6.4.1 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 [1, 2].

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. 6.1) 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.

$$constraint = \{position, boundary, altitude_{start}, altitude_{end}, safetyMargin\} \tag{6.1}$$

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

The *active constraints set* (eq. 6.2) 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.*

$$\begin{aligned} &ActiveConstraints = \dots \\ &\dots = \left\{ \begin{aligned} &constraint \in InformationSource : \\ &\qquad distance(constraint, UAS) \leq AvoidanceGrid.distance, \\ &\qquad constraint.altitudeRange \cap UAS.altitudeRange \neq \varnothing \end{aligned} \right\} \end{aligned} \tag{6.2}$$

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* [2] is used.

First the *set of contraint edges* (eq. 6.3) is a enclosed set of 2D edges between neighboring points defined as follow:

$$edges(constraint) = \left\{ \left[point_i, point_j\right] : \begin{array}{c} point \in boundary, \\ i \in \{1, \dots, |boundary|\}, \\ j \in \{2, \dots, |boundary|, 1\} \end{array} \right\} \tag{6.3}$$

The *constraint circle boundary* with calculated center on 2D horizontal plane and radius (representing body margin) is defined in (eq. 6.4).

$$circle(constraint) = \begin{bmatrix} center = \dfrac{\sum boundary.point}{|boundary.point|} + correction \\ radius = smallestCircle(edges(constraints)) \end{bmatrix} \tag{6.4}$$

The ($cell_{i,j,k}$ and *constraint* intersection (eq. 6.5) 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 $]radius, radius + safetyMargin]$.

$$intersection, constraint) = \dots$$
$$\dots = \begin{cases} hard & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) \leq \dots \\ \quad \dots \leq circle(constraint).radius, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \emptyset, \end{bmatrix} \\ \\ soft & : \begin{bmatrix} distance(cell_{i,j,k}, circle(constraint)) > \dots \\ \quad \dots > circle(constraint).radius, \\ distance(cell_{i,j,k}, circle(constraint)) \leq \dots \\ \quad \dots \leq circle(constraint).radius + safetyMargin, \\ constraint.altitudeRange \cap cell_{i,j,k}.altitudeRange \neq \emptyset, \end{bmatrix} \\ \\ none & : otherwise \end{cases} \tag{6.5}$$

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:

$$cell_{i,j,k}.softConstraints = \begin{Bmatrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = soft \end{Bmatrix} \tag{6.6}$$

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

$$cell_{i,j,k}.hardConstraints = \begin{Bmatrix} constraint \in ActiveConstraints : \\ \quad intersection(cell_{i,j,k}, constraint) = hard \end{Bmatrix} \tag{6.7}$$

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

6.4.2 Moving Constraints

Moving Constraints: The basic ideas is the same as in case *static constraints* (sec. 6.4.1). 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. **??**), which means concept from static obstacles can be fully reused.

Definition: The *moving constraint definition* (eq. 6.8) covers minimal data scope for moving constraint, assuming linear constraint movement.

Definition 1. *Moving Constraints The original definition (eq. 6.1) 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.*

$$\begin{aligned} constraint = \{position, boundary, \dots \\ \dots, velocity, detectionTime, \dots \\ \dots altitude_{start}, altitude_{end}, safetyMargin\} \end{aligned} \tag{6.8}$$

Cell Intersection: The *intersection algorithm* follows (eq. 6.5), only shift of the *center and boundary points* is required.

First let us introduce $\Delta time$ (eq. 6.9), which represents difference between the constraint detection time and expected cell leave time (eq. **??**).

$$\Delta time = UAS_{leave}(cell_{i,j,k}) - detectionTime \tag{6.9}$$

The constraint boundary is shifted to:

$$\begin{aligned} shiftedBoundary(constraint) = \{newPoint = point + velocity \times \Delta time : \dots \\ \dots \forall point \in constraint.boundary\} \end{aligned} \tag{6.10}$$

The constraint center is shifted to:

$$shiftedCenter(constraint) = constraint.center + velocity \tag{6.11}$$

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 [3], the selected algorithm is *Shamos-Hoey* [4].

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

Bibliography

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

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

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

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