6.5 Static Obstacles and Constraints

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

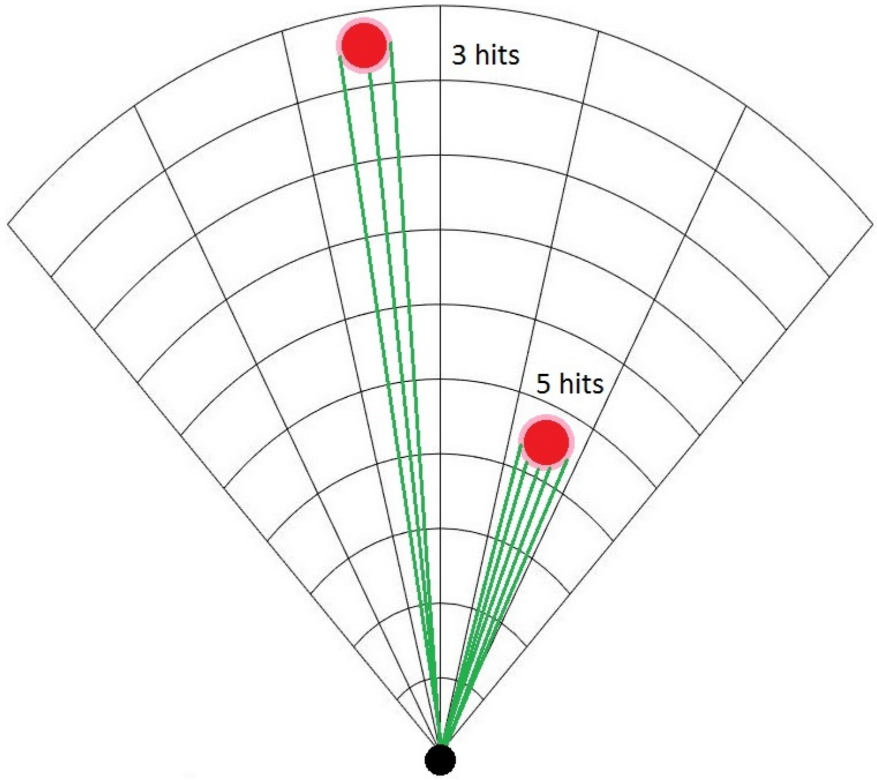


Figure 6.1: Different count of LiDAR hits with different distance from UAS.

The surface of area given by some distance d, and unitary offsets$\partial horizontal^{\circ}, \partial vertical^{\circ}$ 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).

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].
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).

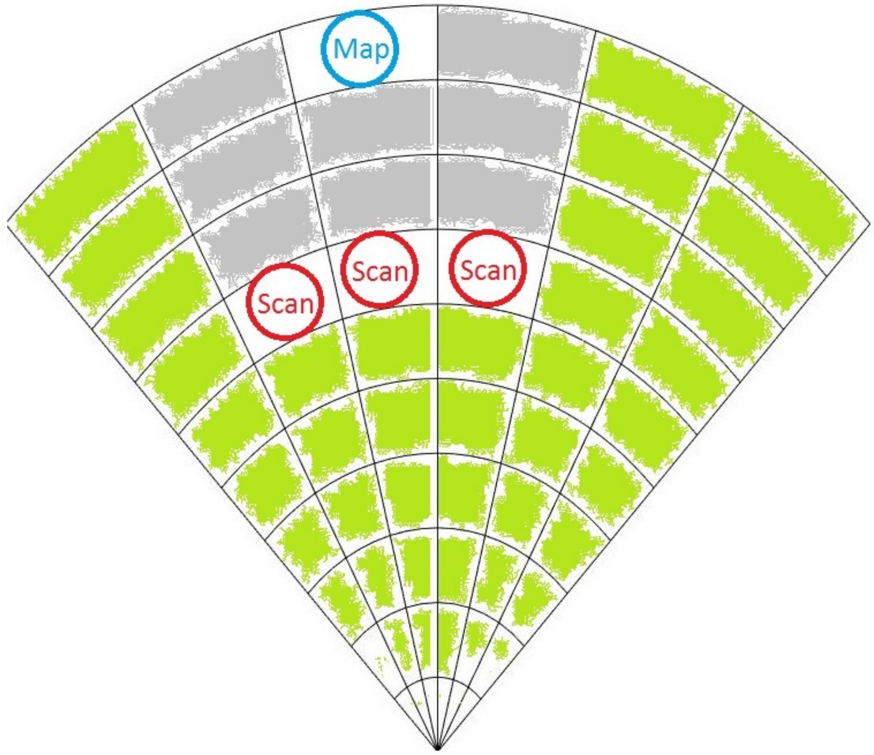


Figure 6.2: Overshadowed map obstacle by detected obstacles.

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.

6.5.1 Detected Obstacles

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

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

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

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:

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

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:

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

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).

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

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

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

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

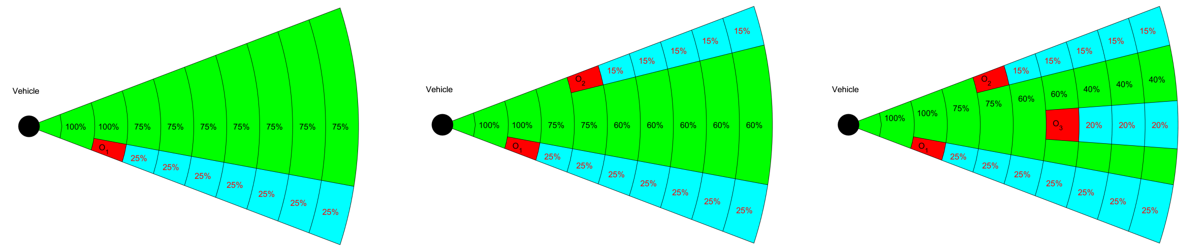
$$visibility(cell_{i_c,j_c,k_c}) = \dots$$
$$\dots = 1 - \sum_{index \in \mathbb{N}^+}^{index<i_c} hindrance(cell_{a,j_c,k_c} : cell_{a,j_c,k_c} \in cellRow(j_c, k_c) \tag{6.10}$$

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.

6.5.2 Map Obstacles

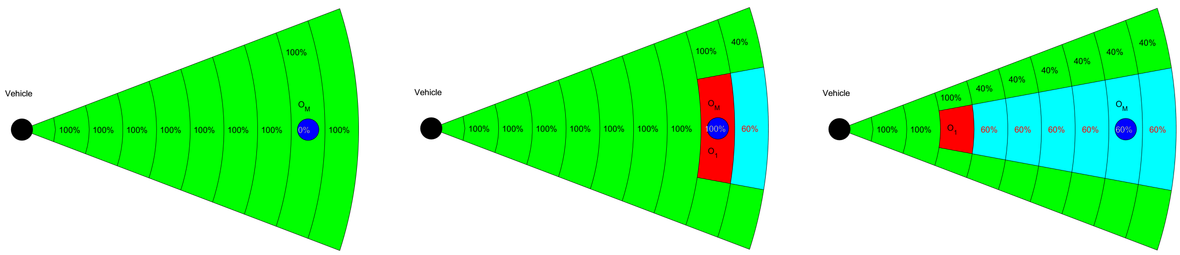
Idea: Use *stored LiDAR readings* from previous mission to build an compact obstacle map [3]. Then use *this map* as a additional information source.

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.



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

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

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. **??**).

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.

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

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).

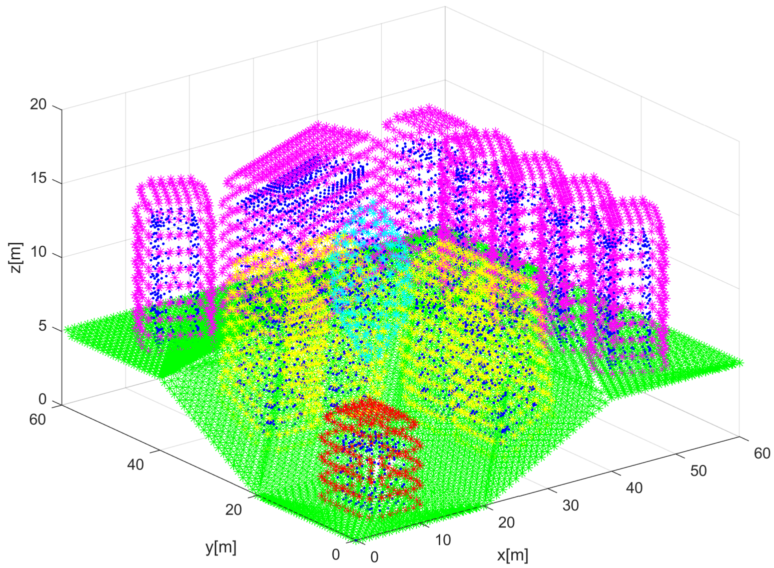


Figure 6.5: Example of Extracted Map Obstacle [3].

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).

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

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.

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

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

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

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. **??**).

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

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

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

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).

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

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.

6.5.3 Static Constraints

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

Converting constraints into valued *point-cloud* is not feasible, due the *huge amount of created points* and low *intersection rate*. The *polygon intersection* or *circular boundary of 2D polygon* is simple and effective solution [4, 5].

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. 6.21) is defined as structure vector including:

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

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

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

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

$$constraint = \{position, boundary, altitude_{start}, altitude_{end}, safetyMargin\} \quad (6.21)$$

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

The *active constraints set* (eq. 6.22) 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.*

$$\begin{aligned} ActiveConstraints &= \dots \\ \dots &= \left\{ \begin{aligned} &constraint \in InformationSource : \\ &\qquad distance(constraint, UAS) \leq AvoidanceGrid.distance, \\ &\qquad constraint.altitudeRange \cap UAS.altitudeRange \neq \varnothing \end{aligned} \right\} \end{aligned} \quad (6.22)$$

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* [5] is used.

First the *set of contraint edges* (eq. 6.23) is a enclosed set of 2D edges between neighboring points defined as follow:

$$edges(constraint) = \left\{ \left[point_i, point_j\right] : \begin{aligned} &point \in boundary, \\ &i \in \{1, \dots, |boundary|\}, \\ &j \in \{2, \dots, |boundary|, 1\} \end{aligned} \right\} \quad (6.23)$$

The *constraint circle boundary* with calculated center on 2D horizontal plane and radius (representing body margin) is defined in (eq. 6.24).

$$circle(constraint) = \begin{bmatrix} center = \dfrac{\sum boundary.point}{|boundary.point|} + correction \\ radius = smallestCircle(edges(constraints)) \end{bmatrix} \quad (6.24)$$

The ($cell_{i,j,k}$ and *constraint* intersection (eq. 6.25) 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 $]radius, radius + safetyMargin]$.

$$intersection, constraint) = \dots$$

$$\dots = \begin{cases} hard & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) \leq \dots \\ \quad \dots \leq circle(constraint).radius, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \emptyset, \end{bmatrix} \\ \\ soft & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) > \dots \\ \quad \dots > circle(constraint).radius, \\ distance(cell_{i,j,k}, circle(constraint)) \leq \dots \\ \quad \dots \leq circle(constraint).radius + safetyMargin, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \emptyset, \end{bmatrix} \\ \\ none & : otherwise \end{cases} \quad (6.25)$$

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:

$$cell_{i,j,k}.softConstraints = \begin{Bmatrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = soft \end{Bmatrix} \quad (6.26)$$

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

$$cell_{i,j,k}.hardConstraints = \left\{\begin{aligned} &constraint \in ActiveConstraints : \\ &\quad intersection(cell_{i,j,k}, constraint) = hard \end{aligned}\right\} \tag{6.27}$$

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

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

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