset Free(t) \quad (6.14)$$

## 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 [2]. Semi-optimal concatenation trajectory search like ours can be found in [3].

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

**Space Assessment Principle:** The *Avoidance Grid* is fed trough *Data Fusion* (sec. 6.7.1). The process of *ratings assessment* (tab. 6.1) is given in (fig. 6.1):

1. *Obstacle detection* (fig. 6.1a) - assessment of *detected obstacles* (eq. 6.2). The red (O) *cells* have Detected obstacle set as *true*. The other threats: *map obstacles* (eq. 6.3), *intruders* (eq. 6.4), *constraints* (eq. 6.5) are false. The red (0) *cells* are representing *Occupied*( $t_i$ ) (eq. 6.10) space in *Avoidance Grid* at decision time  $t_i$ .
2. *Uncertainty assessment* (fig. 6.1b) - the uncertain cells are cells which status can not be *assessed*. The *Visibility* (eq. 6.1) is low. The *Uncertain* cells (yellow (U) mark) are equal to *Uncertain*( $t_i$ ) (eq. 6.9) in *Avoidance Grid* in *decision time*  $t_i$ . The *Constrained*( $t_i$ ) (eq. 6.10) space is equal to  $\emptyset$  in this example.
3. *Trajectory reachability evaluation* (fig. 6.1c) - the *Reach Set* given as *Trajectory Set* (eq. ??). is then projected trough *Avoidance Grid* and pruned according to (def. ??). *Reachable Trajectories* (eq. 6.7) are only those contained in  $Free(t_i)$  space (eq. 6.12). The *Reachable Trajectories* are denoted as *green lines*. The *Unreachable* trajectory segments are denoted as *red lines*.
4. *Cell reachability evaluation* (fig. 6.1d) - the evaluation of *cells* reachability is going according to (eq. 6.8). The *Reachable cells* are those which *contains* at least one *Reachable Trajectory Segment*.

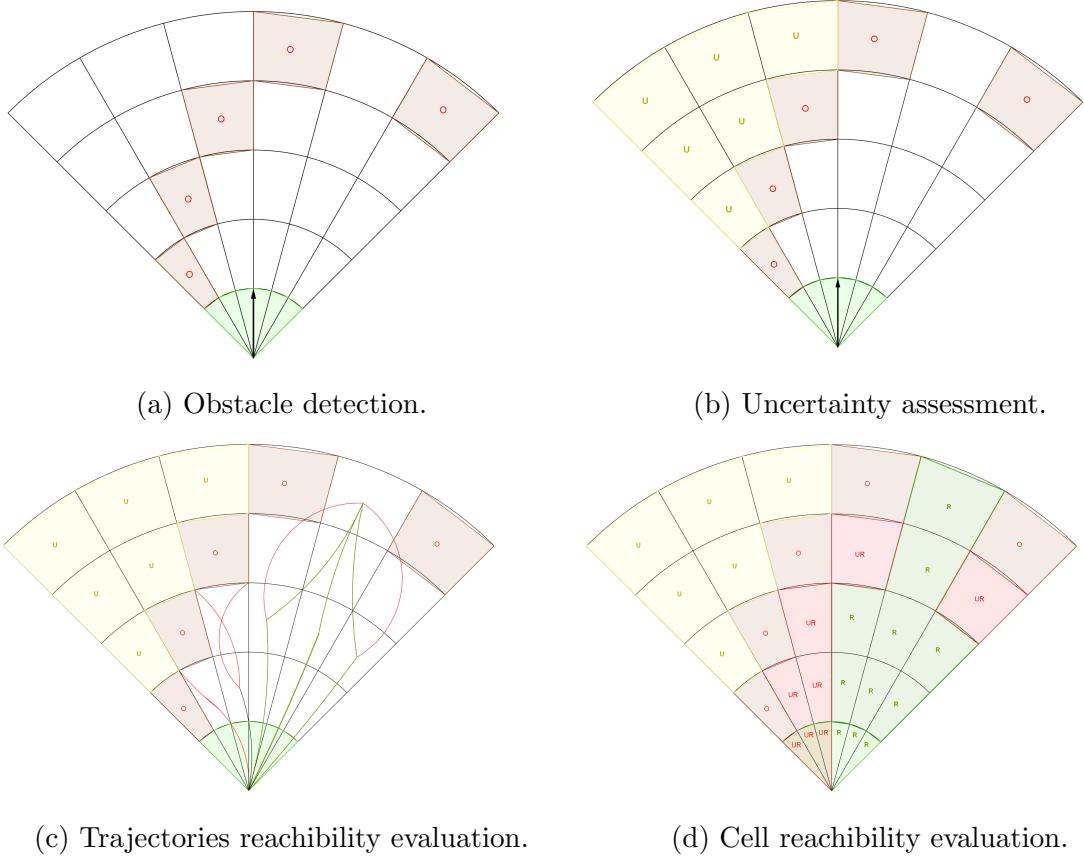


Figure 6.1: Significant steps of *Avoidance grid run* (inner loop).

**Finding Best Path:** <sup>1</sup> Each  $cell_{i,j,k}$  in *Avoidance Grid* at *decision time*  $t_i$  has assessed ratings according to *data fusion procedure* (tab. 6.1). The following properties are known prior the *trajectory* selection:

1. *Reachability* for each  $cell_{i,j,k}$  (eq. 6.8).
2. *Reachability* for each  $Trajectory(o)$  (eq. 6.7).
3. *Free Space* as non empty set of *cells* in *Avoidance Grid* (eq. 6.12), with *Reachable Space* (eq. 6.13).
4. *Goal Waypoint*  $WP_G$  from *Mission Control Run* (sec. 6.7.3).

The *Algorithm* (alg. 6.1) 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  $cell_{i,j,k}$  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*.

<sup>1</sup>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 6.1:** Find best *Path* in *Avoidance Grid*

---

**Input :** Cell[] reachable (eq. 6.13), Waypoint goal, AvoidanceGrid( $t_i$ ) grid

**Output:** Trajectory avoidancePath, Error message

# Initialization & Reachability test;

avoidancePath =  $\emptyset$ ;

**if** reachable ==  $\emptyset$  **then**

**return** [avoidancePath, "No path available, empty Reach Set"]

**end**

avoidanceCell = GetRandomCell(reachable);

# Look for for goal cell;

**if** goal  $\in$  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 $_{i,j,k} \in$  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  $\in$  avoidance Cell && trajectory.Reachable == true **do**

**if** trajectory.Cost < avoidancePath.cost **then**

        avoidancePath = trajectory;

**end**

**end**

message =  $\emptyset$ ;

**return** [avoidancePath,message]

---

*Note.* Outer cell is a 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. 6.2). 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.

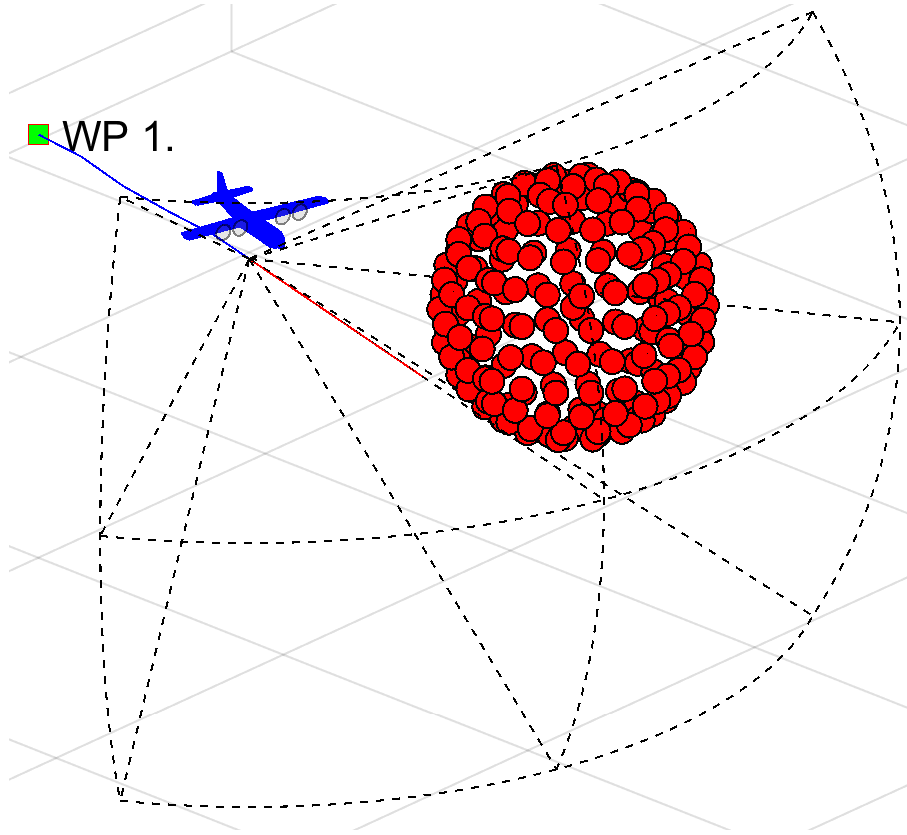


Figure 6.2: Example: The situation to be evaluated by *Avoidance Run*.

**Visibility Assessment:** The visibility assessment (fig. 6.3) divides the *Avoidance Grid* into two

1. *Visible space* (blue filled cells) is space *trough* which *LiDAR* rays roamed freely until they hit an *Obstacle*.
2. *Uncertain space* (black filled cells) is space where no *LiDAR ray* passed nor hit. Therefore its status is uncertain.

*Note.* The *detected obstacle cells* are part of *visible space*, because there is certainty about its containment.

The *Reach Set* trajectories are colored based on their visibility, blue for *uncertain* trajectories and *green* for visible trajectories.

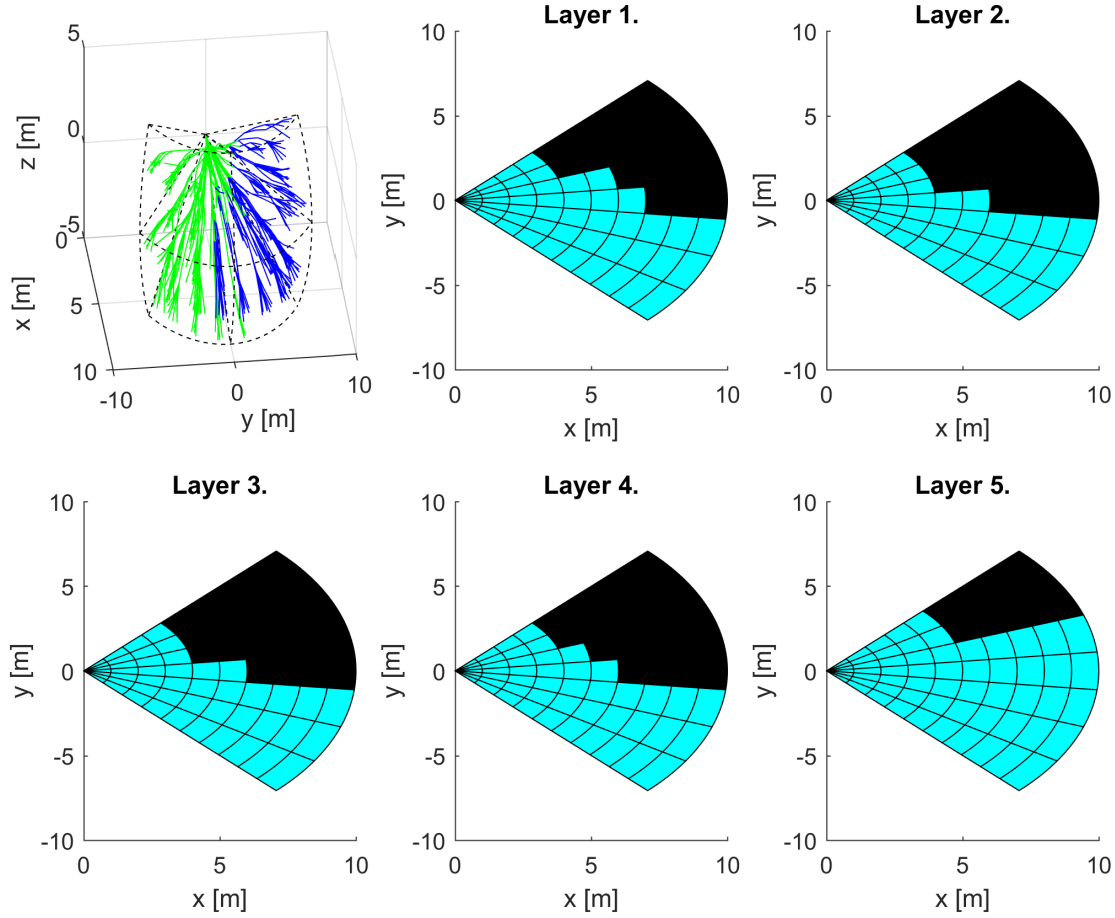


Figure 6.3: Example: The *Visibility* evaluation by *Avoidance Run*.

**Reachability Assessment:** For Each trajectory the *Reachability* is assessed (fig. 6.4). The *Obstacle Space* and *Uncertain Space* are rendering *reachability*, effectively separating *trajectories* into two categories:

1. *Unreachable Trajectories* (red lines) - there is at least one trajectory segment leading through *Obstacle* or *Uncertain* space.
2. *Reachable Trajectories* (green lines) - all trajectory segments are lying in *Free* space.

Cells in Avoidance grid are divided in 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*.

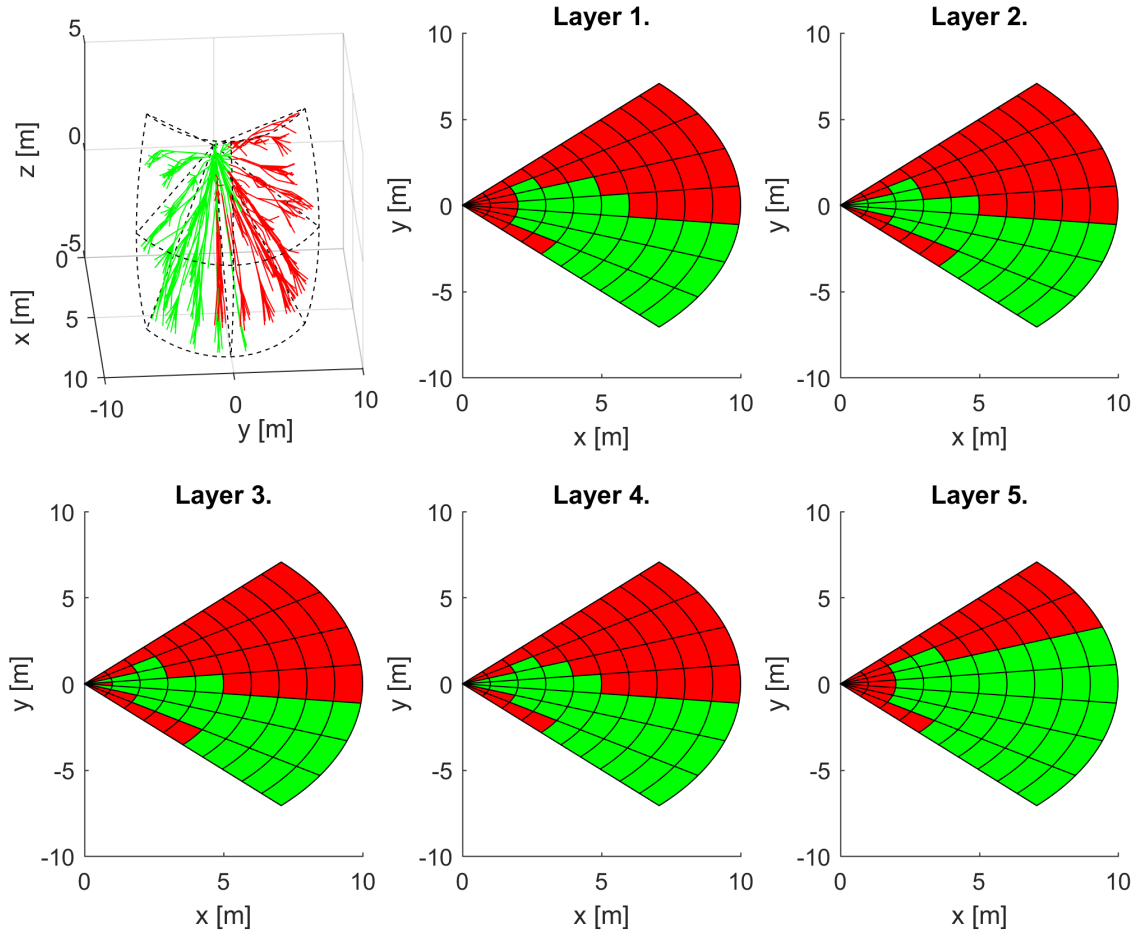


Figure 6.4: Example: The *Reachability* evaluation by *Avoidance Run*.

*Note.* The *best avoidance path* is selected from *reachable outer cells* (green fill in fig. 6.4), depending on *goal waypoint* according to (alg. 6.1).

### 6.7.3 Mission Control Run

**Introduction and Motivation:** This section will introduce *Navigation Concept* using *Reach Set Approximation*. The *Avoidance Framework Concept* (fig. ??) 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  $t_i$ .

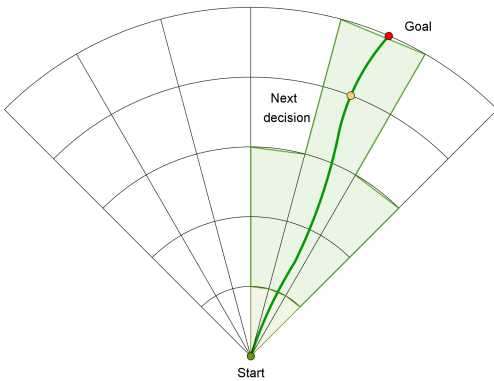
There is a need to build a trajectory between *Waypoints* which are further away than *distance* of one *Avoidance Grid*. The *UAS* is controlled via *Movement Automaton*. The *Movements* which are in *Movement Buffer* can be replaced with another movements. This feature of *Movement Automaton* is called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

To join the multiple *Avoidance Grids* paths following terminology needs to be established (fig. 6.5a):

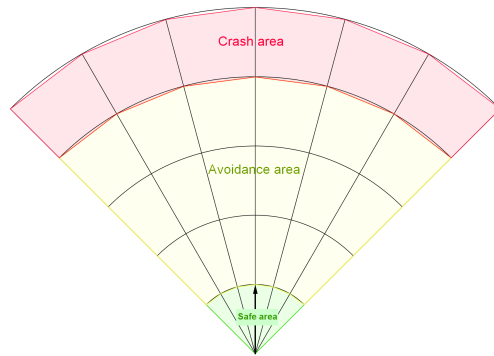
1. *Goal* (Selecting Goal of Navigation) - the point where UAS want to get in global coordinate frame. The selection needs to be defined.
2. *Next Decision* - the point when the next *Avoidance Grid Run* is applied. The outline of events and triggers is required. The *decision* will be made in *next decision time*  $t_{i+1}$ .

The *Avoidance Grid* from *UAS* viewpoint can be separated into following zones (fig. 6.5b):

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. ??) is important in this area.
3. *Safe Zone* (first layers) - there is space for safe return or damage mitigation.



(a) Mission control run example.

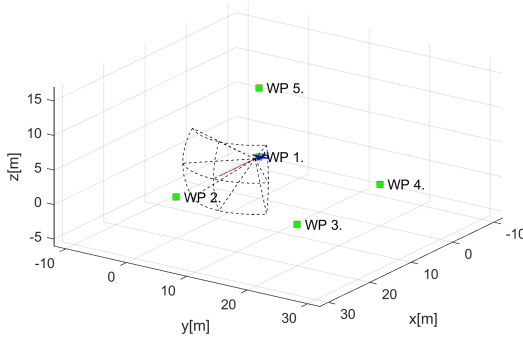


(b) Grid Zones.

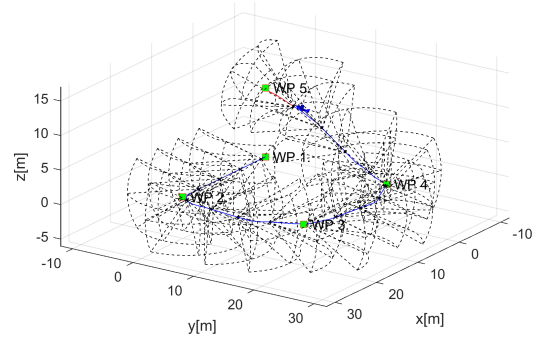
Figure 6.5: Definitions for *Mission Control Run* (outer loop).

Joining *Avoidance Grid Runs* (fig. 6.6) 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. 6.7) run is shown:

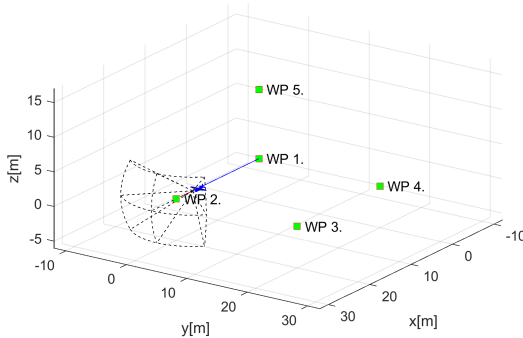
1. *Mission Start* (fig. 6.6a) - UAS at the start of the mission have one *Avoidance Grid* at its position to determine the *Navigation Path* to *Waypoint 2* (goal waypoint). The planned path (red line) is leading directly to *Avoidance Grid* boundary (black dashed line).
2. *Mission End* (fig. 6.6b) - UAS have reached *last waypoint*. All *Avoidance Grid* boundaries (black dashed line) for all *runs* are drawn along flown trajectory.
3. *Waypoint Reach* (fig. 6.6c) - 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. 6.6d) - the new *Goal Waypoint* is selected, the UAS moves to new goal (invoking *Avoidance Grid Runs* when necessary).



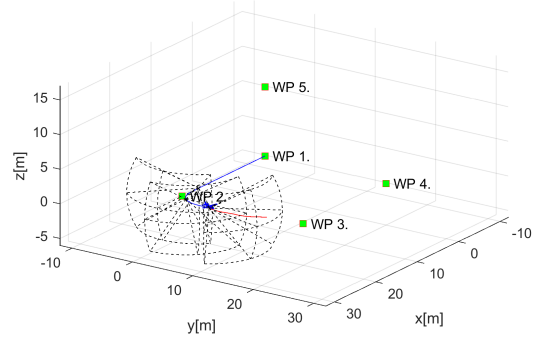
(a) Mission start.



(b) Mission end.



(c) Waypoint reach.



(d) Next waypoint.

Figure 6.6: Joining multiple *Avoidance Grid Runs* for achieve Navigation.

**General Concept:** <sup>2</sup> The *General Concept* is taken from [4, 5], 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.

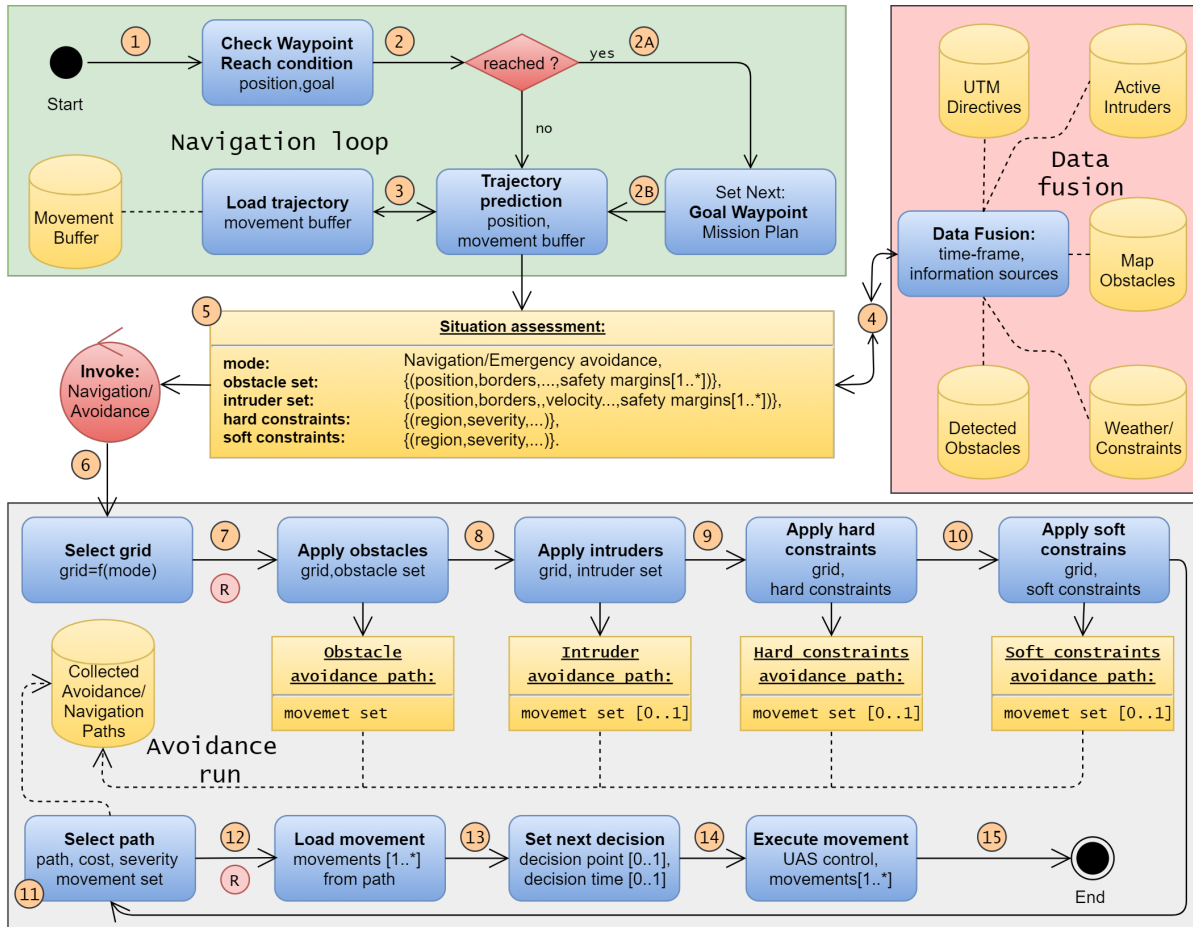


Figure 6.7: Mission control run activity diagram.

The main changes to *Navigation architecture* are given in *Mission Control Run* activity diagram (fig. 6.7):

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.

<sup>2</sup>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 an instance of *Avoidance Grid* (sec. ??) with initialized *Navigation Reach Set* (ex. *Harmonic Reach Set Approximation* (sec. ??)).
2. *Emergency Avoidance Mode* - the *UAS* is *threatened* by obstacle, intruder, hard constraint or *soft constraint*, the *UAS* is navigating through *Airspace* following *safe avoidance patterns* and *minimizing the impact* of possible damages. The *Avoidance Grid* is a term used for *Emergency Avoidance Mode*. The *Avoidance Reach Set Approximation* is initialized in *Avoidance Grid* (ex. *Chaotic Reach Set Approximation* (sec. ??))

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

The *Navigation Grid* and *Avoidance Grid* share the space 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. ??).

The *UAS System* (sec. ??) controlled by *Movement Automaton Implementation* (sec. ??) *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  $1/\text{velocity}$  second (tab. ??, ??).

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 hand, 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. ??) with given *Sensors*:

$$\text{Sensors} = \{\text{LiDAR}, \text{ADS} - B\} \quad (6.15)$$

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

The *Data Fusion Procedure* is given as follow:

$$DataFusion = \{RatingBasedDataFusion \quad (sec.6.7.1)\} \quad (6.16)$$

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

$$MovementBuffer = \{\} \quad (6.17)$$

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:

$$\begin{aligned} AvoidanceGrid(0) &= \{UAS.position(0), AvoidanceReachSet(UAS.ReachSet)\} \\ NavigationGrid(0) &= \{UAS.position(0), NavigationReachSet(UAS.ReachSet)\} \\ OperationMode &= Navigation \end{aligned} \quad (6.18)$$

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

$$\begin{aligned} Mission &= \{Waypoint_1 \dots Waypoint_n\} \\ GoalWaypoint &= Mission.waypoint_1 \\ LastWaypoint &= Mission.waypoint_n \end{aligned} \quad (6.19)$$

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

$$obstacles = \{\}, intruders = \{\}, hardConstraints = \{\}, softConstraints = \{\} \quad (6.20)$$

**Navigation Loop (1<sup>st</sup>-3<sup>rd</sup> 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. 2) and *Unreachable waypoint* (def. 3).



**Definition 2.** *Waypoint Reach Condition* for current decision time  $t_i$  for UAS position and current Goal Waypoint is satisfied only if:

$$\begin{aligned} & \text{distance}(\text{UAS.position}(t_i), \text{GoalWaypoint}(t_i)) \\ & \leq \\ & 2 \times \max \{ \text{length}(\text{movement}) : \forall \text{movement} \in \text{MovementSet} \} \end{aligned} \quad (6.21)$$

*Note.* The movements in our solution have *uniform length* of 1 m (tab. ??, ??), 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 3.** *Unreachable Waypoint.* The Goal Waypoint is evaluated as *unreachable* in decision time  $t_i$  when Avoidance Grid Run (alg. 6.1) 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:

$$\text{Grid}(t_i).\text{range} \geq \text{distance}(\text{UAS.position}(t_i), \text{GoalWaypoint}(t_i)) \quad (6.22)$$

and following condition is satisfied:

$$\begin{aligned} & \forall \text{cell}_{i,j,k} \in \text{Grid}(t_i) \neg \text{cell}_{i,j,k}.\text{Reachable} == \text{true} \wedge \dots \\ & \dots \wedge \text{distance}(\text{cell}_{i,j,k}, \text{GoalWaypoint}(t_i)) \leq \dots \\ & \dots \leq 2 \times \max \{ \text{length}(\text{movement}) : \forall \text{movement} \in \text{MovementSet} \} \end{aligned} \quad (6.23)$$

The Goal Waypoint is *unreachable*.

Then the *Navigation Loop* is invoked every decision time  $t_i$ , *Mission Control Run* (fig. 6.7), it is described as sequence of following steps:

**1<sup>st</sup> Check Waypoint Reach Condition** - the UAS position for given time frame  $t_i$  is checked under condition (eq. 6.21). If condition is met continue with 2<sup>nd</sup> step otherwise continue with 3<sup>rd</sup> step.

**2<sup>nd</sup> Set Next Waypoint** - until following condition is met:

$$\text{GoalWaypoint} == \text{LastWaypoint}$$

Set next goal waypoint like follow:

$$\text{GoalWaypoint} = \text{Mission.getNextWaypoint}()$$

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

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

$$\begin{aligned} PredictedTrajectory = \\ Trajectory(state = UAS.state(t_i), buffer = futureMovements) \end{aligned}$$

The *Predicted Trajectory* is used in 5<sup>th</sup> step *Situation Assessment*.

**Data Fusion (4<sup>th</sup> 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. 6.1) for *threat* classification.

*Note.* Avoidance Grid's Data fusion (sec. 6.7.1) is run in 7<sup>th</sup>- 10<sup>th</sup> step (fig. 6.7).

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

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

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

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

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

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

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

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

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

**Invoke Navigation/Avoidance based on Situation Assessment (5<sup>th</sup>-6<sup>th</sup> step):**

The *deciding events* depending on *Trajectory Prediction* (3<sup>rd</sup> step) and *Data Fusion* (4<sup>th</sup> step) (fig. 6.7) 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* ( $MovementBuffer = \emptyset$ ) - if there is no movement in *Movement buffer* to be executed (from 3<sup>rd</sup> step: Load Trajectory), the *Avoidance Run* is enforced to run with *Navigation/Avoidance Reach Set Approximation* to generate new path.
- b. *Waypoint Reached* (2<sup>nd</sup> 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 Reachability* (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. ??), 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. ??) 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. ??)) *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. ??), moving constraints (sec. ??)) *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.

**5<sup>th</sup> 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 7<sup>th</sup> step, otherwise wait for *next decision time*  $t_{i+1}$ , execute movement and continue with 1<sup>st</sup> step.

**6<sup>th</sup> 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. 6.12) for all cells.

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

If the *Operation Mode* is set as *Navigation Mode* the algorithm continues with 11<sup>th</sup> step. Otherwise the *Avoidance Grid Space Assessment* is run multiple times to obtain  $Reachable(t_{i+1})$  (eq. 6.13). The *Threat Data* obtained from 4<sup>th</sup> step are used.

**7<sup>th</sup> Apply Obstacles** - The *Space assessment* (tab. 6.1) for *Avoidance Grid* is calculated with following threat modification:

$$intruders = \emptyset, softConstraints = \emptyset, hardConstraints = \emptyset$$

The *Find Best Path* (alg. 6.1) is applied, the resulting *avoidance path* is labeled as *Obstacle Avoidance Path*.

**8<sup>th</sup> Apply Intruders** - The *Space assessment* (tab. 6.1) for *Avoidance Grid* is calculated with following threat modification:

$$softConstraints = \emptyset, hardConstraints = \emptyset$$

The *Find Best Path* (alg. 6.1) is applied, the resulting *avoidance path* is labeled as *Intruders Avoidance Path*.

**9<sup>th</sup> Apply Hard Constraints** - The *Space assessment* (tab. 6.1) for *Avoidance Grid* is calculated with following threat modification:

$$hardConstraints = \emptyset$$

The *Find Best Path* (alg. 6.1) is applied, the resulting *avoidance path* is labeled as *Hard Constraint Avoidance Path*.

**10<sup>th</sup> Apply Soft Constraints** - The *Space assessment* (tab. 6.1) for *Avoidance Grid* is calculated without any modification.

The *Find Best Path* (alg. 6.1) is applied, the resulting *avoidance path* is labeled as *Soft Constraints Avoidance Path*.

*Note.* The 7<sup>th</sup> to 10<sup>th</sup> steps are code-optimized for efficient calculation.

**11<sup>th</sup> 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. 6.1) procedure. The *Navigation Reach Set Approximation* can be constrained by *Rule Engine* (fig. ??).

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

*Note.* The *Waypoint Reachability* (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*.

**12<sup>th</sup> Load Movements** - the *Movement Buffer* is flushed for *future decision times*  $t_{i+1}, \dots, t_{i+k}$ . 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).

**13<sup>th</sup> 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. 6.5b) 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  $UAS.Position(t_{i+1})$ .

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

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

**Decision Frame:** The *mission control run* (fig. 6.7) describes overall process in *sequence*. The *orchestration overview* is given in (fig. 6.8).

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*.

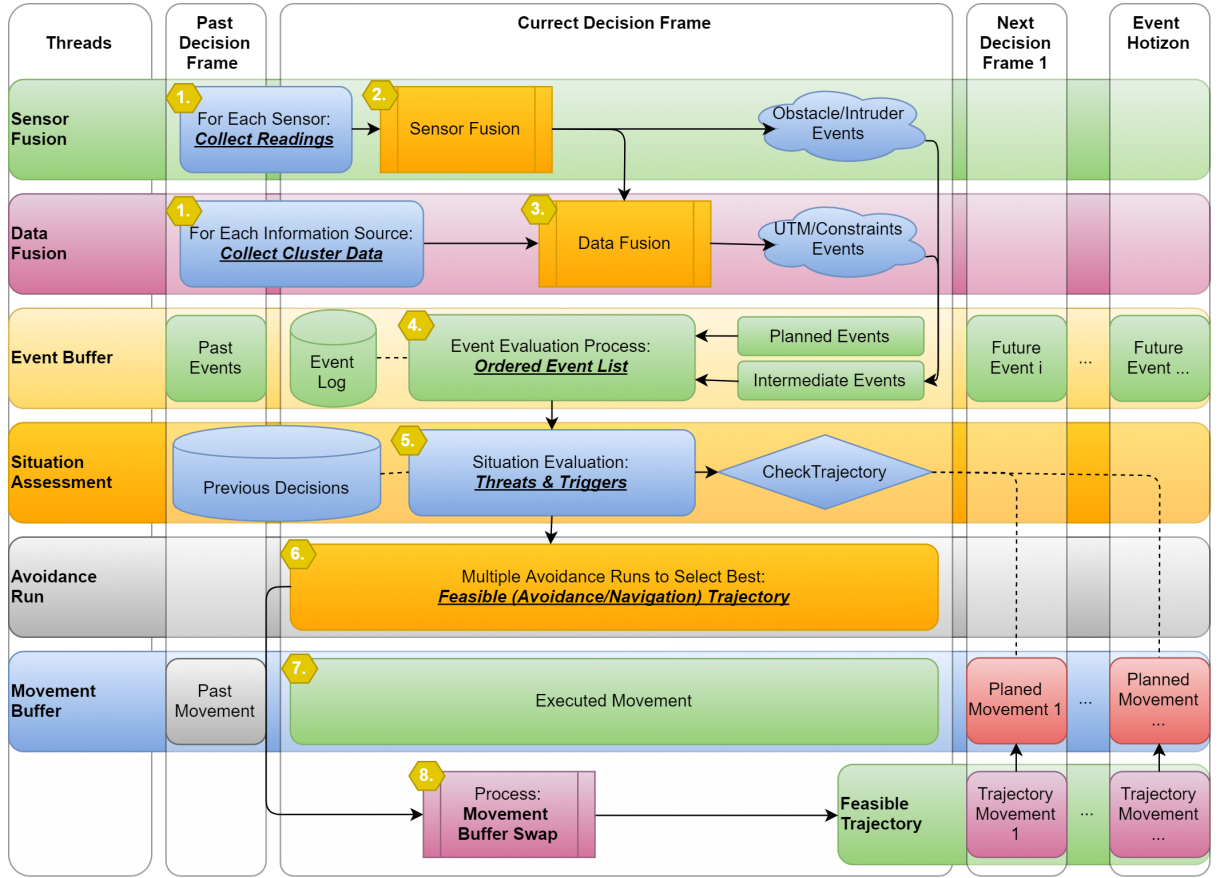


Figure 6.8: Mission control orchestration diagram.

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

The *implemented events* are listed in 5<sup>th</sup>-6<sup>th</sup> 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. ??).
  - 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<sup>th</sup>-11<sup>th</sup> step of *mission control run*.
  6. *Movement Buffer* - represents *movement automaton implementation* (sec. ??). 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. ??).
  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. 6.7). The numbered steps in (fig. 6.8) 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. ??, ??).
3. *Data fusion* - the information sources are preprocessed according to (sec. ??, ??).
4. *Event evaluation process* - the events are evaluated, if there is any triggering event (5<sup>th</sup>-6<sup>th</sup> mission control run steps) the situation evaluation process is called.
5. *Situation evaluation process* - the situation is evaluated according to 5<sup>th</sup>-6<sup>th</sup> mission control run steps.
6. *Feasible trajectory selection process* - from collected *navigation/avoidance trajectories* (7<sup>th</sup>-10<sup>th</sup> 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 trough modules to assess notable calculations/algorithms complexity on high abstraction level.

**Navigation Loop:** I the navigation loop, the *waypoint reach condition* (eq. 6.21) is checked, this is unitary operation with worst complexity  $\mathcal{O}(1)$ . The selection process of the next *Goal Waypoint* can get trough all waypoints in the mission if they are all unreachable the complexity is  $\mathcal{O}(|waypoints|)$ .

The *notable steps* complexity is following:

$$\begin{aligned} \text{Reach Condition:} & \quad \mathcal{O}(1) \\ \text{Select Next Waypoint:} & \quad \mathcal{O}(|waypoints|) \end{aligned}$$

**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:

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

*Note.* The *real-time clustering* is hard non-polynomial problem [6]. 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<sup>th</sup> to 10<sup>th</sup> 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:

$$\text{Insert Obstacles: } \mathcal{O}(|cells|)$$

The *intruders intersection model* type impact the insertion complexity. The *linear intersection* (sec. ??) is going through maximum of *layers count* cells.

The *body volume intersection model* (sec. ??) 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. ??) 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:

$$\text{Insert Intruders: } \mathcal{O} \left( \sum \left[ \begin{array}{l} |linearIntersections| \times |layers| \\ |bodyvolumeIntersections| \times |cells| \\ |cells|^{horizontalSpread \times verticalSpread} \end{array} \right] \right)$$

*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.

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

For *Hard/Soft Constraints* The algorithm used for intersection polygons was selected based on study [7], the selected algorithm *Shamos-Hoey* [8]. 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<sup>th</sup> to 10<sup>th</sup> step. All ratings (tab. 6.1) expect *Reachability*( $cell_{ij,k}$ ) and *Reachability*(*Trajectory*) are calculated. The *calculation complexity* boundary for one *reachability rating* is  $\mathcal{O}(1)$ . (eq. 6.7, 6.8). The *Recalculate Reachability* operation applied  $4\times$  have maximal *complexity* boundary given as follow:

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

Each time at the end of in 7<sup>th</sup> to 10<sup>th</sup> 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. 6.1) iterates over all *cells* in avoidance grid and over all *trajectories* passing trough that cell. The complexity boundary for *path selection* is given as follow:

$$\text{Select Path: } \mathcal{O}\left(4 \times \left(|cells| + \frac{|nodes|}{|cells|}\right)\right)$$

**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* [9].

### 6.7.5 Safety Margin Calculation

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

$$\text{maximalBodyRadius} \leq \text{safetyMargin} \leq 2 \times \text{turningRadius} \quad (6.24)$$

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* [10].

**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):

$$\text{safetyMarginBloat} = \begin{pmatrix} \text{intruderVelocity} \times \dots \\ \text{intruderDecisionFrame} \end{pmatrix} [m, ms^{-1}, s] \quad (6.25)$$



# Bibliography

- [1] ICAO. 4444: Procedures for air navigation services. Technical report, ICAO, 2018.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. A microeconomic view of data mining. *Data mining and knowledge discovery*, 2(4):311–324, 1998.
- [7] Jon Louis Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, (9):643–647, 1979.
- [8] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *17th annual symposium on foundations of computer science*, pages 208–215. IEEE, 1976.
- [9] 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.
- [10] 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.