6.5 Situation Representation in the Avoidance Grid

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

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

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

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

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

6.5.1 Obstacles

Introduction: The *static obstacles* were used in original concept [1], the *Avoidance Grid* and *Movement Automaton* were repurposed to enable *finite time deterministic* avoidance. An *Constraint based path search* and *obstacle modeling* is summarized in [2].

This section is handling basic problems of *static obstacle* detection and its focused on following real-world fixed position threats:

1. *Static Obstacles* - detected by LiDAR sensor or fused from *Obstacle Map* information source.

2. *Geo-fencing Areas* - defined by offline/online information source as permanent flight restriction zones. There is usually no physical obstacle. The space is considered as *hard/soft constraint.*

3. *Long-term bad weather Areas* - the *weather* is changing often (hour period), there are *weather events* which lasts for *hours* or *days.*

Changing Scanning Density of LiDAR: A LiDAR sensor is scanning in conic section given by $distanceRange$, $horizontalRange$, $verticalRange$, where distance range is in interval $[0, maxDistance]$, horizontal offset range is in $[-pi, pi]$, and vertical offset range is in $[\varphi_s, \varphi_e]$.

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

The surface of area given by some distance d, and unitary offsets$\partial horizontal^{\circ}$, d$vertical^{\circ}$ is changing with *distance*. The minimal triggering area of object surface is not changing. This fact has an impact on count of the hits on object surface.

The example is given in (fig. 6.1) where we have two identical objects (red circle) in distances 5 and 10 meters. The closer object consumes 5 LiDAR beam hits and the farther object consumes only 3 LiDAR beam hits. The probability of obstacle encounter is remaining the same for closer and farther object. The *detected obstacle rate* assessment should return the same detected obstacle collision rate for objects with same scanned surface (with different LiDAR ray hit count).

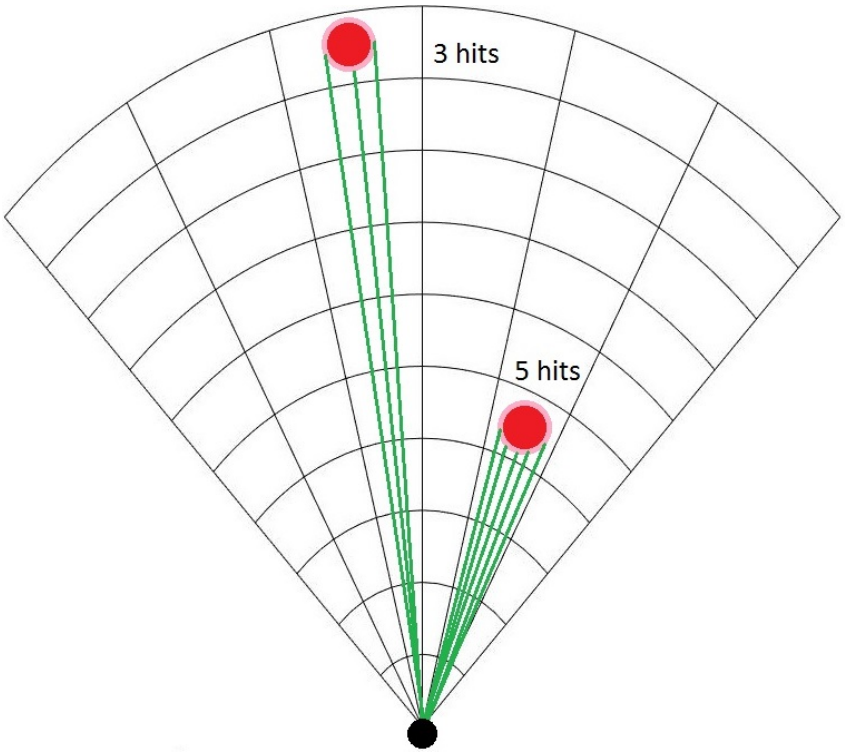


Figure 6.1: Different count of LiDAR hits with different distance from UAS.

Map and Detected Obstacles Fusion: The concept of *offline/online obstacle map* is mandatory in modern obstacle avoidance systems and increases the safety of navigation/avoidance path. The *older* concept was considering only LiDAR reading or *real-time sensor readings* in general [1].

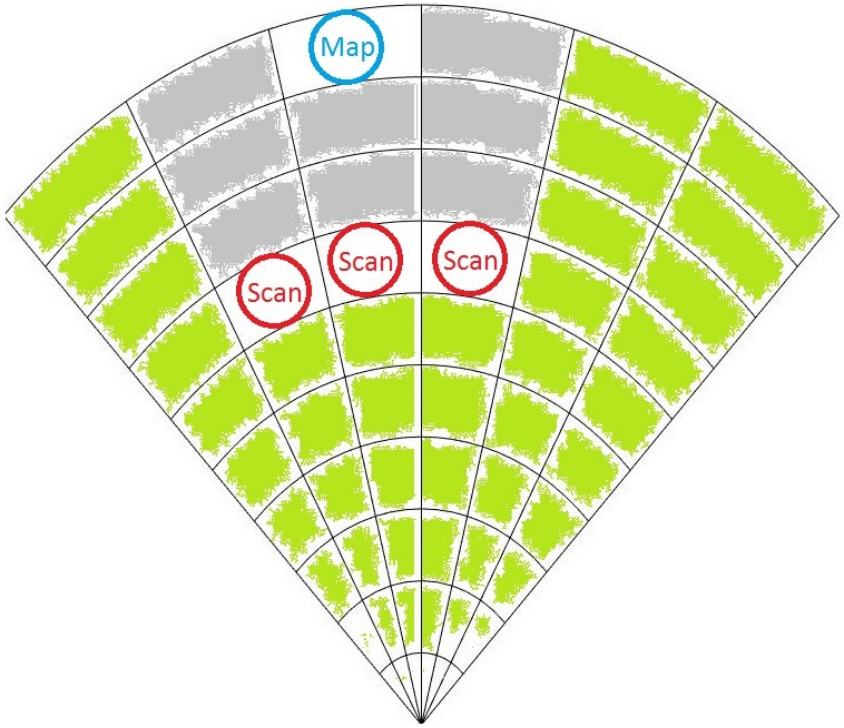


Figure 6.2: Overshadowed map obstacle by detected obstacles.

The fusion of real time sensor readings and obstacle map (prior knowledge) is required. Data fusion of these two sources is strongly depending on visibility property, because there are three basic scenarios:

1. *Dual detection* - the obstacle is marked on the map and detected by sensory system at some point of the time (older concept works).

2. *Hindered vision* - the detected obstacles are hindering vision to map obstacle therefore map obstacle uncertainty arises (older concept fails).

3. *False-positive map* - map obstacle occupied space is visible by sensory system, but negative detection is returned. Therefore the map is giving *false-positive* information.

The second case is given in fig. 6.2, where map obstacle (blue circle) is overshadowed by three scanned obstacles (red circle). The visible space is denoted by green fill, the invisible space is denoted by gray fill.

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

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

merged information (eq. 6.43). The *sensor field* can contain *multiple static obstacle sensors*.

Detected Obstacle Rate for LiDAR: Lets have only one sensor set as homogeneous two axis rotary LiDAR. For one $cell_{i,j,k}$ there exists set of passing LiDAR beams:

$$lidarRays(cell_{i,j,k}) = \left\{ \begin{bmatrix} horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \tag{6.1}$$

The horizontal and vertical offset of LiDAR ray is homogeneous. Meaning the horizontal/vertical distances between each two neighbouring LiDAR beams are equal.

The set $lidarRays(cell_{i,j,k}))$ (eq. 6.1) is finite countable and nonempty for any $c_{i,j,k}$, otherwise it will contradict the definition of avoidance grid (def. **??**).
The hit function $lidarScan()$ returns a distance of single beam return for beam with dislocation $[horizontal^\circ, vertical^\circ] \in lidarRays(cell_{i,j,k})$ angle offsets. The set of LiDAR hits (eq. 6.2) in cell $cell_{i,j,k}$ is defined like follow:

$$lidarHits(cell_{i,j,k}) = \left\{ \begin{bmatrix} distance = lidarScan(), \\ horizontal^\circ \in horizontalOffsets, \\ vertical^\circ \in verticalOffsets \end{bmatrix} \in \mathbb{R}^2 : \begin{array}{r} distance \in cell_{i,j,k}.distanceRange, \\ horizontal^\circ \in cell_{i,j,k}.horizontalRange, \\ vertical^\circ \in cell_{i,j,k}.verticalRange \end{array} \right\} \tag{6.2}$$

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

$$obstacle^{LiDAR}_{cell_{i,j,k}} = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \tag{6.3}$$

Note. The *naive obstacle rate* (eq. 6.3) ignores that *LiDAR rays* are getting more far apart from each other. The *cell surface* is increasing with cell distance from *UAS*.

The hindrance (eq. 6.4) rate is naturally defined as supplement to naive obstacle rate. This definition is sufficient, because its reflecting the *remaining sensing capability* of LiDAR.

$$hindrance^{LiDAR}_{cell_{i,j,k}} = 1 - \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \tag{6.4}$$

Cell Density Function: Let's start with differential form of cell surface (eq. **??**). The target object have several hits in *Avoidance Grid*. Target $cell_{i,j,k}$ has following properties which are used in surface calculation:

1. *Horizontal span* - defines range of horizontal scanner partition.
2. *Vertical span* - defines range of vertical scanner partition.

By rewriting (eq. **??**) and using horizontal range parameter and inverted vertical range parameter following surface integral is obtained (eq. 6.5).

$$Area(cell_{i,j,k}) = \int_{horizontal^\circ_{start}}^{horizontal^\circ_{end}} \int_{vertical^\circ_{end}}^{vertical^\circ_{start}} radius^2 \cos(vertical^\circ) \quad \mathrm{d}vertical^\circ \mathrm{d}horizontal^\circ \tag{6.5}$$

Note. The *radius* parameter is *average* distance of hits landed in $cell_{i,j,k}$. This helps to reflect real *scanned surface*.

Numerically stable integration exist for boundaries $horizontal^\circ\ in[-\pi, \pi]$, $vertical^\circ \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ given as follow:

$$Area(radius, horizontalRange, vertical^\circ_{start}, vertical^\circ_{end}) = \ldots$$
$$= \begin{cases} vertical^\circ_{start} < 0, vertical^\circ_{end} \leq 0 : \\ \qquad radius^2(\sin|vertical^\circ_{start}| - \sin|vertical^\circ_{end}|) \times horizontalRange) \\ vertical^\circ_{start} < 0, vertical^\circ_{end} > 0 : \\ \qquad r^2(\sin|vertical^\circ_{start}| + \sin|vertical^\circ_{end}|) \times horizontalRange) \\ vertical^\circ_{start} \geq 0 vertical^\circ_{end} < 0 : \\ \qquad r^2(\sin vertical^\circ_{end} - \sin vertical^\circ_{start}) \times horizontalRange) \end{cases} \tag{6.6}$$

An intersection surface for cell is defined in (eq. 6.6). Area covered by LiDAR hits (eq. 6.7) is defined as LiDAR hit rate (hits to passing rays ratio) multiplied by *Average* cell intersection surface (eq. 6.6).

$$lidarHitArea(cell_{i,j,k}) = \frac{lidarHits(cell_{i,j,k})}{lidarRays(cell_{i,j,k})} \times Area \begin{pmatrix} radius, horizontalRange, \\ vertical^\circ_{start}, vertical^\circ_{end} \end{pmatrix} \tag{6.7}$$

There is user defined parameter for *LiDAR treshold area*, which represents minimal considerable surface area for obstacle to be threat. The *detected obstacle rate* considering surface is defined in (eq. 6.8) and it removes bias of naive approach (eq.6.3).

$$obstacle(LiDAR, cell_{i,j,k}) = \min\left\{\frac{lidarHitArea(cell_{i,j,k})}{UAS.lidarThresholdArea}, 1\right\} \tag{6.8}$$

Visibility Rate for LiDAR: For each $cell_{i,j,k}$ and each sensor in sensor field there exist hindrance rate, which defines how much vision is clouded in single cell. Example of hindrance calculation for LiDAR has been given by (eq. 6.4). Let us consider cell row $cellRow(j_{fix},\ k_{fix})$ with fixed horizontal index j_fix and vertical index k_{fix} is given as series of cells (eq. 6.9).

$$cellRow(j_{fix}, k_{fix}) = \left\{ cell_{i,j,k} \in AvoidanceGrid : \begin{array}{l} i \in \{1, .., layersCount\}, \\ j = j_{fix}, k = k_{fix} \end{array} \right\} \tag{6.9}$$

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

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

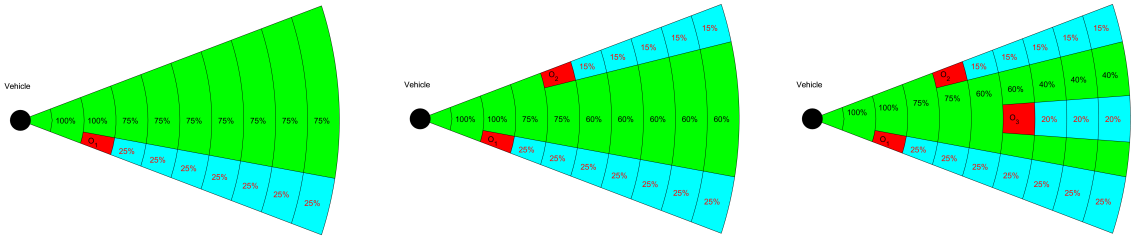
and for one selected $cell_{i,j,k}$ the visibility rate is naturally defined as a supplement to hindrance from previous cells. The visibility is defined in (eq. 6.10).

$$\begin{aligned} visibility(cell_{i_c,j_c,k_c}) &= \ldots \\ \cdots &= 1 - \sum_{index \in \mathbb{N}^+}^{index<i_c} hindrance(cell_{a,j_c,k_c} : cell_{a,j_c,k_c} \in cellRow(j_c, k_c) \end{aligned} \tag{6.10}$$

Example: Let be $cell_{4,j_{fix},k_{fix}}$ is selected for visibility rate assessment, then $cell_{1,j_{fix},k_{fix}}$, $cell_{2,j_{fix},k_{fix}}$, and $cell_{3,j_{fix},k_{fix}}$, are used as a base of cumulative hindrance rate.
The cumulative hindrance rate for any $cellRow(j_{fix}, k_{fix})$ is bounded:

$$0 \leq \sum_{cell \in cellRow(j_{fix},k_{fix})} visibility(cell) \leq 1 \tag{6.11}$$

Note. A *cumulative hindrance rate* does not always reach 1 in case of LiDAR sensor, because some rays may pass or hit after leaving *avoidance grid* range.



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

Figure 6.3: Obstacle hindrance impact on visibility in *Avoidance Grid Slice*.

For one cell row $cellRow(j_{fix}, k_{fix})$, where count of layers is equal to 10, and layers have equal spacing. There is LiDAR sensor

During consequent LiDAR scans $s(t_0)$, $s(t_1)$, $s(t_2)$, and $s(t_3)$ the obstacle sets $\mathcal{O}_1(t_1) = \{o_1\}$, $\mathcal{O}_2(t_2) = \{o_1, o_2\}$, and $\mathcal{O}_3(t_3) = \{o_1, o_2, o_3\}$ are discovered. Assigned hindrance rates are like follow:

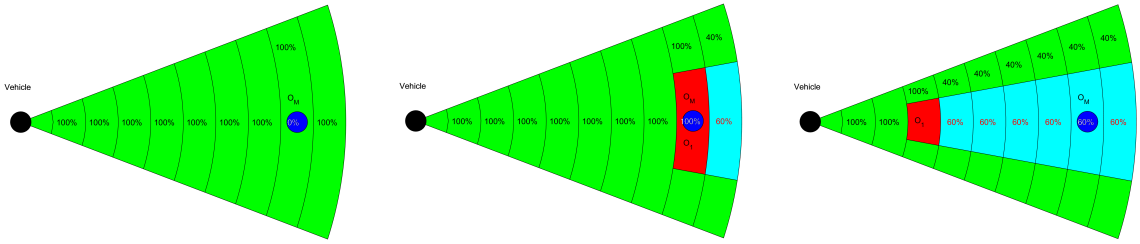
1. *Time* t_0 - there is no obstacle nor hindrance, all cells are fully visible.

2. *Time* t_1 (fig. 6.3a) - $\mathcal{O}_1(t_1) = \{o_1\}$ was detected, the hindrance rate for $cell_{3,j_{fix},k_{fix}}$) is equal to 0.25. The visibility rate in cells $cells_{4-10,j_{fix},k_{fix}}$ is 0.75.

3. *Time* t_2 (fig. 6.3b) - $\mathcal{O}_2(t_2) = \{o_1, o_2\}$ was detected, the additional hindrance rate for $cell_{5,j_{fix},k_{fix}}$ is 0.15. The visibility rate in $cells_{6-10,j_{fix},k_{fix}}$ is lowered by additional 0.15 and its set to 0.60 now.

4. *Time* t_3 (fig. 6.3c) - $\mathcal{O}_3(t_3) = \{o_1, o_2, o_3\}$ was detected the additional hindrance rate for $cell_{7,j_{fix},k_{fix}}$ is 0.20. The visibility rate in $cells_{8-10,j_{fix},k_{fix}}$ is lowered by additional 0.20 and its set to 0.40 now.

Map Obstacles: Use *stored LiDAR readings* from previous mission to build an compact obstacle map [3]. Then use *this map* as a additional information source.



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

Figure 6.4: Map obstacle states after *Data fusion*.

Concept: A *map obstacle* state has very simple logic, there are three possible cases:

1. *Undetected* - Map obstacle O_M is charted on map (fig. 6.4a), but is undetected by any sensor in sensor field, therefore the probability of map obstacle occurrence is equal to 0.

2. *Detected* Map obstacle O_M is charted on map and detected by any sensor in sensor field (fig. 6.4b). The map obstacle rate is equal to detected obstacle rate, usually its equal to 1.

3. *Hindered* Map obstacle O_M is hindered behind other detected obstacle O_1 (fig. 6.4c). The detected obstacle O_1 is in $cell_{i,j,k}$ and is reducing visibility in follow up $cellRow_{i_f>i,j,k}$ by 60 percent.

Implementation: The formulation of final map obstacle rate $map(cell_{i,j,k})$ was outlined in previous examples. These examples are showing the *desired behaviour* and its solved by *data fusion* (sec. 6.5.4).

First we start with obstacle map definition. The obstacle map (eq. 6.12) defines an map obstacle set of information vectors with position in global coordinate frame , orientation bounded to global coordinate reference frame, safety margin and additional parameters.

$$obstacleMap = \left\{ \begin{bmatrix} position, \\ orientation, \\ safetyMargin, \\ parameters \end{bmatrix} : \begin{array}{l} position \in \mathbb{R}^3(GCF), \\ orientation \in \mathbb{R}^3(GCF), \\ safetyMargin \in \mathbb{R}^+(m), \\ parameters \in \{\dots\} \end{array} \right\} \tag{6.12}$$

The *Map Obstacle* concept is taken from my *master student work* [3], implementing *compact representation* of point-cloud obstacle map. Te example of *cuboid obstacles* with *safe zone* is given in (fig. 6.5).

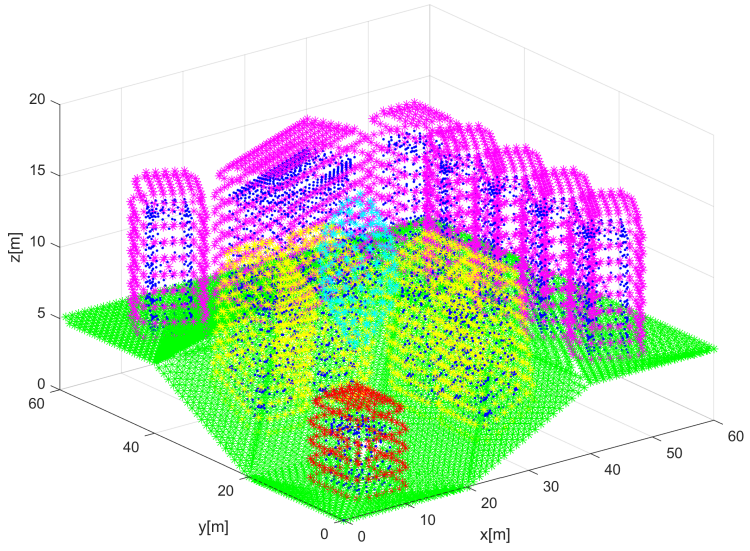


Figure 6.5: Example of Extracted Map Obstacle [3].

The space covered by any obstacle is non-empty by definition. There are following types of map charted obstacles which are implemented in framework:

1. *Ball obstacle parameters* $= \varnothing$ - simple ball with center at *position*, with offset safety margin.

2. *Line obstacle parameters* $= [length]$ - simple line bounded by length $\in]0, \infty[$ with center at *position* and given orientation with respect to main axis in global coordinate frame, with safety margin < 0.

3. *Plane obstacle parameters* $= [length, width]$ - bounded rectangle plane partition defined by length $\in]0, \infty[$, and width $w \in]0, \infty[$ with center at $\vec{p}$ and given orientation $\vec{o}$ with respect to main axis in global coordinate frame, with safety margin.

4. *Cuboid obstacle parameters* $= [length, width, depth]$ - bounded cuboid space partition defined by length $\in]0, \infty[$, width $\in]0, \infty[$, and depth $d \in]0, \infty[$ with center at *position* and rotated in orientation with respect to main axis in global coordinate frame, with safety margin.

The *map obstacles* are stored in clustered database. The *selection criterion* is given in (eq. 6.13).

$$avoidanceGrid.radius \geq distance(UAS.position, mapObstacle) - totalMargin \quad (6.13)$$

The *total margin* is combination of *safety margin* and *body margin* (in case of line, plane, cuboid obstacle). The *selection* was implemented as standard cluster select, selecting 26 surrounding clusters around UAS + own UAS cluster.
The *compact obstacle representation* is transformed into *h*omogeneous point-cloud representations:

1. *Body Point-cloud* - representing obstacle body approximation by geometrical shape (eq. 6.14). This point cloud is considered as hard constraints.

$$bodyPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapObstacleBody\} \quad (6.14)$$

2. *Safety Margin Point Cloud* - representing safety coating around mapped obstacle body approximation (eq. 6.15). This point cloud is considered as soft constraint.

$$marginPointCloud = \{point \in \mathbb{R}^3(GCF) : point \in mapSafetyMargin\} \quad (6.15)$$

Note. The *safety margin point cloud* is hollow in relationship to an *body point cloud*, therefore:

$$bodyPointCloud \cap marginPointCloud = \varnothing$$

The *map obstacle* discretization to point cloud leads to problem how to calculate *impact rate*. The *theoretical impact rate* for *obstacle* is given as:

$$impactRate = \frac{volume(mapObstacle \cap cell_{i,j,k})}{volume(cell_{i,j,k})} \in [0, 1]$$

The *map obstacle related point clouds* (eq. 6.14, 6.15) are homogeneous [3]. That means *each point* in point clouds covers similar portion of object volume. There is *threshold volume* (eq. 6.16) which represents minimal object volume to be considered as an *obstacle*.

$$0 < thresholdVolume \leq \frac{volume(pointCloud)}{|pointCloud|} \tag{6.16}$$

The *impact rate* of one point when intersecting a $cell_{i,j,k}$ is given as count of *threshold obstacle bodies* in *point cloud covered mass* multiplied by inverted point count (eq. 6.17).

$$point.rate = \frac{pointCloudVolume}{thresholdVolume} \times \frac{1}{|pointCloud|} \tag{6.17}$$

The *intersection set* between *point cloud* and $cell_{i,j,k}$ is defined in (eq. 6.17). The *cell* intersection with points is defined in (eq. **??**).

$$intersection(map, cell_{i,j,k}) = \ldots$$
$$\ldots \{points \in \mathbb{R}^3 : (point \rightarrow AvoidanceGridFrame) \in cell_{i,j,k}\} \tag{6.18}$$

The *map obstacle rating* for $cell_{i,j,k}$ and obstacle for our *information source* is defined in (eq. 6.19).

$$map(cell_{i,j,k}, obstacle) = \max \left\{ \sum_{\forall point \in intersection(map, cell_{i,j,k})} point.rate, 1 \right\} \tag{6.19}$$

The *map obstacle rating* for $cell_{i,j,k}$ and *our information source* is given as maximum of all possible cumulative ratings form each obstacle in *active map obstacles* set (eq. 6.20).

$$map(cell_{i,j,k} = \max \{map(cell_{i,j,k}, obstacle) : \forall obstacle \in ActiveMapObstacles\} \tag{6.20}$$

Note. The *body point clouds* (eq. 6.14) never intersects, because they are created for inclusive obstacles. The *safety margin point clouds* (eq. 6.15) can intersects, because they represents protection zones around physical obstacles. Therefore the *maximum obstacle rating* (eq. 6.20) needs to be selected.

6.5.2 Intruders

Intruder behaviour: *Adversarial behaviour* of moving obstacle is trying to destroy avoiding our UAS. The *Intruder* UAS [4] is not trying to hurt our *UAS* actively. The *Adversarial behaviour* is neglected in this work. The non-cooperative avoidance is assumed, it can be relaxed to *cooperative avoidance* in *UTM controlled airspace*.

Intruder information: The *observable intruder information set* for any kind of intruder, obtained through sensor/C2 line, is following:

1. *Position* - position of intruder in *local* or *global* coordinate frame, which can be transformed into *avoidance grid coordinate frame*.

2. *Heading and Velocity* - intruder heading and linear velocity in avoidance grid coordinate frame.

3. *Horizontal/Vertical Maneuver Uncertainty Spreads* - how much can an *intruder* deviate from *original linear path* in *horizontal/vertical* plane in *Global coordinate Frame*.

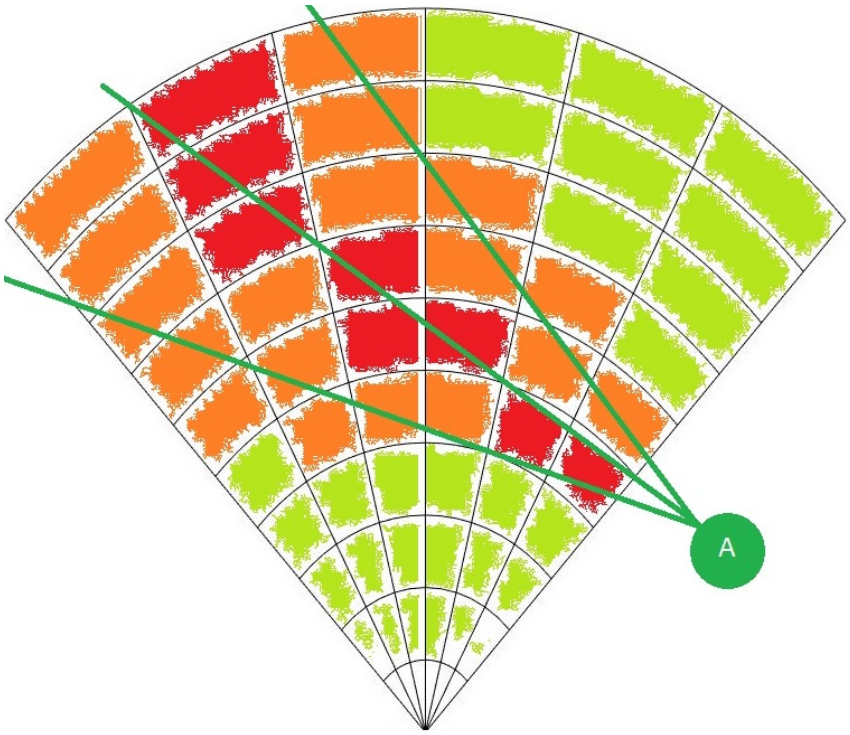


Figure 6.6: Intruder UAS intersection rate along expected trajectory.

Example of Intruder Intersection: Lets neglect the *time-impact* aspect on *intersection*. The *intruder* (black "I" circle) is intersecting one *avoidance grid horizontal slice* (fig. 6.6). The intruder is moving along linear path approximation based on velocity (middle green line). The *Horizontal Maneuver Uncertainty spread* is in *green line boundary area intruder intersection rating* is denoted as green-orange-red cell fill reflecting intersection severity: red is high rate of intersection, orange is medium rate of intersection and green is low rate of intersection.

Moving Threats: The *UAS* can encounter following threats during the *mission execution*:

1. *Non-cooperative Intruders* - the intruders whom does not implement any approach to ensure mutual avoidance efficiency.

2. *Cooperative Intruders* - the intruders whom actively communicate or follow common agreed behaviour pattern (ex. Rules of the Air).

3. *Moving Constraints* - the constrained portion of *free* space which is shifting its boundary over time (ex. Short term bad weather).

Note. Our approach considers only *UAS* intruders, because *Data Fusion* considers data received through *ADS-B* messages. The *Intruders* extracted from *LiDAR* scan were not considered (ex. birds). The proposed *intruder intersection models* are reusable for other *intruder sources*.

Approach Overview: The *Avoidance Grid* (def. **??**) is adapted to *LiDAR* sensor. The *euclidean grid intersections* are fairly simple. The *polar coordinates grid* are not. The need to keep *polar coordinates grid* is prevalent, because of fast *LiDAR* reading assessment. There are following commonly known methods to address this issue:

1. *Point-cloud Intersections* - the *threat impact area* is discredited into sufficiently thick point cloud. This point-cloud have *point impact rate* and *intersection time* assigned to each point. The *point-cloud* is projected to *Avoidance Grid*. If *impact point* hits $cell_{i,j,k}$ the cell's impact rate is increased by amount of *point impact rate*. The final *threat impact rate* in $cell_{i,j,k}$ is given when *all* points from point cloud are consumed. Close point problem [5] was solved by application of method [6].

2. *Polygon Intersections* - the *threat impact area* is modeled as polygon, each $cell_{i,j,k}$ in *Avoidance Grid* is considered as *polygon*. There is a possibility to calculate cell space geometrical inclusive intersection. The *impact rate* is then given as rate between *intersection volume* and $cell_{i,j,k}$ volume. The algorithm used for intersection selected based on:[7] the selected algorithm *Shamos-Hoey* [8].

Note. The *Intruder Intersection* models are based on *analytically geometry* for *cones and ellipsoids* taken from [9].

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

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

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

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

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

$$\mathrm{d}position/\mathrm{d}time = velocity \quad | \quad \begin{aligned} position_x(t) &= position_x(0) + velocity_x \times t \\ position_y(t) &= position_y(0) + velocity_y \times t \\ position_z(t) &= position_z(0) + velocity_z \times t \end{aligned} \tag{6.21}$$

Position vector in euclidean coordinates $[x, y, z]$ is transformed into *Avoidance Grid* coordinate frame. Velocity vector for $[x, y, z]$ is *estimated and not changing*. The time is in interval $[entry, leave]$, where *entry* is intruder entry time into avoidance grid and *leave* is intruder leave time from avoidance grid.

Note. If *intruder* is considered, time of entry is marked as $intruder_{entry,k}$ where k is intruder identification, time of leave is marked as $intruder_{leave,k}$ where k is intruder identification.

Cell Entry and Leave Times $UAS_{entry}(cell_{i,j,k})$ and $UAS_{leave}(cell_{i,j,k})$ are depending on intersecting *Trajectories* and *bounded cell space* (eq. **??**). There is *Trajectory Intersection* function from (def. **??**) which evaluates *Trajectory segment* entry and leave time.

The UAS *Cell Entry* time is given as minimum of all *passing trajectory segments* entry times (eq. 6.22), if there is no *passing trajectories* the UAS *entry time* is set to 0.

$$UAS_{entry}(cell_{i,j,k}) = \min \left\{ \begin{matrix} 0, entry(Trajectory, cell_{i,j,k}) : \\ Trajectory \in PassingTrajectories \end{matrix} \right\} \tag{6.22}$$

The UAS *Cell Leave* time is given as maximum of all *passing trajectory segments* entry times (eq. 6.23), if there is no *passing trajectories* the UAS *leave time* is set to 0.

$$UAS_{leave}(cell_{i,j,k}) = \max \left\{ \begin{matrix} 0, leave(Trajectory, cell_{i,j,k}) : \\ Trajectory \in PassingTrajectories \end{matrix} \right\} \tag{6.23}$$

Time Intersection Rate: The key idea is to calculate how long the *UAS* and *Intruder* spends together in same space portion ($cell_{i,j,k}$). The *Intruder* can spent some time in $cell_{i,j,k}$ bounded by interval of *intruder* entry/leave time.
The *UAS* can spent some time, depending on *selected trajectory* from *Reach Set*. The time spent by UAS is bounded by entry (eq. 6.22) and leave (eq. 6.23).

The intersection duration of these two intervals creates *time intersection rate* numerator, the *maximal duration* of *UAS* stay gives us *denominator*. The *time intersection rate* is formally defined in (eq. 6.24).

$$time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} = \circ \end{pmatrix} = \frac{\left| \begin{matrix} [intruder_{entry}(\circ), intruder_{leave}(\circ)] \\ \cap \\ [UAS_{entry}(\circ), UAS_{leave}(\circ)] \end{matrix} \right|}{|[UAS_{entry}(\circ), UAS_{leave}(cell_{\circ})]|} \tag{6.24}$$

Intruder Intersection Rate: The *Intruder Intersection Rate* (eq. 6.25) is calculated as *multiplication* of *space intersection rate* (defined later) and *time intersection rate* (eq. 6.24).

$$intruder \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} = time \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \times space \begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \tag{6.25}$$

Note. If there is no information to derive *Intruder* entry/leave time for cells the *time intersection rate* is considered 1.

The *Intruder cell reach* time (eq. 6.26) is bounded to discrete point in intersection model [5, 6]. The intruder *entry/leave time* is calculated similar to *UAS cell entry (eq. 6.22)/leave (eq. 6.23) time.*

$$pointReachTime(Intruder, point) = \frac{distance(Intruder.initialPosition, point)}{|Intruder.velocity} \tag{6.26}$$

Space Intersection Rate: The *Space Intersection Rate* reflects probability of *Intruder* intersection with portion of space bounded by $cell_{i,j,k}$, to be precise with intruder trajectory or vehicle body shifted along the trajectory. The principles for *space intersection rate* calculation are following:

1. *Line trajectory* - *intruder* trajectory is given by linear approximation (eq. 6.21), depending on *intruder size* the intersection with avoidance grid can be:

   a. *Simple line* - intersection is going along the trajectory line line defined by intruder model (eq.6.21).

   b. *Volume line* - intersection is going along the trajectory line defined by intruder model (eq. 6.21) and intruder‘s *body radius* is considered in intersection.

2. *Elliptic cone* - initial position is considered as the top of a cone, the main cone axis is defined by intruder linear trajectory (eq. 6.21) $time \in [0, \infty]$. The cone width is set by horizontal and vertical spread.

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

The constraint intersection with *avoidance grid* is done in *fixed decision Time*, for cell in *fixed cell leave time* (eq. 6.23), which means concept from static obstacles can be fully reused.

Definition: The *moving constraint definition* (eq. 6.38) covers minimal data scope for moving constraint, assuming linear constraint movement.

Definition 1. *Moving Constraints The original definition (eq. 6.31) is enhanced with additional parameters to support constraint moving:*

1. Velocity - *velocity vector on 2D horizontal plane.*
2. Detection time - *the time when* constraint *was created/detected, this is the time when* c*enter and boundary points position were valid.*

$$\begin{aligned} constraint = \{&position, boundary, \dots \\ &\dots, velocity, detectionTime, \dots \\ &\dots altitude_{start}, altitude_{end}, safetyMargin\} \end{aligned} \quad (6.27)$$

Cell Intersection: The *intersection algorithm* follows (eq. 6.35), only shift of the *center and boundary points* is required.

First let us introduce $\Delta time$ (eq. 6.39), which represents difference between the constraint detection time and expected cell leave time (eq. 6.23).

$$\Delta time = UAS_{leave}(cell_{i,j,k}) - detectionTime \quad (6.28)$$

The constraint boundary is shifted to:

$$\begin{aligned} shiftedBoundary(constraint) = \{&newPoint = point + velocity \times \Delta time : \dots \\ &\dots \forall point \in constraint.boundary\} \end{aligned} \quad (6.29)$$

The constraint center is shifted to:

$$shiftedCenter(constraint) = constraint.center + velocity \quad (6.30)$$

Note. The $\Delta time$ is calculated separately for each $cell_{i,j,k}$, because UAS is also moving and reaching cells in different times. The *cell leave time* can be calculated in advance after reach set approximation.

Alternative Intersection Implementation: The alternative used for intersection selected based on polygon intersection algorithms review [7], the selected algorithm is *Shamos-Hoey* [8].

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

6.5.3 Constraints

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

Converting constraints into valued *point-cloud* is not feasible, due the *huge amount of created points* and low *intersection rate*. The *polygon intersection* or *circular boundary of 2D polygon* is simple and effective solution [10, 11].

The key idea is to create *constraint barrels* around dangerous areas. Each *constraint barrel* is defined by circle on *horizontal plane* and *vertical limit range*.

Representation: The *minimal representation* is based on (sec. **??**, **??**) and geo-fencing principle. The *horizontal-vertical separation* is ensured by *projecting boundary* as 2D polygon oh horizontal plane and *vertical boundary* (barrel height) as *altitude limit*.

The *static constraint* (eq. 6.31) is defined as structure vector including:

1. *Position* - the center position in global coordinates*2D horizontal plane*.

2. *Boundary* - the ordered set of boundary points forming edges in global coordinates*2D horizontal plane*.

3. *Altitude Range* - the *barometric altitude* range $[altitude_{start}, altitude_{end}]$.

4. *Safety Margin* - the *protection zone* (soft constraint) around constraint body (hard constraints) in meters.

$$constraint = \{position, boundary, altitude_{start}, altitude_{end}, safetyMargin\} \tag{6.31}$$

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

The *active constraints set* (eq. 6.32) is defined as set of *constraints* from all *reliable Information Sources* where the *the distance* between UAS and constraint body (including safety margin) is lesser than the avoidance grid range. The *horizontal altitude range* of avoidance grid musts also intersect with *constraint altitude range*.

$$\begin{aligned} &ActiveConstraints = \dots \\ &\cdots = \left\{ \begin{array}{l} constraint \in InformationSource : \\ \qquad distance(constraint, UAS) \leq AvoidanceGrid.distance, \\ \qquad constraint.altitudeRange \cap UAS.altitudeRange \neq \varnothing \end{array} \right\} \end{aligned} \tag{6.32}$$

Cell Intersection: The *importance of constraints* is on their impact on *avoidance grid cells*. The *most of the constraints* (weather, ATC) are represented as 2D convex polygons. Even the *irregularly shaped constraints* are usually split into smaller convex 2D polygons.

The idea is to represent convex polygon boundary as sufficiently large circle to cover polygon. The Welzl algorithm to find *minimal polygon cover circle* [11] is used.

First the *set of contraint edges* (eq. 6.33) is a enclosed set of 2D edges between neighboring points defined as follow:

$$edges(constraint) = \left\{ \begin{array}{l} \\ \left[point_i, point_j\right] : i \in \{1, \ldots, |boundary|\}, \\ \\ \end{array} \begin{array}{l} point \in boundary, \\ \\ j \in \{2, \ldots, |boundary|, 1\} \end{array} \right\} \tag{6.33}$$

The *constraint circle boundary* with calculated center on 2D horizontal plane and radius (representing body margin) is defined in (eq. 6.34).

$$circle(constraint) = \begin{bmatrix} center = \dfrac{\sum boundary.point}{|boundary.point|} + correction \\ radius = smallestCircle(edges(constraints)) \end{bmatrix} \tag{6.34}$$

The ($cell_{i,j,k}$ and *constraint* intersection (eq. 6.35) is classification function. The *classification* is necessary, because one *constraint* induce:

1. *Body Constraint* (hard constraint) - the distance between $cell_{i,j,k}$ closest border and *circular boundary* center is in interval $[0, radius]$.

2. *Protection Zone Constraint* (soft constraint) - the distance between $cell_{i,j,k}$ closest border and *circular boundary* center is in interval $]radius, radius + safetyMargin]$.

$$intersection, constraint) = \ldots$$

$$\ldots = \begin{cases} hard & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) \leq \ldots \\ \quad \ldots \leq circle(constraint).radius, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\ soft & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) > \ldots \\ \quad \ldots > circle(constraint).radius, \\ distance(cell_{i,j,k}, circle(constraint)) \leq \ldots \\ \quad \ldots \leq circle(constraint).radius + safetyMargin, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \varnothing, \end{bmatrix} \\ none & : otherwise \end{cases} \tag{6.35}$$

The *intersection impact* of constraint is handled separately for *soft* and *hard* constraints. The *avoidance* of hard constraints is *mandatory*, the *avoidance* of soft constraints is *voluntary*.

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

$$cell_{i,j,k}.softConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = soft \end{matrix} \right\} \tag{6.36}$$

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

$$cell_{i,j,k}.hardConstraints = \left\{ \begin{matrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = hard \end{matrix} \right\} \tag{6.37}$$

Note. The final *constraint rate value* (eq. 6.46) is determined based on *mission control run* feed to *avoidance grid* (fig. **??**) defined in 7^{th} to 10^{th} step.

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

The constraint intersection with *avoidance grid* is done in *fixed decision Time*, for cell in *fixed cell leave time* (eq. 6.23), which means concept from static obstacles can be fully reused.

Definition: The *moving constraint definition* (eq. 6.38) covers minimal data scope for moving constraint, assuming linear constraint movement.

Definition 2. *Moving Constraints The original definition (eq. 6.31) is enhanced with additional parameters to support constraint moving:*

1. Velocity - *velocity vector on 2D horizontal plane.*
2. Detection time - *the time when* constraint *was created/detected, this is the time when* c*enter and boundary points position were valid.*

$$\begin{aligned} constraint = \{&position, boundary, \dots \\ &\dots, velocity, detectionTime, \dots \\ &\dots altitude_{start}, altitude_{end}, safetyMargin\} \end{aligned} \quad (6.38)$$

Cell Intersection: The *intersection algorithm* follows (eq. 6.35), only shift of the *center and boundary points* is required.

First let us introduce $\Delta time$ (eq. 6.39), which represents difference between the constraint detection time and expected cell leave time (eq. 6.23).

$$\Delta time = UAS_{leave}(cell_{i,j,k}) - detectionTime \quad (6.39)$$

The constraint boundary is shifted to:

$$\begin{aligned} shiftedBoundary(constraint) = \{&newPoint = point + velocity \times \Delta time : \dots \\ &\dots \forall point \in constraint.boundary\} \end{aligned} \quad (6.40)$$

The constraint center is shifted to:

$$shiftedCenter(constraint) = constraint.center + velocity \quad (6.41)$$

Note. The $\Delta time$ is calculated separately for each $cell_{i,j,k}$, because UAS is also moving and reaching cells in different times. The *cell leave time* can be calculated in advance after reach set approximation.

Alternative Intersection Implementation: The alternative used for intersection selected based on polygon intersection algorithms review [7], the selected algorithm is *Shamos-Hoey* [8].

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

6.5.4 Data fusion

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

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

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

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

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

The *visibility* for the cell is given in (eq. 6.42) as minimal visibility calculated from all capable sensors in *Sensor Field*.

$$visibility(cell_{i,j,k}) = \min \left\{ \begin{matrix} visibility(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \tag{6.42}$$

The example of *visibility* calculation for *LiDAR* sensor is given in (fig. 6.4).

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

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

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

$$obstacle(cell_{i,j,k}) = \max \left\{ \begin{matrix} obstacle(cell_{i,j,k}, sensor_i) : \\ \forall sensor_i \in SensorField \end{matrix} \right\} \times \ldots \\ \ldots \times visibility(cell_{i,j,k}) \tag{6.43}$$

The example of *detected obstacle rating* calculation for *LiDAR* sensor is given in (eq. 6.3).

Map Obstacle: The *Information Sources* are feeding *Avoidance Grid* with partial information of *Map obstacle rating*. *Map Obstacle Rating* shows the certainty that *charted*

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

The *Map Obstacle Rating* for a cell (eq. 6.44) is calculated as the product of maximal *Map Obstacle Rating* and *inverse visibility*. This gives *visibility biased* certainty of *Map Obstacle*.

$$map(cell_{i,j,k}) = \max \left\{ \begin{matrix} map(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \times \ldots \\ \cdots \times (1 - visibility(cell_{i,j,k})) \quad (6.44)$$

The example of *Map Obstacle Rating* calculation is given in (fig. 6.4).

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

The *geometrical bias* is used for cumulative of multiple intruders; the *intruders are not cooperative*; therefore their occurrence cannot be addressed by the simple *maximum*. The proposed formula (eq. 6.45) is simply bypassing the intruder rating if there is one intruder. If there are more intruders, the geometrical bias is applied.

$$intruder(cell_{i,j,k}) = 1 - \prod_{\forall intruder_i \in Intruders} \left(1 - intersection \begin{pmatrix} cell_{i,j,k}, \\ intruder_i \end{pmatrix} \right) \quad (6.45)$$

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

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

The *Constraints rating* (eq. 6.46) is in the *range* $[0, 1]$ reflecting certainty of constraint application in the cell (usually 1).

$$constraint(cell_{i,j,k}) = \max \left\{ \begin{matrix} constraint(cell_{i,j,k}, source_i) : \\ \forall source_i \in InformationSources \end{matrix} \right\} \quad (6.46)$$

The *Constraint Rating* calculation example for *static* constraints is given in (sec. 6.5.3), the example for *moving* constraints is given by (def. 2).

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

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

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

Definition 3. *The Threat is considered as any source of harm. The threat is a* maximal aggregation *of various harm ratings. Our* threat *for a specific cell is defined by (eq. 6.47).*

$$threat(cell_{i,j,k}) = \max \begin{Bmatrix} obstacle(cell_{i,j,k}), map(cell_{i,j,k}), \\ intruder(cell_{i,j,k}), constraint(cell_{i,j,k}) \end{Bmatrix} \tag{6.47}$$

Reachability: The *Reachability* for trajectory reflects how safe is the *path along*. The *Threat* (def. 3) for each cell has been already assessed. The set of *Passing Cells* is defined in *Trajectory Footprint* (eq. **??**).

The *Trajectory Reachability* is given as a product of *Threats* along the trajectory (eq. 6.48). The *Trajectory Reachability* can be calculated for each *trajectory segment* given as $\{movement_1, \ldots, movement_i\} \subset Buffer$ originating from $state_0$.

$$reachibility(Trajectory) = \prod_{PassingCells}^{\forall cell_{i,j,k} \in} (1 - threat(c_{i,j,k})) \tag{6.48}$$

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

The *Reachability* of the *cell* is given by the best trajectory segment passing through the *given cell*. This is given by property, that every trajectory is originating from root $state_0$, which means that one safe route is sufficient to reach space in the cell.

The *Trajectory segment* reachability is sufficient, because the overall performance is not interesting, the *local reachability* is sufficient. The cell reachibility is formally defined in (eq. 6.49).

$$reachability(cell_{i,j,k}) = \max\{Trajectory.Segment(cell_{i,j,k}).Reachability : \forall Trajectory \in PassingTrajectories(celli, j, k)\} \tag{6.49}$$

Note. Function Trajectory.Segment($cell_{i,j,k}$). Reachability gives same results for any segment in $cell_{i,j,k}$, because (eq. 6.48) 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* [12] demands the fail rate below 10^{-7} because there is no definition for *UAS* the *minimal fail rate* is expected to be at a similar level.

The *fuzzy values* $[0, 1]$ are projected to *Boolean* properties of *cell* and *Trajectory* in the following manner (tab. 6.1).

The high values of *Visibility* (eq. 6.42) and *Reachability* (eq. 6.49, 6.48) are expected. The low *threshold* for *threats* values is expected. The error margin is solved by *Sensor Fusion*, therefore, initial *false positive* cases have a low rate. The *Detected Obstacle Rate* (eq. 6.43), *Map Obstacle Rate* (eq. 6.44), *Intruder Rate* (eq. 6.45), and *Constraint Rate* (eq. 6.46) thresholds are considered low.

Threshold $= 10^{-7}$

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

Table 6.1: Changing ratings from fuzzy to Boolean parameters.

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

The *Uncertain* space for decision time t_i is a portion of *Avoidance Grid* which *UAS* cannot *read* with *Sensor Field*. The *cells* with a $\neg Visible$ property. The *Uncertain* space is given by (eq. 6.50).

$$Uncertain(t_i) = \{cell_{i,j,k} : cell_{i,j,k} \in AvoidanceGrid(t_i), cell_{i,j,k}.\neg Visible\} \tag{6.50}$$

The *Occupied* space for decision time t_i is the set of cells which are classified as *Detected Obstacles*. The *Visibility* is not an issue, due to the initial damping in (eq. 6.43). The formal definition is the space portion where it is possible to detect *obstacle bodies* or their portions (eq. 6.51).

$$Occupied(t_i) = \left\{ cell_{i,j,k} : \begin{matrix} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.DetectedObstacle \end{matrix} \right\} \tag{6.51}$$

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. 6.52).

$$Constrained(t_i) = \left\{ cell_{i,j,k} : \begin{array}{c} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Visible, \\ cell_{i,j,k}.Constraint \vee cell_{i,j,k}.Intruder \end{array} \right\} \tag{6.52}$$

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. 6.53).

$$\begin{aligned} Free(t_i) = AvoidanceGrid(t_i) - \ldots \\ \cdots - (Uncertain(t_i) \cup Occupied(t_i) \cup Constrained(t_i)) \end{aligned} \tag{6.53}$$

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. 6.54).

$$Reachable(t_i) = \left\{ cell_{i,j,k} : \begin{array}{c} cell_{i,j,k} \in AvoidanceGrid(t_i), \\ cell_{i,j,k}.Reachable \end{array} \right\} \tag{6.54}$$

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

$$|Reachable(t_i)| > 0, \quad Reachable(t_i) \subset Free(t) \tag{6.55}$$

Bibliography

[1] Alojz Gomola, Pavel Klang, and Jan Ludvik. Probabilistic approach in data fusion for obstacle avoidance framework based on reach sets. In *Internal publication collection*, pages 1–93. Honeywell, 2017.

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

[3] Maria Cerna. Usage of maps obtained by lidar in uav navigation. Master thesis, Institute of Automotive Mechatronics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovicova 3, Bratislava. Slovak Republic, jun 2018.

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

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

[6] Jon Louis Bentley, Bruce W Weide, and Andrew C Yao. Optimal expected-time algorithms for closest point problems. *ACM Transactions on Mathematical Software (TOMS)*, 6(4):563–580, 1980.

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

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

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

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