6.9 (R) Rule Engine

This section is follow up of *UTM functionality definition* (sec. **??**), outlining realization of *UTM directives* on *UAS* side (sec. 6.9.1, 6.9.2).

Reasoning: The *Avoidance* process and *UTM directives fulfillment* is different in every national airspace. The ICAO issues recommendation [1, 2] which are implemented by every member country, some of procedures are stricter some are implemented differently.

The *UTM* collision case calculation and procedures may be universal, but their realization by *UAS* will be heavily impacted by local legislation and procedures. The *approach* must account the need of *variable parts* of *obstacle avoidance process*. The *dynamic parts* needs to be woven to hard-coded processes.

Note. Please refer to *Template Programming* and *Aspect Oriented Programming* for further explanation.

Inspiration: There was a *Maritime Rules* implementation [3] in form of *Movement Restrictions* and *Waypoint Changes*.

6.9.1 (R) Architecture

Purpose: The *core process* of *Avoidance Grid Run* (sec. **??**) and *Mission Control Run* (sec. **??**) needs to be enhanced based on situation. The architecture is based on *aspect oriented approach* [4]. The key ideas and concepts are taken from rule engine implementation for multiagent navigation system [5].

Rule Engine: The program module to inject and run *rules* modifying standard workflow based on triggering events. The *aspect oriented* approach enables to configure rules in *run-time* via predefined process hooks - *Decision Points*.

The rules in context of this work are pieces of code which have semi-static structure consisting from following parts (fig. 6.1):

1. *Decision Point* - hook point in process where the rule can be attached/detached. If more than one rule is hooked the priority of execution needs to be defined.

2. *Context* - the *run time context* in time of *invocation* in our case the *copy* of *Mission Control*, *Avoidance/Navigation Grid* and *Collision Cases*.

3. *Parser Method* - optional helper method to parse interesting data set from *Context*. The *parsed data* have better readability.

4. *Condition Check Method* - implementation of the trigger. If the sufficient condition is met, the rule body is applied.

5. *Rule application* - calculations and data structure changes. Mainly *disabling trajectories* in *Reach Set* in our implementation.

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

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

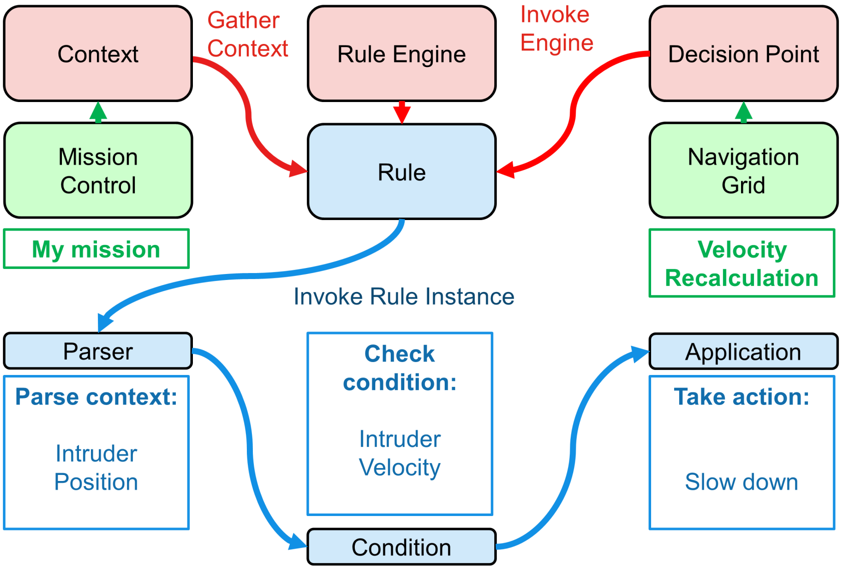


Figure 6.1: Rule engine components overview.

Example: The *UAS* is flying in controlled airspace. The *intruder* shows in front of *UAS*. The *UAS* is faster than *intruder*. The *UAS* tries to obtain permission for *Overtake*. The *UTM* does not allow *overtake*, because of *insufficient UAS maneuverability capability*. The *Rule* (fig. 6.1) with:

1. *Context* - UAS Mission Control, containing the actual mission goal and UAS IMU parameters.

2. *Decision Point* (Joint Point) - Navigation grid, containing projected constraints and reach set approximation.

3. *Rule is invoked:*

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

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

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

6.9.2 (R) Rule Implementation

Configuration: The *Rule Engine Architecture* (fig. 6.1) is configured to handle *UTM functionality* for *Collision Case Resolution* (sec. ??). The overview of *Context* (Green), *Decision Points* (red) and *Rules to be Invoked* (cyan) is given in (fig. 6.2).

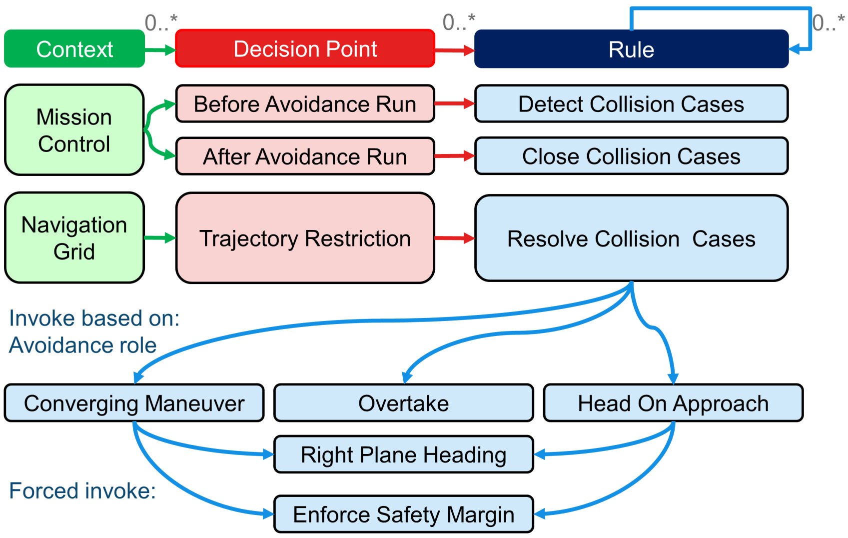


Figure 6.2: Rule engine initialization with Rules of the air.

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

Decision Points: The *Decisions* are bounded to *Mission Control Run Process* (fig. ??) in following manner:

1. *Before Avoidance Run* (before step 7.) - Context: *Mission Control* (Received Collision Cases) - the *UTM* can send directives. It is required to find which ones are impacting our *UAS*.

2. *Trajectory Restrictions* (after step 7.) - Context: *Navigation Grid* (Trajectory Restrictions) - adaptation of *behavior* imposed by *active collision cases*.

3. *After Avoidance Run* (after step 11.) - Context: *Mission Control* (Collision Case Resolutions) - our *UAS* will update the status of *Collision Cases* then it checks the *avoidance conditions*. The *Resolution Notification* resolution notifications are sent to UTM afterwards.

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

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

   a. *Detect Collision Cases* (sec. 6.9.3) - the detection of active participation in received *collision cases* and generation of *restrictions*.

   b. *Resolve Collision Cases* (sec. 6.9.4) - the enforcement of *active avoidance roles* in *collision cases*. The one *Restriction Rule* is invoked directly.

   c. *Close Collision Cases* (sec. 6.9.5) - impact calculation and *Resolution Notification* to *UTM* authority.

2. *Restriction Rules* - restricting the *Navigation Grid* trajectories or altering *goal waypoint* based on *selected collision cases*:

   a. *Converging Maneuver* (sec. 6.9.7) implementation of *Converging Avoidance* (sec. **??**).

   b. *Head On Approach* (sec. 6.9.6) implementation of *Virtual Roundabout Enforcement* (sec. **??**).

   c. *Overtake* (sec. 6.9.8) implementation of *overtake maneuver* for *Overtaking plane* (sec. **??**).

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

   a. *Right Plane Heading* (sec. 6.9.9) - restrict all trajectories heading to space separated by parametric plane in *Avoidance Grid* which are heading or belonging to plane.

   b. *Enforce Safety Margin* (sec. 6.9.10) - restrict all *Trajectories Segments* which are in proximity of *Collision Point* lesser than *Enforced Safety Margin*.

6.9.3 (R) Rule: Detect Collision Cases

This rule is activated each *UAS avoidance run*. *UTM* sent out all related *collision cases* (6.1) based on our *UAS identifier*. Creation of *collision case* is given in sec. **??** based on air traffic periodical*position notifications* (sec **??**).

$$UTM \times timeFrame \to UTMCollisionCases \tag{6.1}$$

If there are available *position notifications* (sec **??**) from surrounding air-traffic, UAS will calculate own *collision cases* (6.2).

$$uasStatus \times positionNotification \times utmTimeFrame \to UASCollisionCases \tag{6.2}$$

Then UAS merges *own collision cases* with *UTM collision cases*, if there exist following disparities UAS will take action:

1. $distamce(ownCollisionPoint, utmCollisionPoint) \geq threshold$, send UTM notificaiton, use $utmCollisionPoint$

2. $utmMargin \geq ownMargin$, use safety margin from UTM.

3. $utmAvoidanceRole == active, ownAvoidanceRole == inactive$, use UTM avoidance role.

4. $utmCollisionCase == active, ownCollisionCase == uncertain$, use UTM provided collision case, not all *position notifications* are available.

5. $utmCollisionCase == inactive, ownCollisionCase == active$, notify UTM with new collision case, ignore collision case until UTM approves.

Note. Avoidance role is classified as *inactive* if and only if UAS has *right of the way*, it is classified as *active* otherwise.

Safety margin determined by UTM has priority, because not all calculations factors are available for UAS.

Collision Case unknown to UTM are ignored, due to safety reasons (false data spoofing), collison case is activated after UTM confirmation. If there is real intruder not confirmed by UTM it is handled via *non-cooperative* or *emergency* avoidance procedure

Selection process of active *collision cases* is based on UAS *avoidance role* in each *collision case.*

- If the *avoidance roles* are following: *Head On Approach, Converging Maneuver, or Overtake* in all *collision cases* UAS system will stay in cooperative mode.

- If there exists at least one *collision case* with *own avoidance role* or *intruder avoidance role* set as *emergencing*, the UAS will notify UTM and ask for *diversion order*, meanwhile it sets itself into *Emergency avoidance* mode.

- If there exist multiple *Overtake avoidance roles* or combination of *Overtake avoidance role* and *Other active role*, the UAS will decrease its cruising speed like follows:

$$UASSpeed = \max \begin{Bmatrix} minimalUASCruisingSpeed, \\ \min\{intruderSpeed\} \quad \forall activeCollisionCases \end{Bmatrix} \tag{6.3}$$

  During *slow-down* UAS switchs to *emergency avoidance mode* and asks for *divergence order* from UTM.

Ordering of collision cases starts if and only if the *UAS* is in *cooperative avoidance mode.* The cases are ordered for processing based on severity rating which is calculated based on:

1. *Safety Margin* - the greater safety margins are prioritized.

2. *Intruder vehicle class* - the more dangerous intruders are prioritized.

3. *Collision point distance* - closer collision points are prioritized.

4. *UAS avoidance role* - *Head on Approach* is favored upon *Converging maneuver*, due to direct collision severity.

Rule engine invocation for each *active collision case* is then applied on *descending severity sorted* list.

The rule is summarized in table 6.1.

Invocation: Every *Decision point* in*UAS main loop*

Objective:

1. Fetch UTM *Collision cases* for given decision time frame.

2. Create/update own *own collision cases* based on received *Position notifications* from surrounding *Intruders.*

3. Merge *Collision cases* based on *UTM priority order.*

4. Select active *collision cases* based on following conditions:

   a. *Active participation* in *collision case* where *avoidance role* $\neq$ *Right of the way.*

   b. *Collision point* is int front of UAS.

   c. *Emergency mode detection* there exist at least one non-cooperative participant.

5. Order *collision cases* based on *severity.*

6. If there is at least one *active collision case* enforce rule *Resolve collision case* (tab 6.2) for each *active collision case.*

Context	*Condition*	*Application*
UAS Mission control, Before Avoidance Run, UTM/UAS collision cases	Clean *avoidance grid*, No emergency	Active collision case selection, Prioritization

Table 6.1: Detect collision cases rule definition.

6.9.4 (R) Rule: Resolve Collision Case

Active collision cases are processed one by one. All collision cases are applied to *Navigation grid. Navigation grid* contains all possible *trajectories* in form of *Reach set.* All *trajectories* are *reachable* at the beginning of UAS *avoidance frame.* Each application of *collision case resolution* rule disables some subset of feasible *trajectories.* For this reason are *active collision cases* sorted by severity.

It is assumed that UAS is in *cooperative avoidance mode.* If previous application of this rule forced UAS into *emergency mode* the rule is not applied to save system resources. *Emergency* mode is invoked if *rule application* disable all *trajectories* in *Navigation grid.* If there is at least one *feasible trajectory* in *avoidance grid* follow-up rule is invoked based on UAS *avoidance role.*

The rule is summarized in table 6.2.

Invocation: This rule is invoked if exists at least one *active collision case* in given *navigation grid time-frame*, moreover *avoidance grid* must be empty and *cooperative avoidance mode* is enforced. *Objective:* Based on *active collision case* and *UTM directives* enforce behaviour based on *own avoidance role*: 1. *Head on approach* - rule 6.4. 2. *Converging maneuver* - rule 6.5. 3. *Overtake* - rule 6.6. 4. *Emergency mode* - switch from *active avoidance mode* to *emergency mode.*

Context	*Condition*	*Application*
UAS mission control, Trajectory restriction, Collision cases,	Active merged collision case, Resolution mandate from UTM	Enforce Rules of Air or Enforce emergency

Table 6.2: Resolve collision case rule definition.

6.9.5 (R) Rule: Close Collision Cases

Collection of rule results detected by rule 6.1 and *resolved* by rule 6.2 is done via *context of rule engine.* For each *time-frame* and each $trajectory \in NavigationGrid$, there exists rule engine *context* query (6.4) which returns *trajectory status* and *list of applied rules on trajectory.*

$$Context(trajectory, timeFrame) \rightarrow \{State : Enabled/Disabled, Rule(s)\} \tag{6.4}$$

Calculation of possible trajectories in *navigation grid* is using *collected rule results* (6.4). If the *trajectory state* and linked *rule reason* is sufficient, the *trajectory* is disabled for given *time frame. Standard navigation algorithm* (TBD - reference to section with outer navigaiton loop) is used to select *feasible trajectory.*

Rules of the air and their applicaiton in *General Aviation* cases is consistent. Increasing trafic density can impose new layers of rules, which may cause the *soft deadlock* in *manuverability.* In this case *Navigation grid* will have all *possible trajectories* exhausted. Following procedure is executed:

1. UAS switch into *Non-cooperative avoidance mode* or *Emergency avoidance mode* depending on situation severity (One conflict can be handled with *vertical separation* of conflicting aircrafts).

2. UAS broadcasts *warning message* to all nearby aircrafts, and *separation message(s)* to conflicting aircraft. *Separation message* contains *expected collision point* and *preferred separation type.*Each conflicting aircraft then reacts and sends *action notification* to UTM.

3. If UAS switchs into *emergency mode*, non cooperative avoidance using *avoidance grid* is induced. Each relevant intruder is projected as *timed body volume intruder* (TBD reference to intruder probabilistic model), where *safety margin* is used as *body radius.*

UAS notifies *UTM* with *course change*, *planned avoidance trajectory*, *avoidance mode. UTM* approves planned changes or sends *plan corrections* (out of scope). The rule summary is given in table 6.3

Invocation: There exists at least one *active collision case* which had impact on *Navigation grid.*

Objective: Ensure that multiple *avoidance rules* application gives feasible *avoidance strategy*, enter into *emergency avoidance mode* otherwise. Following steps are executed:

1. *Collect rules* applied on *navigation grid* from *active collision cases.*
2. *Calculate possible trajectories for avoidance*, there may be none.
3. If there is no *feasible route*, for each *intruder* from related *collision cases*:
   a. Issue *warning message* containing *expected collision point* and *preferred separation type.*
   b. Create appropriate *intruder object* for *avoidance grid.*
   c. Calculate *evasive maneuver* based on expected *separation type.*
4. *Notify UTM* with *collision case resolution* for each *active collision case. Nofigy UTM* with *planned trajectory* and *avoidance mode*

Context	*Condition*	*Application*
UAS Mission control, After avoidance run, Collision resolutions	At least one trajectory in Navigation grid, Emergency check	Force *Emergency mode* OR Close Collision Case

Table 6.3: Close collision case rule definition.

6.9.6 (R) Rule: Head on Approach

Rule (6.4) is invoked based on *angle of approach* range condition, defined *collision case* section **??**. The handling of *head on* avoidance is given in section **??**.

Virtual round-abound for UAS and intruder is created by UTM. The center of virtual round-abound and *corrections for participants margins* are determined based on:

1. *Collision case center* - contributes to round-abound center median point.
2. *UAS and intruder maneuverability* - determines *attendants avoidance mode* and *maximal avoidance margins.*
3. *Surrounding air-traffic* - contributes to round-abound center median point, determines ideal *ideal avoidance margins* due to *wake turbulence* prevention.

Virtual round-abound center is calculated as *corrected median* (6.5) taking *cluster of collision cases* and calculates median of their collision points corrected by *weather* and *wake turbulence* factor.

$$
\begin{aligned}
correctedMedian = &\sum_{c_i \in collisionCases} \left(c_i.center + correction\right)/count(collisionCases) + \\
&+ corrrection(Weather) + correction(WakeTurbulence)
\end{aligned}
\tag{6.5}
$$

Corrected margin needs to be calculated for each *participating aircraft*, because of the *virtual roundabout* center correction (6.5). Each *round-abound participant* is ordered based on importance (lowest maneuverability first). Then for each *round-abound participant* obtains *corrected margin* (6.6) calculated from *collision case safety margin*, corrections based on other *more important vehicles*, *weather*, *wake turbulence*.

$$
correctedMargin = \min \begin{bmatrix} caseMargin + correction \begin{pmatrix} ImportantVehicles, \\ Weather, \\ WakeTurbulence \end{pmatrix} \\ maximalAvoidanceMargin \end{bmatrix} \tag{6.6}
$$

Invocation: When *UAS avoidance role* is *Head on* avoidance and *avoidance grid* is empty.

Objective: Ensure that *UAS body* does not enter into *intruder's well clear zone.*

1. Prevent *left-side leading* maneuvers (rule 6.7).
2. Prevent head on *safety margin* breach(rule 6.8).
3. Return to original course, when *navigation grid* is clear.
4. Prevent *wake turbulence* (by safety margin correction).
5. Enforce *Round-about* behaviour (by clustering collision cases).

Context	*Condition*	*Application*
UAS Navigation Grid, Collision Point, Avoidance role	None	Run rules referenced in objective listing.

Table 6.4: Head on Approach rule definition.

6.9.7 (R) Rule: Converging Maneuver

Rule is invoked based on *angle of approach* range defined in *collision case calculation* (sec. **??**). Behaviour enforced to this rule is equal to rule 6.4 except the *intruder* stays on his original path. UAS behaviour is described in section **??**. The *rule summari* is given by table 6.5.

Invocation: When *UAS avoidance role* is *Converging* and *avoidance grid* is empty. *Objective:* Ensure that *UAS body* does not enter into *intruder's well clear zone.* 1. Prevent *left-side leading* maneuvers (rule 6.7). 2. Prevent head on *safety margin* breach(rule 6.8). 3. Return to original course, when *navigation grid* is clear. 4. Prevent *wake turbulence* encounter (by safety margin correction).

Context	*Condition*	*Application*
UAS Navigation grid, Collision point, Avoidance role	None	Run rules from objective.

Table 6.5: Converging maneuver rule definition.

6.9.8 (R) Rule: Overtake

During overtake maneuver there is our *UAS* and *Intruder* cruising at same *flight level. Angle of approach* (α) is lesser than 70°. *UAS* absolute velocity is much greater than *overtaken* absolute velocity.

It is assumed that during *overtake* maneuver *overtaken* intruder will keep constant heading and velocity. If this assumption is broken *UAS* system will invoke *Emergency avoidance* procedure. *UTM* will calculate such *divergence* and *convergence* waypoints that *overtake safety condition* (6.7) is satisfied.

$$distance(uasPosition, overtakenPosition) \geq utmMargin, \forall t \in manueverTime \quad (6.7)$$

Where *utmMargin* is calculated based on *Collision case* resolution. *Main idea*is to calculate *Safe offset for Overtake maneuver*, lets have:

$$velocityDifference = \|uasVelocity - overtakenVelocity\| \quad [ms^{-1}, ms^{-1}, ms^{-1}] \quad (6.8)$$

Decision distance (6.9) is given as distance when *UTM mandate* takes effectiveness, its assumed that *UTM* knows *utm decision frame* [s]:

$$decisionDistance = velocityDifference \times uasDecisionFrame \quad [m, ms^{-1}, s] \quad (6.9)$$

Overtake middle distance(6.10) is length of hypotenuse for triangle where *positional difference* and *utm margin* for overtake are cathetuses:

$$overtakeMiddle = \sqrt{\begin{array}{l}\|uasPosition - collisionPoint\|_2 + \\ + safetyMargin^2\end{array}} \quad [m, \vec{m}, \vec{m}, m] \tag{6.10}$$

Safe offset (6.11) is considered as combination of *overtake middle distance* (6.10), *decision distance* and uas *waypoint reach margin.*

$$safeOffset = \begin{array}{c} overtakeMiddle \\ +decisionDistance \\ +waypointReachMargin \end{array} \quad [m, m, m, m] \tag{6.11}$$

Note. Waypoint reach margin [m] is property of own *UAS navigation algorithm.* It represents maximal distance of vehicle position and waypoint at time when waypoint is considered reached.

Local coordinate frame: UAS and Overtaken are in Local coordinate frame heading in X^+ axis direction (X^+ front of aircrafts, X^- back of vehicles, Y^- right side, Y^+ left side, $flightLevel \rightarrow Z = 0$), Collision Point is considered as $\vec{0}$,

Divergence point (6.12) in local coordinates is given as right offset of *(*UTM margin) and *decision distance*:

$$divergence = \begin{bmatrix} 0 \\ -decisionDistance - utmMargin \\ 0 \end{bmatrix} \quad [\vec{m}, m, m] \tag{6.12}$$

Convergence point (6.13) in local coordinates is given frontal *safe offset* (6.11) and right offset of *UTM margin* and *decision distance*:

$$convergence = \begin{bmatrix} safeOffset \\ -decisionDistance - utmMargin \\ 0 \end{bmatrix} \quad [\vec{m}, m, m] \tag{6.13}$$

Convergence (6.14) and *Divergence* (6.15) waypoint in global coordinate frame is obtained via transformation function R_{XYZ} as follow:

$$\begin{aligned} divergenceWaypoint =& collisionPoint \\ &+ R_{XYZ}(overtakenOrientation, divergence) \end{aligned} \tag{6.14}$$

$$\begin{aligned} convergenceWaypoint =& collisionPoint \\ &+ R_{XYZ}(overtakenOrientation, convergence) \end{aligned} \tag{6.15}$$

Overtake rule is summarized in table 6.6.

Invocation: Invoked by rule *Collision Case Resolution* (rule 6.2) *Divergence Waypoint* (6.14): waypoint to diverge from original UAS path to ensure Intruder safety, with unchanged intruder velocity and heading. *Convergence Waypoint* (6.15): waypoint when convergence to original UAS path is enabled, within unchanged intruder velocity and heading. *Objective:* 1. Calculate *Divergence Waypoint* and *Convergence Waypoint*. 2. Enforce Divergence/Convergence waypoint during avoidance.

Context	*Condition*	*Application*
UAS Navigation Grid, Collision Point, Avoidance Role	$UASVelocity$ $>>$ $IntruderVelocity$	Calculate & Enforce: • Divergence waypoint, •Convergence waypoint

Table 6.6: Overtake rule definition.

6.9.9 (R) Rule: Right Plane Heading

There is need to check if *trajectory* is heading to *right side* from *collision point.* For this purpose we need to define *separation plane in 3D environment. Separation plane* will be defined according to Samuelson *hyperplane separation theorem* [6].

Separation plane (6.16) is defined by three points in *global coordination frame*:

1. *UAS Position* which is fixed to given *time-frame.*
2. *Collision point* which is not equal to *uas position* by definition.
3. *Gravitational acceleration* vector fitted to *UAS position* and orthogonal to vector $(uasPosition \rightarrow collisionPoint)$.

The properties of these three points guarantees that $scale.usasPosition \neq scale.collisionPoint \neq scale.gravitationalAcceleration$ for any linear $scale \neq 0$.

$$SeparationPlane = Plane \begin{pmatrix} uasPosition, collisionPoint, \\ loc2glob(uasPosition, gravitationalAcceleration) \end{pmatrix} \tag{6.16}$$

Separation plane (6.16) in *right-hand coordinate frame* where $center = uasPosition$ X^+ is given by vector $\vec{x^+}$ $(uasPosition,\ collisionPoint)$ and Z^- is given by vector $\vec{z^-}$ $(uasPosition,\ gravitationalAcceleration)$. Then *right subspace* can be defined as all points where $y \leq 0$ and *left subspace* as all points where $y > 0$.

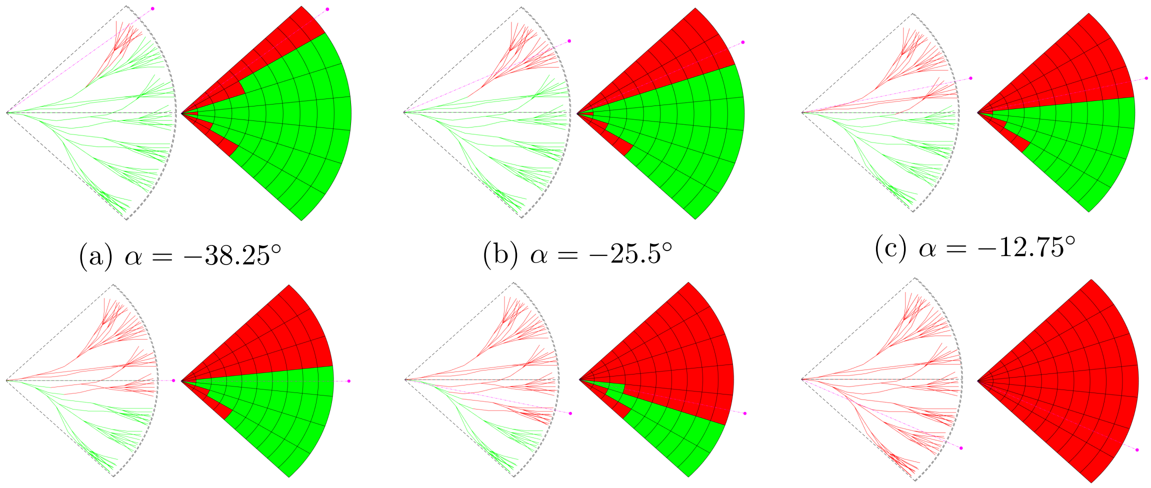
Reach set contains *trajectories*, minimal dataset for trajectory is time-series of *position* and *heading* regardless *underlying nonlinear model.* Let us have *transformation function* which can map UAS *position* and *heading* into *separation plane coordinate frame.*

The *first condition* (6.17) says that each trajectory *point* must lie within *right subspace.*

$$\forall position \in trajectory, \quad position \in rightSubspace \tag{6.17}$$

The *second condition* (6.18) needs to be applied for each *decision point*, when *trajectory* can be re-planned. It must be ensured that in time of reaching *decision point* vehicle is not heading into *left subspace* with given *turning time horizon.* The *minimal information* contains heading (velocity) vector. Checking if linear projection from *position* point with *heading* in given time-frame $[0, horizon]$ is sufficient.

$$\forall t \in [0, horizon], \quad (position + velocity * t) \in rightSubspace \tag{6.18}$$



(a) $\alpha = -38.25°$ (b) $\alpha = -25.5°$ (c) $\alpha = -12.75°$

(d) $\alpha = 0°$ (e) $\alpha = 12,75°$ (f) $\alpha = 25.5°$

Figure 6.3: Right plane heading rule evaluation for various angles of approach α.

Figure 6.3. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left subfigure). These trajectories are divided according to *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. Situation is shown for various *collision point angles of approach* α.

Rule for right plane heading check is summarized in table 6.7.

Invocation: Invoked by other *maneuver rules.*

Objective: Disable all *trajectories* in *Navigation grid's reach set* which are:

1. *Heading into collision zone*
2. *Leading into collision zone*

Context	*Condition*	*Application*
UAS Navigation Grid, Collision point (LOC)	There are feasible trajectories in Navigation Grid.	Disable trajectories in Navigation Grid.

Table 6.7: Right plane heading rule definition.

6.9.10 (R) Rule: Enforce safety margin

Rule 6.7. checks right plane heading for single mass point along *trajectiories*. Rule needs to account *body mass* of *intruder* and UAS, other factors like safe distance, regulations etc. All mentioned factors are included in *safety margin*. *Safety margin* is applied as *radius ball* around *collision point*.

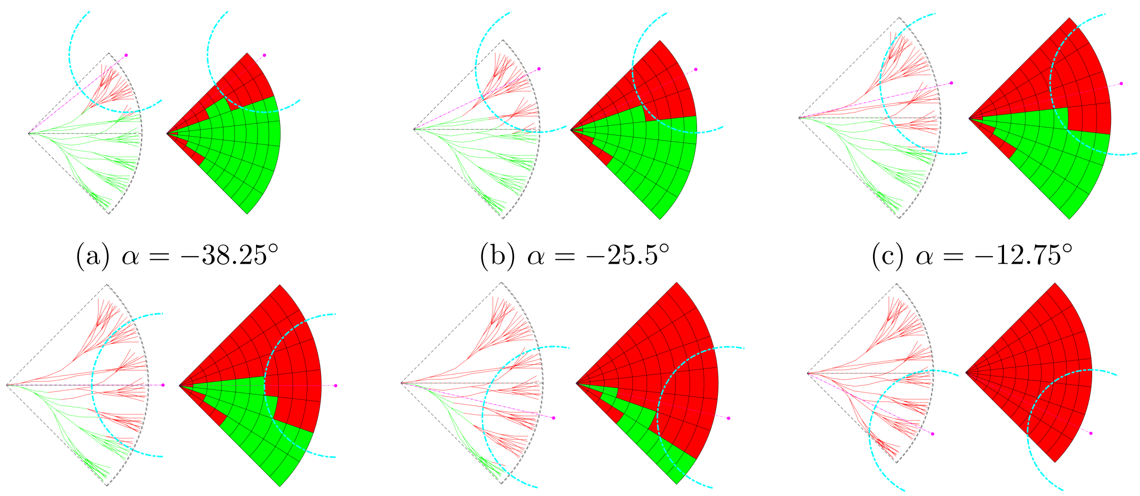
Collision point can be mapped from *global coordinate frame* to *reach set coordinate frame*, based on UAS *position and orientation* in *decision time*. Then comparison of distance between *collision point* and every *trajectory decision point* is trivial.

Trajectory feasibility condition for *non-controlled airspace* (6.19) is given as follow:

$$\forall position \in trajectory, \quad distance(position, collisionpoint) \geq safetyMargin \tag{6.19}$$

Controlled airspace must maintain *well clear condition*. To enforce protective barrel around *collision point* one must compare *global coordinates*. *Trajectory feasibility condition* for *controlled airspace* (6.20) is given as follow:

$$\begin{aligned} \forall position \in & trajectory, \\ & XYdistance(position, collisionPoint) \geq safetyMargin \\ & flightLevelStart \geq Z(position) \geq flightLevelEnd \end{aligned} \tag{6.20}$$



(a) $\alpha = -38.25°$ (b) $\alpha = -25.5°$ (c) $\alpha = -12.75°$

(d) $\alpha = 0°$ (e) $\alpha = 12,75°$ (f) $\alpha = 25.5°$

Figure 6.4: Enforce safety margin rule evaluation for various angles of approach α.

Figure 6.4. shows *enabled* (green line) and *disabled* (red lines) *trajectories* (left sub-figure). These trajectories are divided according to *separation line* (magenta dashed line), given by *vehicle position* and *collision point* (magenta circle). More trajectories are disabled due to *safety margin* (teal dashed line) around the *collision point*. *Space segmentation* (right subfigure) show *reachable* (green fill) *unreachable* (red fill) space. Situation is shown for various *collision point angles of approach* α.

Rule for safety margin check is summarized in table 6.8.

Invocation: Invoked by other *maneuver rules.* *Objective:* Based on type of airspace, for given *collision point* and *safety margin* disable trajectories in: 1. Ball radius for *non-controlled* airspace (6.19). 2. Well clear barrel *controlled* airspace (6.20).

Context	*Condition*	*Application*
UAS Navigation Grid Collision point Safety Margin	There are feasible trajectories for condition application.	Disable trajectories in Navigation Grid.

Table 6.8: Enforce safety margin rule definition.

Bibliography

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

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

[3] Michael R Benjamin, Joseph A Curcio, John J Leonard, and Paul M Newman. Navigation of unmanned marine vehicles in accordance with the rules of the road. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3581–3587. IEEE, 2006.

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

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

[6] Hans Samelson, Robert M Thrall, and Oscar Wesler. A partition theorem for euclidean n-space. *Proceedings of the American Mathematical Society*, 9(5):805–807, 1958.