6.5.1 Intruders

Intruder behaviour: *Adversarial behaviour* of moving obstacle is trying to destroy avoiding our UAS. The *Intruder* UAS [1] 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.*

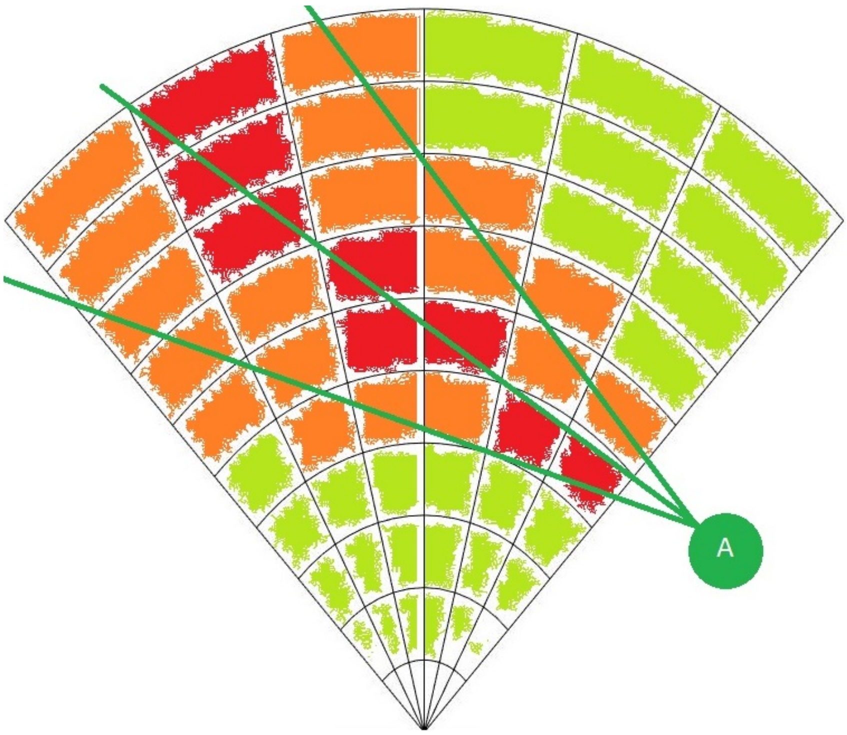

Figure 6.1: Intruder UAS intersection rate along expected trajectory.

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. 6.1). 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 [2] was solved by application of method [3].
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:[4] the selected algorithm *Shamos-Hoey* [5].

Note. The *Intruder Intersection* models are based on *analytically geometry* for *cones and ellipsoids* taken from [6].

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:

$$\mathrm{d}position/\mathrm{d}time = velocity \quad \Big| \quad \begin{aligned} position_x(t) &= position_x(0) + velocity_x \times t \\ position_y(t) &= position_y(0) + velocity_y \times t \\ position_z(t) &= position_z(0) + velocity_z \times t \end{aligned} \tag{6.1}$$

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. 6.2), if there is no *passing trajectories* the UAS *entry time* is set to 0.

$$UAS_{entry}(cell_{i,j,k}) = \min \left\{ \begin{aligned} &0, entry(Trajectory, cell_{i,j,k}) : \\ &\qquad Trajectory \in PassingTrajectories \end{aligned} \right\} \tag{6.2}$$

The UAS *Cell Leave* time is given as maximum of all *passing trajectory segments* entry times (eq. 6.3), if there is no *passing trajectories* the UAS *leave time* is set to 0.

$$UAS_{leave}(cell_{i,j,k}) = \max \left\{ \begin{aligned} &0, leave(Trajectory, cell_{i,j,k}) : \\ &\qquad Trajectory \in PassingTrajectories \end{aligned} \right\} \tag{6.3}$$

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. 6.2) and leave (eq. 6.3).

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. 6.4).

$$time\begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} = \circ \end{pmatrix} = \frac{\left| \begin{matrix} [intruder_{entry}(\circ), intruder_{leave}(\circ)] \\ \cap \\ [UAS_{entry}(\circ), UAS_{leave}(\circ)] \end{matrix} \right|}{|[UAS_{entry}(\circ), UAS_{leave}(cell_{\circ})]|} \tag{6.4}$$

Intruder Intersection Rate: The *Intruder Intersection Rate* (eq. 6.5) is calculated as *multiplication* of *space intersection rate* (defined later) and *time intersection rate* (eq. 6.4).

$$intruder\begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} = time\begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \times space\begin{pmatrix} UAS, \\ Intruder, \\ cell_{i,j,k} \end{pmatrix} \tag{6.5}$$

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. 6.6) is bounded to discrete point in intersection model [2, 3]. The intruder *entry/leave time* is calculated similar to *UAS cell entry (eq. 6.2)/leave (eq. 6.3) time.*

$$pointReachTime(Intruder, point) = \frac{distance(Intruder.initialPosition, point)}{|Intruder.velocity} \tag{6.6}$$

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. 6.1), 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.6.1).

   b. *Volume line* - intersection is going along the trajectory line defined by intruder model (eq. 6.1) 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. 6.1) $time \in [0, \infty]$. The cone width is set by horizontal and vertical spread.

Moving Constraints: The basic ideas is the same as in case *static constraints* (sec. **??**). 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. 6.3), which means concept from static obstacles can be fully reused.

Definition: The *moving constraint definition* (eq. 6.7) covers minimal data scope for moving constraint, assuming linear constraint movement.

Definition 1. *Moving Constraints The original definition (eq.* **??***) 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* center *and boundary points position were valid.*

$$constraint = \{position, boundary, \dots$$
$$\dots, velocity, detectionTime, \dots$$
$$\dots altitude_{start}, altitude_{end}, safetyMargin\} \quad (6.7)$$

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

First let us introduce $\Delta time$ (eq. 6.8), which represents difference between the constraint detection time and expected cell leave time (eq. 6.3).

$$\Delta time = UAS_{leave}(cell_{i,j,k}) - detectionTime \quad (6.8)$$

The constraint boundary is shifted to:

$$shiftedBoundary(constraint) = \{newPoint = point + velocity \times \Delta time : \dots$$
$$\dots \forall point \in constraint.boundary\} \quad (6.9)$$

The constraint center is shifted to:

$$shiftedCenter(constraint) = constraint.center + velocity \tag{6.10}$$

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 [4], the selected algorithm is *Shamos-Hoey* [5].

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

Bibliography

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

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

[3] 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.

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

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

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