6.6 Avoidance Concept

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

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

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

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

6.6.1 Avoidance Grid Run

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

Note. The *rating calculation* is outlined in (sec. ??). Low-cost sensor fusion example usable to feed our data fusion procedure is given in [1]. Semi-optimal concatenation trajectory search like ours can be found in [2].

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

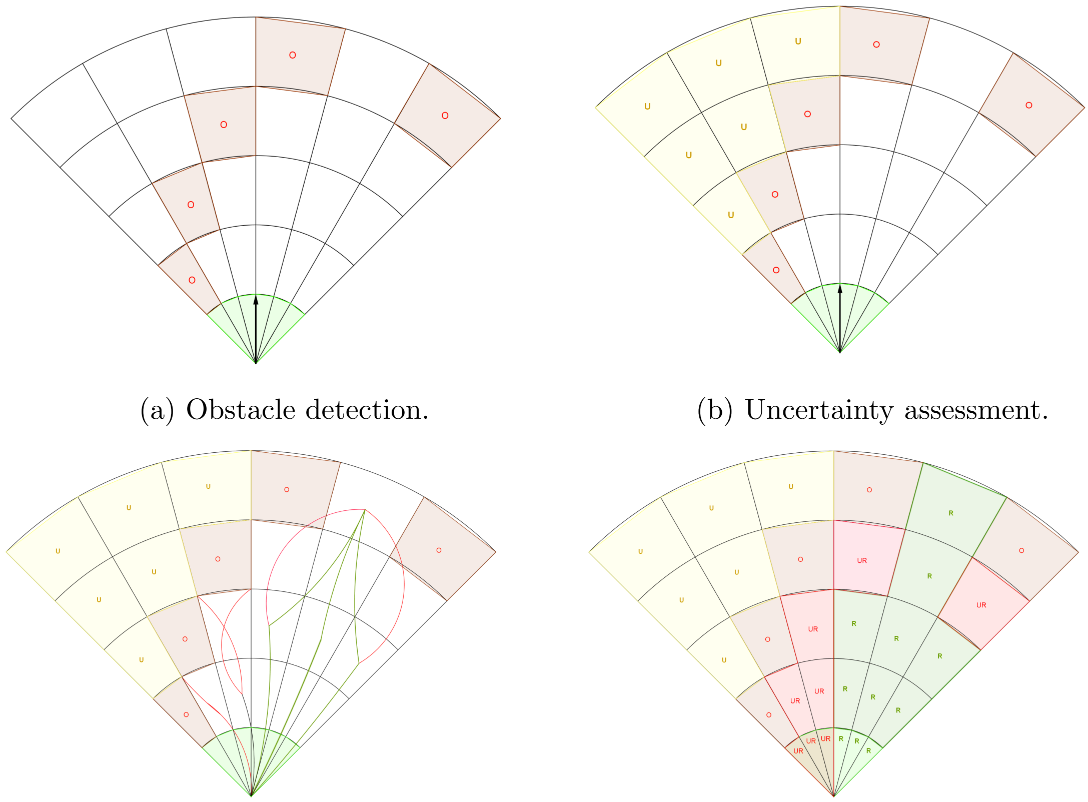
Space Assessment Principle: The *Avoidance Grid* is fed through *Data Fusion* (sec. ??). The process of *rating assessment* (tab. ??) is given in (fig. 6.1):

1. *Obstacle detection* (fig. 6.1a) - assessment of *detected obstacles* (eq. ??). The red (O) *cells* have Detected obstacle set as *true*. The other threats: *map obstacles* (eq. ??), *intruders* (eq. ??), *constraints* (eq. ??) are false. The red (0) *cells* are representing *Occupied*(t_i) (eq. ??) space in *Avoidance Grid* at decision time t_i.

2. *Uncertainty assessment* (fig. 6.1b) - the uncertain cells are cells which status cannot be *assessed*. The *Visibility* (eq. ??) is low. The *Uncertain* cells (yellow (U) mark) are equal to *Uncertain*(t_i) (eq. ??) in *Avoidance Grid* in *decision time* t_i. The *Constrained*(t_i) (eq. ??) space is equal to $\varnothing$ in this example.

3. *Trajectory reachability evaluation* (fig. 6.1c) - the *Reach Set* given as *Trajectory Set* (eq. ??). is then projected through *Avoidance Grid* and pruned according to (def. ??). *Reachable Trajectories* (eq. ??) are only those contained in $Free(t_i)$ space (eq. ??). 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. **??**). The *Reachable cells* are those which *contains* at least one *Reachable Trajectory Segment*.



(a) Obstacle detection.

(b) Uncertainty assessment.

(c) Trajectories reachability evaluation.

(d) Cell reachibality evaluation.

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

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

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

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

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

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

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

Algorithm 6.1: Find best *Path* in *Avoidance Grid*

```
Input  : Cell[] reachable (eq. ??), Waypoint goal, AvoidanceGrid(t_i) grid
Output: Trajectory avoidancePath, Error message
# Initialization & Reachability test;
avoidancePath = ∅;
if reachable == ∅ then
|  return [avoidancePath,"No path available, empty Reach Set"]
end
avoidanceCell = GetRandomCell(reachable);
# Look for for goal cell;
if goal ∈ grid then
|  # Goal is inside Avoidance Grid, Check if reachable;
|  avoidanceCell = grid.selectCellXYZ(goal);
|  if avoidanceCell.Reachable != true then
|  |  return [avoidancePath,"Waypoint not Reachable"]
|  end
else
|  # Goal is outside Avoidance Grid, look for closest reachable cell_{i,j,k};
|  minimalDistance = distance(avoidanceCell,goal);
|  for cell_{i,j,k} ∈ reachable do
|  |  if distance(cell_{i,j,k},goal) < minimalDistance then
|  |  |  if isOuterCell(cell_{i,j,k}) then
|  |  |  |  minimalDistance = distance(cell_{i,j,k},goal);
|  |  |  |  avoidanceCell = cell_{i,j,k};
|  |  |  end
|  |  end
|  end
end
# Reachable cell was found, Look for cheapest reachable trajectory;
avoidancePath = GetRandomTrajectory(avoidanceCell);
for trajectory ∈ avoidance Cell && trajectory.Reachable == true do
|  if trajectory.Cost < avoidancePath.cost then
|  |  avoidancePath = trajectory;
|  end
end
message = ∅;
return [avoidancePath,message]
```

The *Algorithm* (alg. 6.1) is based on *shortest path* search. Navigation is trying to reach *goal waypoint*; therefore it tries to shorten the distance between *final trajectory cell* and *goal waypoint*. If there is *reachable space* two situations can occur:

1. *Goal waypoint is inside the Avoidance Grid* - the *avoidance cell* is $\text{cell}_{i,j,k}$ containing *goal waypoint* if reachable.

2. *Goal waypoint is outside the Avoidance Grid* - the *avoidance cell* is the closest cell considered as an *outer cell* to *goal waypoint*.

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

Note. Outer cell is a $\text{cell}_{i,j,k}$ which has at least one *wall* directly neighbouring with *outer space* $(Universe - KnownWorld(t_i))$. The *outer cell* is selected to prevent navigation to the *trap*.

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

The *UAS* (blue plane) is following a *mission plan* in open space. Then there is a detection of a *collision situation* (fig. 6.2). The *Obstacle* is detected in the *top-right* Avoidance Grid corner.

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

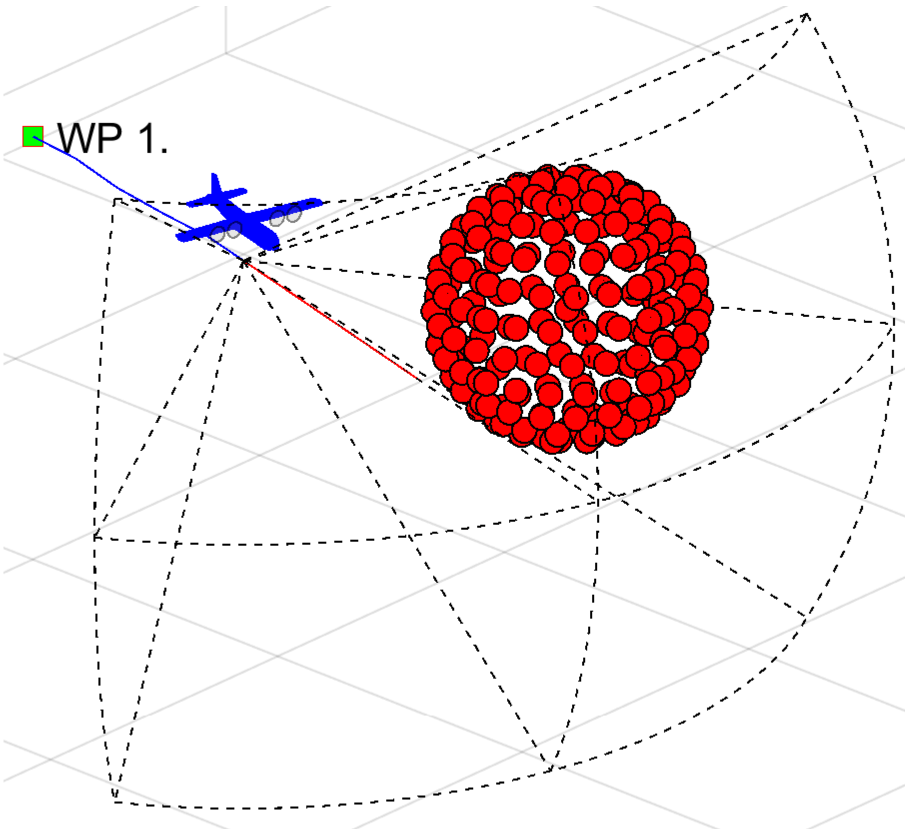


Figure 6.2: Example: The situation to be evaluated by *Avoidance Run*.

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

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

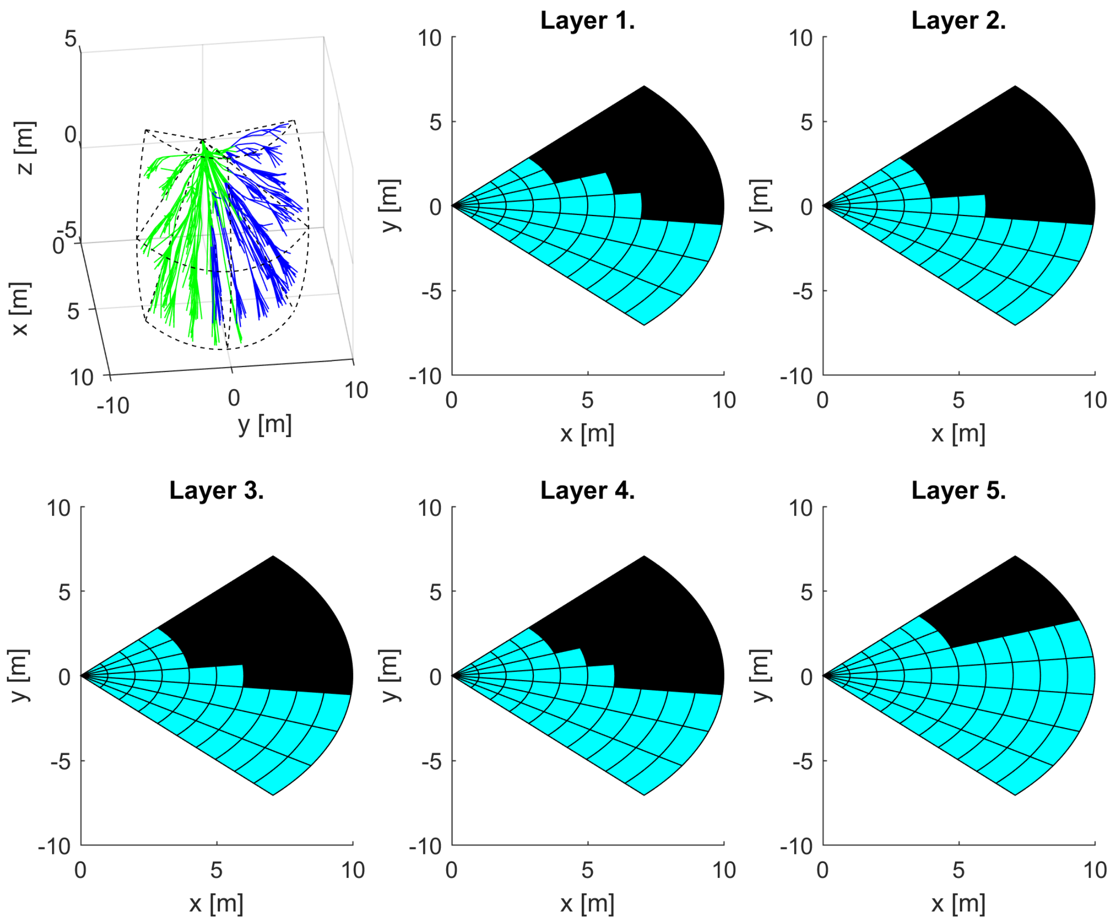


Figure 6.3: Example: The *Visibility* evaluation by *Avoidance Run*.

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 a similar matter, depending on the count of *reachable trajectories* passing through them:

1. *Unreachable Cells* (red fill) - there is no trajectory through *free space* or the *cell* is not in *free space*.

2. *Reachable cells* (green fill) - there is at least one *feasible trajectory* reaching *free cell*.

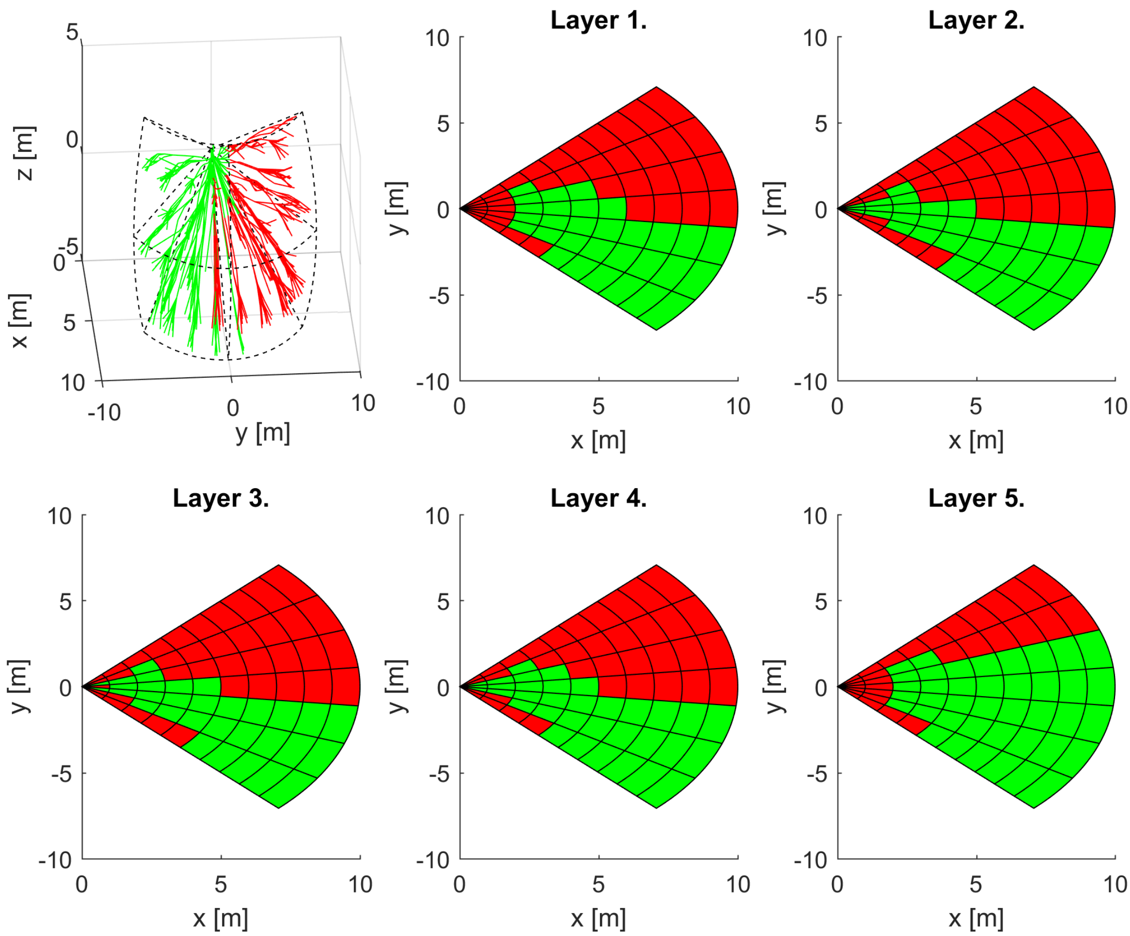


Figure 6.4: Example: The *Reachability* evaluation by *Avoidance Run*.

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

6.6.2 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 a *sub-system* for long term *trajectory tracking*. The *Avoidance Grid Run* (sec. 6.6.1) is solving the *Path Search* problem inside operation space constrained by *Avoidance Grid* for time t_i.

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

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

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

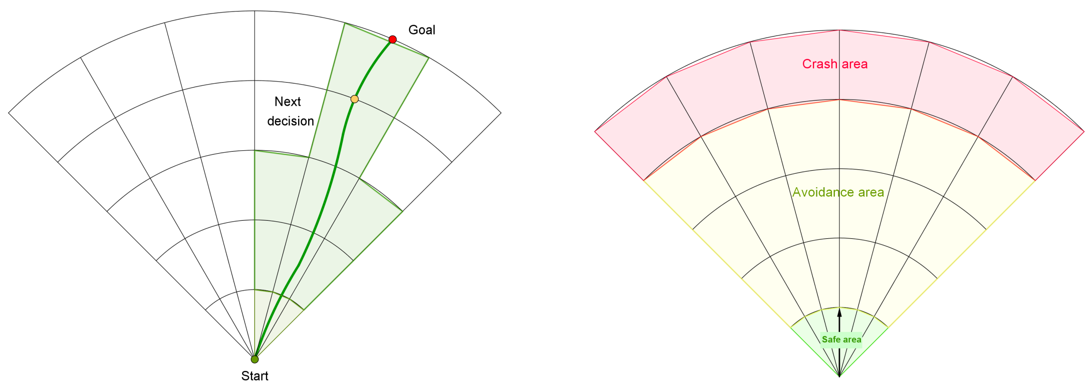
2. *Next Decision* - the point when the next *Avoidance Grid Run* is applied. The outline of events and triggers is required. The *decision* will be made in the *next decision time* t_{i+1}.

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

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).

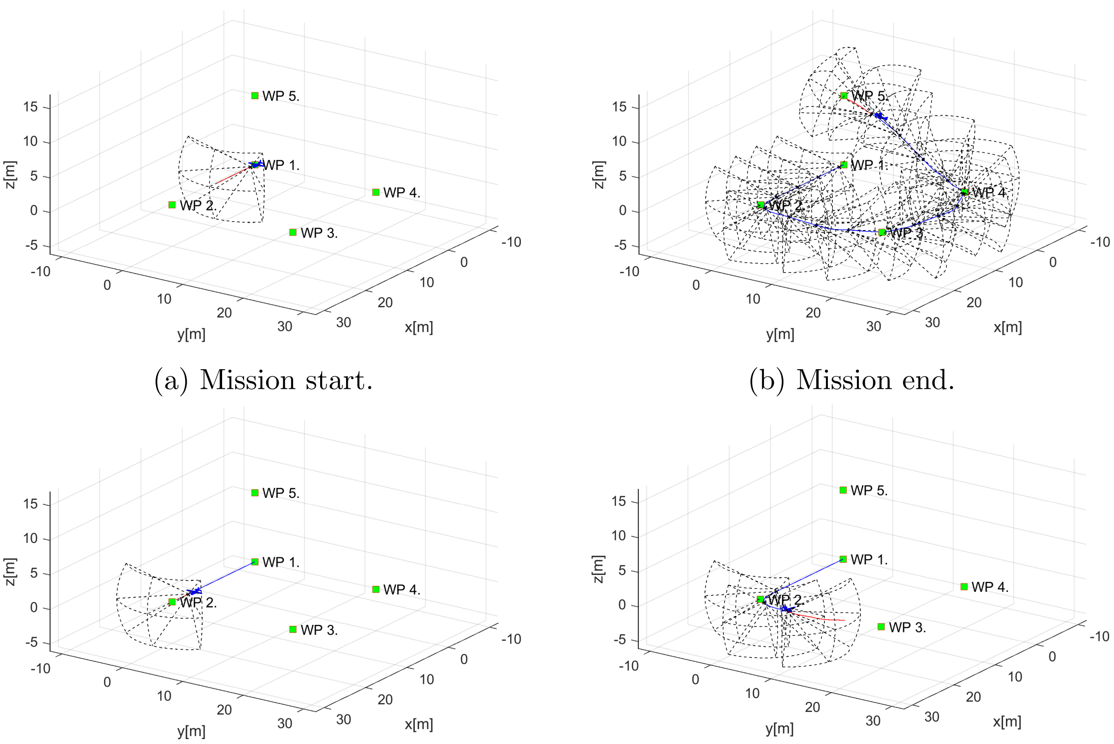
Joining *Avoidance Grid Runs* (fig. 6.6) example portrays *Avoidance Grid Runs* invoked on various *Decision Points* to achieve *Navigation* functionality. The UAS (blue plane) is flying Mission (green numbered waypoints). The *Avoidance Grid* boundary (black dashed line) for each *Decision Point* (UAS position at time t_i). Following the 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 the *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* to achieve Navigation.

General Concept: [2] The *General Concept* is taken from [3, 4], consisting of following main modules:

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

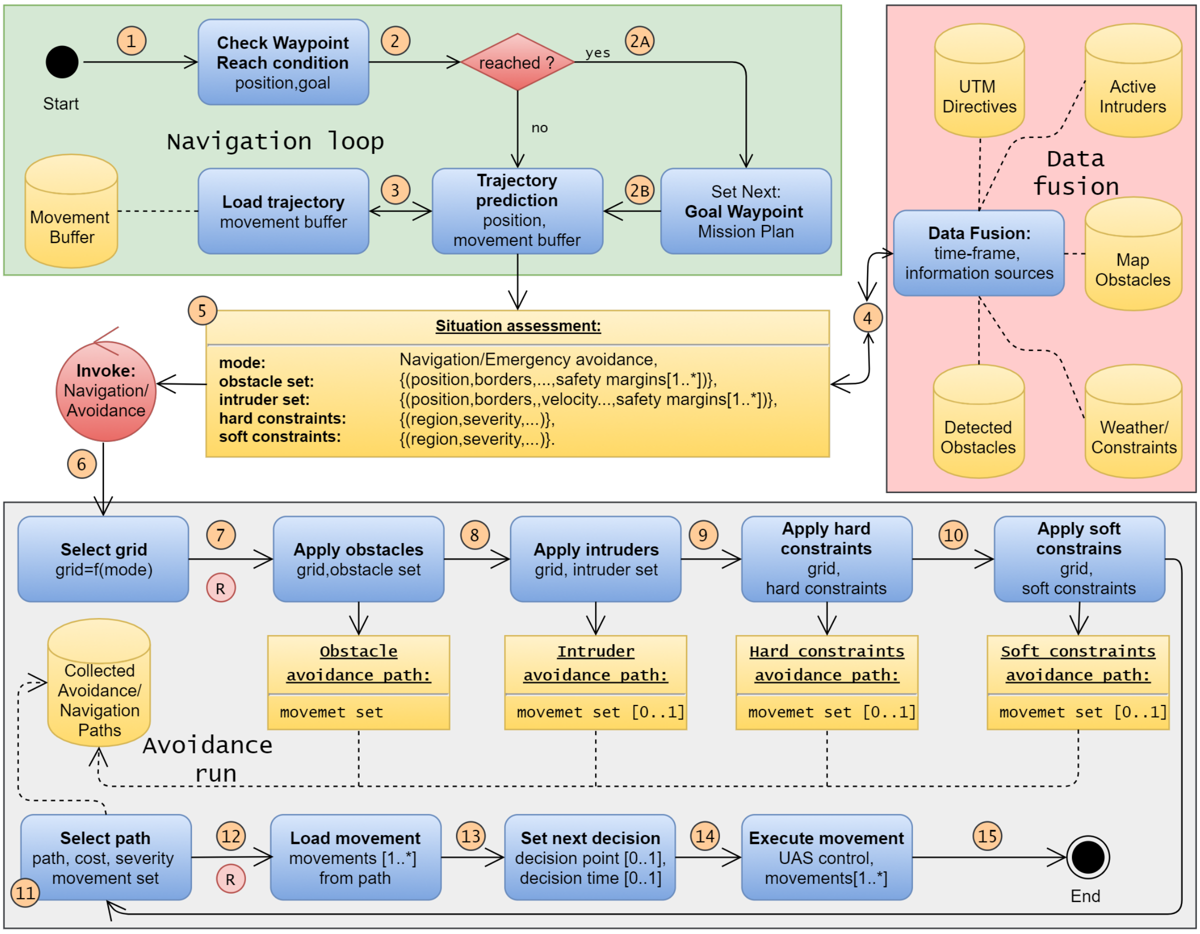


Figure 6.7: Mission control run activity diagram.

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; This is responsible for prioritizing threat avoidance according to a type.

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

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

1. *Navigation Mode* - the *UAS* is navigating through *Airspace* following *cost-effective patterns* and obeying *Airspace Authority* (UTM). The *Navigation Grid* is an instance of *Avoidance Grid* (sec. **??**) with initialized *Navigation Reach Set* (ex. *Turn-Minimizing 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. *Coverage-Maximizing 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 portioning pattern; therefore the *Data Fusion* (sec. **??**) 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/velocity second* (tab. **??**, **??**).

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

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

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

$$Sensors = \{LiDAR, ADS - B\} \tag{6.1}$$

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:

$$DataFusion = \{RatingBasedDataFusion \quad (sec. \textbf{??})\} \tag{6.2}$$

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

$$MovementBuffer = \{\} \tag{6.3}$$

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:

$$\begin{aligned} AvoidanceGrid(0) &= \{UAS.position(0), AvoidanceReachSet(UAS.ReachSet)\} \\ NavigationGrid(0) &= \{UAS.position(0), NavigationReachSet(UAS.ReachSet)\} \\ OperationMode &= Navigation \end{aligned} \tag{6.4}$$

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

$$\begin{aligned} Mission &= \{Waypoint_1 \ldots Waypoint_n\} \\ GoalWaypoint &= Mission.waypoint_1 \\ LastWaypoint &= Mission.waypoint_n \end{aligned} \tag{6.5}$$

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

$$obstacles = \{\}, intruders = \{\}, hardConstraints = \{\}, softConstraints = \{\} \tag{6.6}$$

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

First, start with the definition of *waypoint reach condition* (def. 1) and *Unreachable waypoint* (def. 2).

Definition 1. *Waypoint Reach Condition for* current *decision time* t_i *for* UAS *position and current* Goal Waypoint *is satisfied only if:*

$$\begin{aligned} distance(UAS.position(t_i), GoalWaypoint(t_i)) \\ \leq \\ 2 \times \max\{length(movement) : \forall movement \in MovementSet\} \end{aligned} \quad (6.7)$$

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

Definition 2. *Unreachable Waypoint. The* Goal Waypoint *evaluates as unreachable in decision time* t_i *when* Avoidance Grid Run *(alg. 6.1) cannot find the* navigation/avoidance path *leading to it.*
Formally: The Avoidance/Navigation Grid *has range defined as* final layer distance*. When the* Goal Waypoint *is in* range *of* Grid*:*

$$Grid(t_i).range \geq distance(UAS.position(t_i), GoalWaypoint(t_i)) \quad (6.8)$$

and following condition is satisfied:

$$\begin{aligned} \forall cell_{i,j,k} \in Grid(t_i)\ \nexists cell_{i,j,k}.Reachable == true \wedge \dots \\ \dots \wedge distance(cell_{i,j,k}, GoalWaypoint(t_i)) \leq \dots \\ \dots \leq 2 \times \max\{length(movement) : \forall movement \in MovementSet\} \end{aligned} \quad (6.9)$$

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 a sequence of the following steps:

1st **Check Waypoint Reach Condition** - the *UAS position* for given a *time frame* t_i is checked under condition (eq. 6.7). If the condition is met continue with 2nd step otherwise continue with 3rd step.

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

$$GoalWaypoint == LastWaypoint$$

Set next goal waypoint like follow:

$$GoalWaypoint = Mission.getNextWaypoint()$$

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

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

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

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

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

Note. Avoidance Grid's Data fusion (sec. **??**) is run in the 7th- 10th step (fig. 6.7).

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

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

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

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

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

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

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

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

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

Invoke Navigation/Avoidance based on Situation Assessment (5th-6th step): The *deciding events* depending on *Trajectory Prediction* (3^{rd} step) and *Data Fusion* (4^{th} step) (fig. 6.7) are the following:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7th Apply Obstacles - The *Space assessment* (tab. **??**) 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.*

8th Apply Intruders - The *Space assessment* (tab. **??**) 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.*

9th **Apply Hard Constraints** - The *Space assessment* (tab. **??**) 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*.

10th **Apply Soft Constraints** - The *Space assessment* (tab. **??**) 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 7th to 10th steps are code-optimized for efficient calculation.

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

The *Navigation Path* for *Navigation Mode* is selected by a standard *Find Best Path* (alg. 6.1) 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 the following priority:

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

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

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

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

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

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

1. Navigation Mode (no active collision cases) - *Decision Point* is set as the point before *UAS* enters into *Crash Zone* (fig. 6.5b) 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 the *next movement execution*. Current decision point $UAS.Position(t_i)$, next decision point $UAS.Position(t_{i+1})$.

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

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

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

The key idea is to explain what happens in one *decision frame*. The *mission control run* is implemented as multi-thread application which sends the signals between threads. Each thread is the semi-independent process with forced synchronization on *decision frame switch*.

The notable threads and their roles & responsibilities are summarized like follow:

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

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

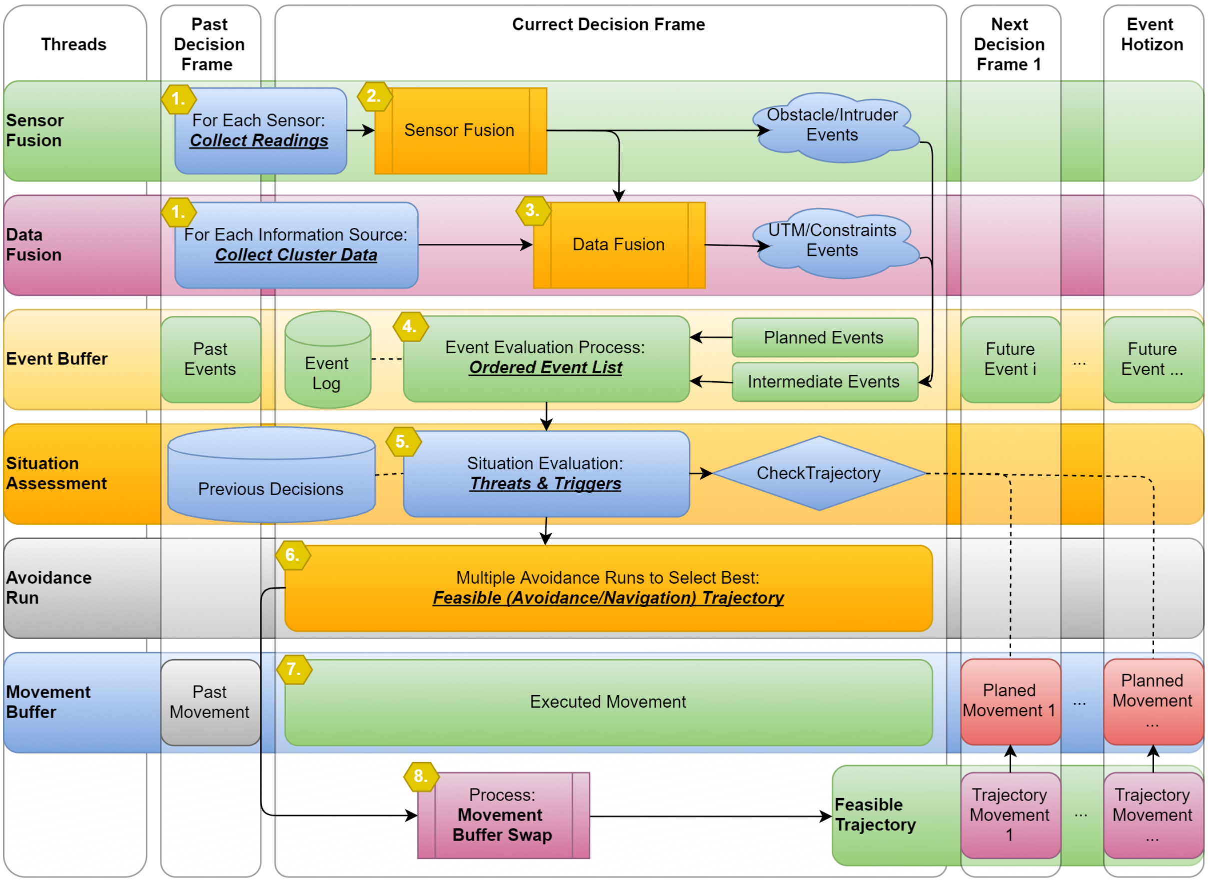


Figure 6.8: Mission control orchestration diagram.

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

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

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

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

   The event buffer thread executes following event-related activities:

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

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

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

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

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

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

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

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

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

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

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

The *roles & responsibilities* of each thread have been explained to outline their orchestration and roles in *mission control run* (fig. 6.7). The numbered steps in (fig. 6.8) shows the threads orchestration in the following manner:

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

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

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

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^{th}-6^{th} mission control run steps) the situation evaluation process is called.

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

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

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

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

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

6.6.3 Computation Complexity

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

Navigation Loop: On the navigation loop, the *waypoint reach condition* (eq. 6.7) is checked, this is a unitary operation with worst complexity $\mathcal{O}(1)$. The selection process of the next *Goal Waypoint* can get through all waypoints in the mission if they are all unreachable the complexity is $\mathcal{O}(|waypoints|)$.

The *notable steps* complexity is following:

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

Data Fusion: The *data fusion* is all about *threat selection.*

If *UAS* is in *controlled airspace*, it needs to iterate over received *collision Cases* to select *active ones.* The complexity of this step is linear; therefore boundary is given as $\mathcal{O}(|collisionCases|)$.

Thresholding *Detected Obstacles* is done by simple comparison of *LiDAR ray hits* in given $cell_{i,j,k}$ of *Avoidance Grid.*

Any loading of *threats* from *information sources* depends on clustering. The *Airspace Clustering* is considered as static for our setup. Therefore the *count of active airspace clusters* has the main impact on complexity. The *count of information sources* is static and not changing over mission time. Information sources usually implement *Hash search function* with complexity $\mathcal{O}\ln|searchedItemSet|$.
The *computation complexity* boundaries for *Data fusion* in our setup are following:

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

Note. The *real-time clustering* is a *hard non-polynomial problem* [5]. Usually, all information sources and sensor have *polynomial complexity* of processing. The *controlled airspace clusters* are usually set for a very long period. Therefore *Obstacle Map*, *Airspace Constraints*, and, *Weather Constraints* can be considered as preprocessed

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

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

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

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

The *threat insertion* is realized in 7^{th} to the 10^{th} step. The first is *Avoidance Grid* filled with *Static Obstacles*. The *Avoidance Grid* is designed to separate rotary *LiDAR* ray space into hit count even cells. Insertion of *LiDAR* scan into *Avoidance Grid* complexity depends on *total cell count*. The *upper boundary* for *insert obstacles* is given like follow:

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

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

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

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

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

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

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

For *Hard/Soft Constraints* The algorithm used for intersection polygons was selected based on a study [6], the selected algorithm *Shamos-Hoey* [7]. The *calculation complexity* boundary is given like follow:

$$\texttt{Hard Constraints Intersection:} \quad \mathcal{O}(|cells| \times |hardConstraints| \times \max |constraintPoints|^2)$$

$$\texttt{Soft Constraints Intersection:} \quad \mathcal{O}(|cells| \times |softConstraints| \times \max |constraintPoints|^2)$$

Each *threat* category application in *Mission Control Run* is done after *each intersection* in 7^{th} to the 10^{th} step. All ratings (tab. **??**) expect $Reachability(cell_{ij,k})$ and $Reachability(Trajectory)$ are calculated. The *calculation complexity* boundary for one *reachability rating* is $\mathcal{O}(1)$. (eq. **??**, **??**). The *Recalculate Reachability* operation applied $4\times$ have maximal *complexity* boundary given as follow:

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

Each time at the end of in 7^{th} to the 10^{th} step the *Avoidance Path is Selected*. The *Worst Case* (expected) scenario is to *select* four paths for each *threat* application. The algorithm for *best path selection* (alg. 6.1) iterates overall *cells* in avoidance grid and over all *trajectories* passing through that cell. The complexity boundary for *path selection* is given as follow:

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

Conclusion: Overall approach complexity is *low*. If proper *Information Sources* with efficient clustering and *intersection models for intruders* are used, the approach will stay within *non-polynomial complexity*. The average load time for *testing scenarios* is summarized in (tab. **??**).

Note. The calculation of *Reach Set* is eliminated by pre-calculation for *state range* [8].

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

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

[8] 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.