Chapter 3

(R) Background Theory

Motivation: Cooperative and Non-Cooperative *Sense and Avoid* (SAA) systems are key enablers for the *Unmanned Aerial Systems* (UAS) to routinely access non-segregated airspace [1]. Both cooperative and non-cooperative SAA systems are being developed to address this integration requirement.

The *SAA capability* is defined as the automatic detection of possible conflicts by the UAS platform under consideration and performing avoidance maneuvers to prevent the identified collisions.

An analysis of the available SAA candidate technologies and the associated sensors for both cooperative and non-cooperative SAA systems is presented in [2].

Non-cooperative *Collision Detection and Resolution* (CD&R) for UAV is considered as one of the major challenges that needs to be addressed [3] for the insertion of UAVs in non-segregated air space. As a result, a number of non-cooperative sensors for the SAA system have been adopted. Light Detection and Ranging (LIDAR)is used for detecting, warning and avoiding obstacles for low-level flying [4].

An approach to the definition of encounter models and their applications to SAA strategies is presented in [5] for both cooperative and non-cooperative scenarios.

Since 2014, there is a visible strong political support for developing rules on drones but regulations are not harmonized yet. The European Aviation Safety Agency (EASA) has been tasked to develop a regulatory framework for drone operations and proposals for the regulation of "low-risk" UAV operations. In achieving this, EASA is working closely with the Joint Authorities for Regulation of Unmanned Systes (JARUS) [6].

Background Areas: Following Areas are introduced in this chapter:

1. *UAS System Model* (sec. 3.1) - continuous and discrete mathematical models.
2. *Reach Sets* (sec. 3.2) - introduction to *Reach set* representation and calculation methods.
3. *Movement Automaton* (sec. 3.3) - intuitive definition and establishment of *hybrid automaton* control and modeling.
4. *LiDAR* (sec. 3.4) - short summary of *LiDAR* technology and terminology introduction.
5. *Complements of Algebra* (sec. 3.5) - necessary algebra complements for *Local* and *Global Coordinate Systems*.

3.1 (R) UAS System Model

This section strongly follows [7].

3.1.1 Continuous-time systems

Consider a class of systems given by functions:

$$\begin{aligned} &StateEvolution : input(time) \rightarrow state(state_0, time) \\ &input(time) : [0, FinalTime] \rightarrow \mathbb{R}^p \\ &input(time) \in \mathbb{R}^p, state(time) \in \mathbb{R}^n \end{aligned} \tag{3.1}$$

Where $input(time)$ and $state(state_0, time)$ are a sets of continuous-time signals. These are often called continuous-time systems because they operate on continuous-time signals.

Frequently, such systems can be defined by differential equations that relate the input signal to the output signal.

A prototypical description of a controlled (there is a control input signal) continuous-time system is:

$$\begin{aligned} &\partial/\partial \text{t state}(time) = \\ &\qquad f(time, state(time), input(time)), input(time) \in Inputs(time) \end{aligned} \tag{3.2}$$

Where $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^p \rightarrow \mathbb{R}^n$ satisfies the conditions for existence and uniqueness of the ordinary differential equation and u is our control [8].

3.1.2 Discrete-time systems

Consider another class of systems given by functions

$$\begin{aligned} &StateEvolution :input(k) \rightarrow state(k), \\ &\qquad k \in \{0, t_s, 2.t_s, 3.t_s, \ldots i.t_s\}, i \in \mathbb{N}^+ \\ &input(k) \in \mathbb{R}^p, state(k) \in \mathbb{R}^n \end{aligned} \tag{3.3}$$

Where $input(k)$, $state(k)$ is a set of discrete-time signals. They can be represented by a function f like $f : \{0, t_s, 2.t_s, 3.t_s, \ldots i.t_s\} \rightarrow \mathbb{R}^n, i \in \mathbb{N}^+$ where t_s is sampling time and i is discrete step [9].

3.1.3 Adversarial behaviour in continuous systems

Consider a subclass of continuous time systems where are two sets of control signals $uas(time)$ and $adversary(time)$ which are accommodated in following system:

$$\begin{aligned} &\partial/\partial \text{t state}(time) = f(t, state(time), uas(time), adversary(time)), \\ &\qquad uas(time) \in UASInputs(time) \subset \mathbb{R}^u, \\ &adversary(time) \in AdversaryInputs(time) \subset \mathbb{R}^v \end{aligned} \tag{3.4}$$

This system representation is often used in definition of problem of pursuit/evasion problem. Krasovskii developed a solution approach to this problem in [10]. A complex example of can be found in article [11].

3.2 Reach Sets

Informally, the *Reach Set* of a UAS system described by a differential equation is the *set of all states that can be reached from an initial state within a given time interval.*

3.2.1 Definitions

For following definitions consider *nonlinear UAS system* described in (sec. 3.1).

Definition 1 (Reach set starting at a given point)**.** *Suppose the initial position and time* $(state_0, time_0)$ *are given. The reach set* $ReachSet[\tau, time_0, state_0]$ *of* nonlinear system *at time* $\tau \geq time_0$*, starting at* $(state_0, time_0)$ *is given by:*

$$ReachSet[\tau, time_0, state_0] = \bigcup\{state(\tau) : input(s) \in Inputs(s), s \in (time_0, \tau]\} \quad (3.5)$$

Reach set starting at given set can be used to determine reach set in case of *hybrid system* input control switch and it is defined as follow:

Definition 2. *set starting at a given set] The reach set at time* $\tau > t_0$ *starting from set* $States_0$ *is defined as:*

$$ReachSet[\tau, time_0, States_0] = \bigcup\{ReachSet[\tau, time_0, state_0] : state_0 \in States_0\} \quad (3.6)$$

Reach set for adversarial behavior can be used to calculate possible escape routes from pursuer and it is defined as follow:

Definition 3 (Reach set under adversarial behavior)**.** *Consider now the case of adversarial behavior([10, 11]). where* $input(t)$ *is our control and* $adversary(t)$ *is adversary control which is independent of* $input(t)$*, let* $differentialControl(t) = input(t) - \sup_{state \in state(t)} adversary(t)$*, which represents worst possible input change in given state and time, then* reach set for system *is represented as:*

$$ReachSet\begin{bmatrix}\tau,\\ time_0,\\ state_0\end{bmatrix} = \bigcup\left\{state(\tau) : \begin{matrix}differentialControl(s) \in\\ DifferentialControlSet(s)\end{matrix}, s \in (time_0, \tau]\right\} \quad (3.7)$$

Reach set under constraints are usable to define state constrained systems in terms of dynamics and technical capabilities.

Definition 4 (Reach set under state constraints)**.** *Suppose the initial position and time* $(state_0, time_0)$ *and state constraints are given* $state(t) \in \mathbb{A} \subset \mathbb{R}^n, \dot{x}(t) \in \mathbb{B} \subset \mathbb{R}^n$*. The reach set* $ReachSet[\tau, time_0, \vec{state_0}]$ *of* nonlinear UAS system *at time* $\tau \geq time_0$*, starting at position and time* $(state_0, time_0)$ *is given by:*

$$ReachSet\begin{bmatrix}\tau,\\ time_0,\\ state_0\end{bmatrix} = \bigcup\left\{state(\tau) : \begin{matrix}\forall s \in (time_0, \tau], state(s) \in \mathbb{A},\\ \dot{state}(s) \in \mathbb{B},\\ \exists input(s) \in Inputs(s)\end{matrix}\right\} \quad (3.8)$$

3.2.2 Computation of Reach Sets

Several techniques for reachability analysis of systems have been proposed. They can be (roughly) classified into two kinds:

1. Purely symbolic methods based on:
   a. the existence of analytic solutions of the differential equations and
   b. the representation of the state space in a decidable theory of the real numbers.
2. Methods that combine
   a. numeric integration of the differential equations
   b. symbolic representations of approximations of state space typically using (unions of) polyhedra or ellipsoids.

These techniques provide the algorithmic foundations for the tools that are available for computer-aided verification of hybrid systems ([12], [13], [14]).

The set-valued Lebesgue integral provides a conceptual tool for the direct computation of the reach set. In what follows we describe techniques from dynamic optimization which are used to compute reach sets for dynamic systems.

The relation between dynamic optimization and reachability was first observed in [15]. A typical problem of optimal control can be formulated as follows:

$$\max \left(\begin{array}{c} \int_{initialTime}^{finalTime} cost(time, state(time), contro(time)) \mathrm{d}time + \ldots \\ \cdots + FinalCost(state(finalTime)) \end{array} \right) \tag{3.9}$$

For nonlinear system:

$$\dot{state}(t) = f(t, state(t), control(t)), control(t) \in ControlSet(t) \subset \mathbb{R}^p \tag{3.10}$$

Where *cost* is given as cost function of time,state and input and $FinalCost$ represents cost functional. There are two main techniques to solve this problem:

1. The maximum principle
2. Dynamic programming. The maximum principle gives necessary conditions of optimality. Dynamic programming may be used to derive sufficientconditions of optimality.

A good reference on the maximum principle is [16]. A less known reference with detailed geometric interpretations is [17]. A good reference on dynamic programming is [18].

3.3 (R) Movement Automaton

Movement Automaton is basic interface approach for discretization of *trajectory evolution* or *control input* for any *continuous or discrete system model*.

Main function of *Movement Automaton is* for system given by equation $\dot{state} = f(time, state, input)$ with initial state $state_0$ to generate *reference trajectory* $\hat{state}(t)$ or *control signal* $input(t)$.

Using *Movement Automaton* as *Control Proxy* will provide us with *discrete command chain* interface. This will reduce the *non deterministic* element from *Evasive trajectory* generation, by reducing infinite maneuver set to finite *movement set*.

Non determinism of *Avoidance Maneuver* have been discussed as an issue in following works:

1. Newton gradient method for evasive car maneuvers [19].
2. Non-holistic methods for trajectory generation [20].
3. Stochastic approach to elliptic trajectories generation [21].

Examples of *Movement Automaton Implementation* as *Control Element* can be mentioned as follows:

1. Control of traffic flow [22].
2. Complex air traffic collision situation resolution system [23, 24].
3. SAA/DAA capable avoidance system [25].

3.3.1 Hybrid Automaton

First the notion of *hybrid* automaton [12, 26, 27] needs to be introduced:

Definition 5. *Hybrid automaton (3.11) is given as structure:*

$$HybridAutomaton(States, SystemState, VectorField, DiscreteTransition, ResetMap) \tag{3.11}$$

States *(Q) is given as set of discrete states, for every time* $t \in$ *Domain hybrid automaton stays in exactly one of* states.

SystemState *(x), is given in domain* $x \in \mathbb{R}^n, n \in \mathbb{N}^+$, *representing the trajectory evolution.*

VectorField *(f) (3.12) is bounded to single State* $\in$ *States and represents local SystemState evolution, when given automaton State is Active.*

$$VectorField : State \times SystemState \to SystemState \tag{3.12}$$

DiscreteTransition *(*φ*) (eq. 3.13) indicates changes of states in automaton, the changes are triggered by satisfying specific condition given by State and SystemState.*

$$DiscreteTransition : State \times SystemState \to State \tag{3.13}$$

ResetMap *(*ρ*) (eq. 3.14) defines changes of State to some default value, this change is triggered by specific automaton State and SystemState.*

$$ResetMap : State \times SystemState \to SystemState \tag{3.14}$$

3.3.2 Specialization of Hybrid Automaton

Definition 6. *Movement Primitive:*
States *from* Hybrid automaton *can be taken as* Movements *in* Movement Automaton. MovementPrimitive *(eq. 3.15) is describing the* Movement *behaviour as transfer function* VectorField *enriched with parameters.*

$$\begin{aligned} &MovementPrimitive(vectorField, minimalDuration, parameters) \\ &VectorField : SystemState \times parameters \rightarrow SystemState \end{aligned} \tag{3.15}$$

Example: Let say that *UAS* system is given as $\dot{position} = velocity$, then let us have two *MovementPrimitives*:

1. *Stay* - $minimalTime = 1s$, $parameters = \{\}$, $VectorField : \dot{position} = 0$.
2. *Move* - $minimalTime = 1s$, $parameters = \{velocity\}$, $VectorField : \dot{position} = velocity$.

Trajectory from Movement Primitives: The *UAS* should *Move* for $5s$ with velocity $10m/s$, then *Stay* for $10s$, then move for $7s$ with velocity $4m/s$, with initial position $position_0 = 0$ and initial time $t_0 = 1$ The standard approach is to derive transfer function $position = \Theta(\dots)$

$$position(t) = \Theta(\dots) \begin{cases} t \in [0, 5] & : 10 \times t + position(0) \\ t \in (5, 15] & : 0 \times (t - 5) + position(5) \\ t \in (15, 22] & : 4 \times (t - 15) + position(15) \end{cases} \tag{3.16}$$

The *example* given by (eq. 3.16) is fairly primitive, but imagine UAS system given by nonlinear dynamics $\dot{x} = f(x, u, t)$ [28]. Then defining transfer function for given command chain can be impossible.

Definition 7. *Movement Transition:*
System state *can be different than intended movement application, the notion of* Transition *is therefore introduced as stabilizing element in movement chaining (eq. 3.17).*

$$Transition : MovementPrimitive \times SystemState \rightarrow MovementPrimitive \tag{3.17}$$

Trajectory with Transitions: Introducing two transitions $Transition(Move, Stay)$ and $Transition(Stay, Move)$ reflecting periods when vehicle stop moving or speed-up to desired velocity. The transfer function (eq. 3.16) can be rewritten as combination of *MovementPrimitives* (eq. 3.15) and *Transitions* (eq. 3.17):

$$\begin{aligned} &Transition(Stay, Move), Move(5s, 10m/s), \\ &\qquad Transition(Move, Stay), Stay(10s), \\ &\qquad\qquad Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{3.18}$$

.

Note. There are two types of *MovementPrimitives*:

1. *Stationary* - when system state is considered neutral and they are considered as entry point for automaton.
2. *Dynamic* - when the system state is considered evolving and they needs to be terminated with *stationary* transition.

Movement Mapping Example: Transition/MovementPrimitive pairs (eq. 3.17) can be mapped into movements (eq. 3.19).

$$\begin{aligned} Move(5s, 10m/s) : Transition(Stay, Move), Move(5s, 10m/s), \\ Stay(10s) : Transition(Move, Stay), Stay(10s), \\ Move(7s, 4m/s) : Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{3.19}$$

Definition 8. *Movement:*
Movement can consist from multiple Transitions *(eq. 3.17) and one* MovementPrimitive *(eq. 3.15), the duration of* MovementPrimitive *can be shortened by* Transitions *duration.* Movement *is defined as follows:*

$$Movement \begin{pmatrix} initialState, \\ initialTime[0..1], \\ duration, \\ parameters[0..1] \end{pmatrix} = Chain \left(\begin{array}{c} InitialTransition(\dots)[0..*], \\ MovementPrimitive \begin{pmatrix} transitionState, \\ remainingDuration, \\ parameters \end{pmatrix} \\ LeaveTransition(\dots)[0..*], \end{array} \right) \tag{3.20}$$

Chain function *connects multiple* initial Transitions *which are appliead at* initialState *at* initialTime. *Then own* MovementPrimitive *(eq. 3.15) is invoked with* transitionnsState. Transitions state *is state changed by* Initial Transitions. *After* Movement Primitive *there can be* Leave Transitions Movement

Minimal Movement Time: Given by (eq. 3.21) for *movement* is given as sum of *MovementPrimitive* (eq. 3.15) minimal time, and *Transition* (eq. 3.17) in/out combined minimal time.

$$\begin{aligned} minimalTime(Movement) = & minimalTime(MovementPrimitive) + \\ & \max_{in/out} \{time(Transition)\} \end{aligned} \tag{3.21}$$

Movement Chaining: *Movements* can be *chained* and applied to initial *system state* to generate *system trajectory*. Example of trajectory is given by (eq. 3.16). Movements are reversibly obtained by participation such *trajectory* into *Movement primitives* and *Transitions*. Then sample *Trajectory* for $n \in \mathbb{N}^+$ movements looks like (eq. 3.22).

$$\begin{aligned}
&Trajectory(t_0) = State(t_0)\\
&Trajectory(t_0, t_1] = Movement_1(Trajectory(t_0), t_0, duration_1, parameters_1)\\
&Trajectory(t_1, t_2] = Movement_2(Trajectory(t_1), t_1, duration_2, parameters_2)\\
&Trajectory(t_2, t_3] = Movement_3(Trajectory(t_2), t_2, duration_3, parameters_3)\\
&\vdots\\
&Trajectory(t_{n-1}, t_n] = Movement_n(Trajectory(t_{n-1}), t_{n-1}, duration_n, parameters_n)
\end{aligned} \tag{3.22}$$

Given *Trajectory* at time t_0 is given as initial *State* of *System*. For time interval (t_0, t_1), which length is equal to $duration_1$, the *State* is given by $Movement_1$ with $parameters_1$ and base time t_0. This behaviour continues for movements $2, \ldots, n$.

Definition 9. *Movement Buffer:*
Movements *can be chained into* Buffer *with assumption of* continuous movement execution. Continuous movement executions *each movement in chain (eq. 3.22) is executed in time interval* $\tau_i = (t_{i-1}, t_i]$ *where* i *is movement order and* $\forall\ Movement_i$ *starting time is* t_0 *or* t_{i-1} *from previous movement. With given assumption* Buffer *is given as (eq. 3.23) with parameters* t_{i-1}, t_i *omitted, due* t_0 *and* $duration_i$ *dependency.*

$$Buffer = \{Movement_i(duration_i, parameters_i)\}\, i \in \mathbb{N}^+ \tag{3.23}$$

Definition 10. *Movement Automaton Trajectory:*
Let say system State$\in \mathbb{R}^n$ *which* Trajectory *is defined by movement chaining (eq. 3.22), applied on some* initial time $t_0 \in \mathbb{R}^+$ *and final time* $t_f = t_0 + \sum_{i=1}^{I} duration_i$*, with movements contained in* Buffer *(eq. 3.23) is given as* Trajectory *(eq. 3.24).*

$$Trajectory(t_0, State(t_0), Buffer)\ or\ Trajectory(State_0, Buffer)\ if\ t_0 = 0 \tag{3.24}$$

Note. The space dimension of *Trajectories* is $\mathbb{R}^{n+1}$ if the space dimension of state *Space* is R^n, because *Trajectory space* contains evolution of *Space* in time interval $T[t_0, t_f]$.

The transformation from *transfer function* (eq. 3.16) to *trajectory* (eq. 3.24) is natural, only set of *Movement primitives* (eq. 3.15) and set of *Transitions* (eq. 3.17) is required.

State Projection: *Trajectory* (eq. 3.24)is naturally evolution of space over time, then there exists *StateProjection* function (eq. 3.25) which returns *State* for specific *Time*.

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{3.25}$$

3.3.3 Definition

Definition 11. *Movement Automaton is given as follow:*

$$InitialState :\in \mathbb{R}^h, h \in \mathbb{N}^+ \tag{3.26}$$

$$System : \dot{State} = f(Time, State, Input) \text{ or } vectorField \tag{3.27}$$

$$Primitives = \left\{ MovementPrimitive_i \begin{pmatrix} vectorField, \\ minimalDuration, \\ parameters \end{pmatrix} \right\} i \in \mathbb{N}^+ \tag{3.28}$$

$$Transitions = \left\{ Transition_j \begin{pmatrix} MovementPrimitive_l, \\ MovementPrimitive_k \end{pmatrix}_{k \neq l} \right\} j \in N^+ \tag{3.29}$$

$$Movements = \left\{ Movement_m \begin{bmatrix} Transition_o[0..*], \\ MovementPrimitive_p \\ Transition_r[0..*], \end{bmatrix}_{o \neq r} \right\} m \in N^+ \tag{3.30}$$

$$Buffer = \{Movement_s(duration_s, parameters_s)\}\, s \in \mathbb{N}^+ \tag{3.31}$$

$$Executed = \{Movement_s(duration_s, parameters_t)\}\, t \in \mathbb{N}^+ \tag{3.32}$$

$$Builder : Movement \times MovementPrimitive \rightarrow Movement \tag{3.33}$$

$$Trajectory : InitialState \times Movement^u \rightarrow State \times Time, u \in N^+ \tag{3.34}$$

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{3.35}$$

System *(eq. 3.27) is given in form of* differential equations $\dot{x} = f(t, x, u)$ *or* other transformable equivalent*, with* initial state *(eq. 3.26).*

Movements *(eq. 3.22) are defined as sequence of necessary* initial transitions *(eq. 3.29),* movement primitive *(eq. 3.28), and,* leave transitions *(3.29).*

Buffer *contains a set of* movement primitives *(eq. 3.28) to be executed in order to achieve desired goal.* Builder *(eq. 3.33) assures that first* movement primitive *(eq. 3.15)from* Buffer *(eq. 3.31) is transformed into* next movement *(eq. 3.30) based on* current movement *(eq.3.30).*

System trajectory *(eq. 3.34) is defined in (eq. 3.24).* State projection *(eqs. 3.25,3.35) is giving* State *variable for time* $t \in [t_0, t_{max}]$ *where* t_max *is given by:*

$$t_{max} = t_0 + \sum_{i=1,u} Buffer.Movement(i).movementDuration \tag{3.36}$$

Note. From Continuous Reach set to Movement Automaton Control Reach Set:

The reach set R (3.37) for system ∂/∂t state $= model(state, input)$ with initial state $state_0 = state(t_i)$ in time interval $[t_i, t_{i+1}[$ is with existing control strategy $input(t) \in ControlStrategy(t)$. The reach set $R(state_0, t_0, t_1)$ where $t_1 > t_0$.

$$R(state_0, t_0, t_1) = \bigcup \{state(s) : input(s) \in ControlStrategy(s), s \in (t_0, t_1]\} \tag{3.37}$$

The reach set $\mathcal{R}$ (3.38) of the system under the control of the *movement automation* consist from the set of trajectories $Trajectory(initialState, buffer)$, which are executed in constrained time period $[t_i, t_{i+1}[$.

$$ReachSet(state_0, t_i, t_{i+1}) = \{Trajectory(state_0, buffer) : \text{duration}(buffer) \leq (t_{i+1} - t_i)\} \tag{3.38}$$

Note. Weak Invariance:

When the UAS is under the control of the movement automaton for the obstacle avoidance problem, by design of the avoidance algorithm, the trajectories of the UAV will not intersect any threat. This means that the controlled system ∂/∂t state $= model(state, input)$ is *weakly invariant* with respect to the complement of the threats, and with respect to the free space. A pair $(state, SafeSpace)$, where ∂/∂t state $= model(state, input)$ and $SafeSpace$ is a closed set, is weakly invariant if there exist controls such that a trajectory starting inside $State_0 \in SafeSpace$ remains inside $State(t) \in SafeSpace$ [29].

3.4 (R) LiDAR

LiDAR(Light Detection And Ranging) is active form of remote sensing: information is obtained from a signal which is sent from a transmitter and reflected by a target, and detected by a receiver back at the source. Following types of information can be obtained:

1. *Range to target* - topographic LiDAR or laser altimeter.
2. *Chemical properties of target* - differential absorption LiDAR.
3. *Velocity of target* - Doppler LiDAR.

Chemical properties of target are out of scope. Velocity of target seems as interesting property to investigate, but this type of LiDAR is usually used for meteorological measurements of wind currents [30]. Extended research in LiDAR as obstacle detection sensor has been executed by research group around Sabatini [4] and Ramasy [31].

LiDAR output is represented as point cloud it is described by following definition.

Definition 12 (Scanned point and Point-cloud)**.** *Consider viewpoint as origin of* $\mathbb{R}^3$ *space, Let point* $\in PolarCoordinates$ *be defined as:*

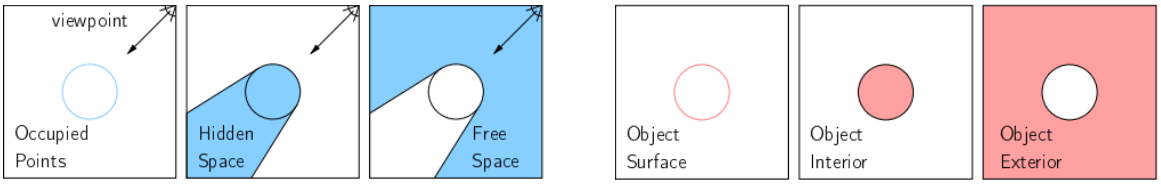
$$point = [distance, horizontal^\circ, vertical^\circ, time]^T \tag{3.39}$$

Where $horizontal^\circ$ *is horizontal angle from origin,* $vertical^\circ$ *is vertical angle to origin, and, time is time of retrieval.*

Point-cloud is set of points scanned in small enough time-frame, based on processing raw point data it can have following representations:

1. *Local point-cloud - position of sensor is used as origin of space and points can be represented in orthogonal or planar representation.*
2. *Global point-cloud -global position of sensor is used as reference to calculate global position of points.*

Point-cloud is usually addressed as *raw point-cloud* in case if its represented in Local planar coordinates. Other forms of point cloud require further processing and they are not feasible for real-time obstacle detection and avoidance [32].



(a) Space type definitions (b) Object properties definitions

Figure 3.1: Six space classifications [33].

Because of real-time obstacle avoidance it is necessary to introduce following terminology:

1. *Occupied points* - points which have been detected by LiDAR (also addressed as visible points).
2. *Hidden space* - space which is hidden behind occupied points, from viewpoint it is uncertain what is in that space.
3. *Free space* - space which is visible from viewpoint and it is not occupied by known objects.
4. *Object surface* - detected and undetected object surface
5. *Object interior* - occupied space by object.
6. *Object exterior* - free space around known objects.

Existing method for space segregation [33] leads to following definition:

Definition 13 (Accessible space)**.** *Consider known space as space explored by sensor (it can have different viewpoint along previous 3D trajectory). Intersection between object exterior (Exterior) and free space Free gives us Accessible space (Accessible).*

$$Accessible = Exterior(object) \cap Free(object) \tag{3.40}$$

Accessible space S_A (def. 13) is our bordering limitation for reachable space of system $ReachSet[\tau, time_0, state_0]$ (def. 1.).

3.5 (R) Complementary Definitions

Cartesian Space: 3D Cartesian space defined by an X, Y, and Z axes (describing position based on horizontal placement, vertical placement, and depth respectively). The coordinates for any point within this space are shown as a vector $[x, y, z]$. *Coordinate system* used this work is the right-handed system (thumbs points at positive direction of x-axis, index finger is pointing to positive direction of y-axis, the positive of z axis given by remaining fingers).

Base Works: *Euler* outlined *universal rotation theorem* which was presented in [34]. Rigid body dynamics and rotation matrices defined by Schaub [35].

UAS Coordinate System: *Local Coordinate frame* is defined by UAS mass center as space center, $Z-$ in direction of gravitational force, $X+$ in direction of UAS heading. This local coordinate system is called Euler Normalized Unit-frame (ENU).

Rotation Matrices: Following *Rotation Matrices* are used to transform between two displaced coordinate systems. Roll angle rotation is defined around X-axis by matrix (eq. 3.41) on YZ-plane. Pitch angle rotation matrix is defined around Y-axis by matrix (eq. 3.42) on XZ-plane. Yaw angle rotation matrix is defined around Z axis by matrix (eq. 3.43) on XY-plane.

$$R_{YZ} = R_{roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & \sin(roll) & \cos(roll) \end{bmatrix} \tag{3.41}$$

$$R_{XZ} = R_{pitch} = \begin{bmatrix} \cos(pitch) & 0 & \sin(pitch) \\ 0 & 1 & 0 \\ -\sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \tag{3.42}$$

$$R_{XY} = R_{yaw} = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.43}$$

The full rotation matrix in X,Y,Z is given by (eq. 3.44).

$$R_{XYZ} = R_{roll,pitch,yaw} = R_{XY} * R_{XZ} * R_{YZ} = R_{yaw} * R_{pitch} * R_{roll} \tag{3.44}$$

Note. The rotation matrix R_{XYZ} (eq. 3.44) and its inverse R^{-1}_{XYZ} which gives identity $R_{XYZ} \times R^{-1}_{XYZ} = I$ are used all over this work in transformation.

Gimbal Lock Prevention: To keep solution numerically stable and rotations numerically stable gimbal lock prevention is necessary [36]. Gimbal lock occurs when one of matrices (eq. 3.41, 3.42, 3.43) is singular or final matrix for X,Y,Z rotation is singular (eq. 3.44). Gimbal lock leads to loose of one or more degree of freedom, depending on rank and space dimension of singular matrix. To prevent gimbal lock it is necessary to introduce mechanism to check if rotation matrix is regular. For this purpose normative reset function is introduced:

$$[roll, pitch, yaw]^T = f(t, roll^-, pitch^-, yaw^-), \quad \text{norm}(R_{roll,pitch,yaw}) = 3 \tag{3.45}$$

Function resets yaw or roll angle to initial position to keep degree of rotation matrix. Simpler but not fault tolerant solution is to keep angles $roll, pitch, yaw \in (-\pi, \pi]$ range.

Polar coordinates: A *polar coordinate system* represents point in form of vector:

$$point_{polar} = [distance, horizontalDislocationAngle, verticalDislocationAngle]^T$$

which is ideal for representation of LiDAR scanned point, because usually total point distance and pair of dislocation angles are returned. Using most common LiDAR with horizontal rotation $horizontal^\circ$ and vertical mirror inclination $vertical^\circ$, one can define polar coordinate $point_{polar} = [distance_{x,y,z}, horizontal^\circ, vertical^\circ]$ which is dual to Cartesian coordinate $point_{cartesian} = [x, y, z]$. If rotation angle ranges are $horizontal^\circ, vertical^\circ \in (-\pi, \pi]$ transformation function is bijection.

Polar → Cartesian: Transformation from polar to Cartesian representation is defined by following series of functions (eq. 3.46, 3.47,3.48, 3.49).

$$distance_{xy} = \cos(horizontal^{\circ}) \times distance_{xyz} \tag{3.46}$$

$$z = \sin(hirizontal^{\circ}) \times distance_{xyz} \tag{3.47}$$

$$y = \sin(horizontal^{\circ}) \times distance_{xy} \tag{3.48}$$

$$x = \cos(vertical^{\circ}) \times distance_{xy} \tag{3.49}$$

Cartesian → Polar: Transformation from Cartesian to polar representation is defined by following series of functions (eq 3.50, 3.51,3.52, 3.53).

$$distance_{xyz} = \sqrt{x^2 + y^2 + z^2} \tag{3.50}$$

$$distance_{xy} = \sqrt{x^2 + y^2} \tag{3.51}$$

$$horizontal^{\circ} = \arctan\left(\frac{y}{x}\right) \tag{3.52}$$

$$vertical^{\circ} = \arctan\left(\frac{z}{d_{xy}}\right) \tag{3.53}$$

Definition 14. *Global Coordinate System (GCS) $\mathcal{X}_{\mathcal{G}}$ takes as center $c_{\mathcal{G}0}$ well known point (for example center of geo-reference model in GNSS systems) every reference distance, plane or angle is calculated taking this center to mind.*

Definition 15. *Local Coordinate System (LCS) $\mathcal{X}_{\mathcal{L}}$ takes as center $c_{\mathcal{L}0}$ frame of vehicle and can be changing position and orientation in global coordinate frame $\mathcal{X}_{\mathcal{G}}$.*

Definition 16. *Global position of polar obstacle $o_i \in \mathcal{O}_{3D}$. Let $o_i = [d_o, \theta_o, \varphi_o]^T$ be polar position of obstacle o_i in local coordinate frame of vehicle with global Cartesian position $[x_{,v}, y_v, z_v]^T$ and normalized orientation angles $[roll_v, pitch_v, yaw_v]$.*
Then Cartesian position of obstacle oi, $[x_o, y_o, z_o]^T$ in local coordinate frame is given by transformation functions x_o (eq. 3.49), y_o (eq. 3.48), z_o (eq. 3.47).
Global position of polar obstacle oi, $[x_g, y_g, z_g]^T$ is given by following equation:

$$\begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} = \begin{bmatrix} R_{XYZ}(roll_v, pitch_v, yaw_v) \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} \end{bmatrix} \tag{3.54}$$

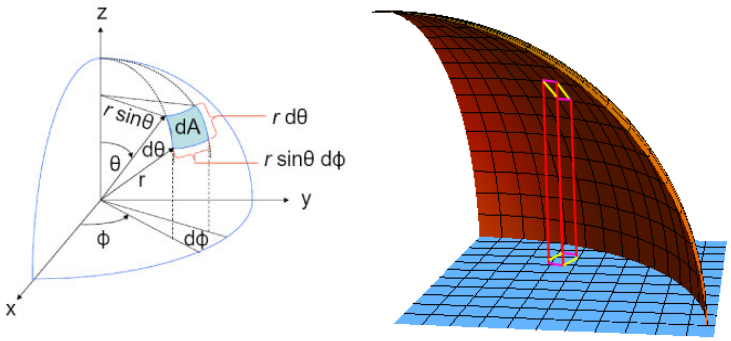
Definition 17. *Local position of global coordinate $[x_g, y_g, z_g]^T \in \mathbb{R}^3$. Let there be vehicle with global Cartesian position $[x_{,v}, y_v, z_v]^T$ and normalized orientation angles $[roll_v$, $pitch_v$, $yaw_v]$. in global coordinate frame $\mathcal{X}_{\mathcal{X}}$.*

Then local Cartesian coordinate position $[x_l, y_l, z_l]^T$ *of point* $[x_g, y_g, z_g]^T$ *is given by following equation:*

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \left[R_{XYZ}(-roll_v, -pitch_v, -yaw_v) \left(\begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} - \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} \right) \right] \tag{3.55}$$

Local polar position is given as $[distance_l, horizontal_l^\circ, horizontal_l^\circ]$*, where* $distance_l$ *is given by (eq. 3.50),* $vertical_l^\circ$ *is given by (eq. 3.52).* $vertical_l^\circ$ *is given by (eq. 3.53), where* $[x_l, y_l, z_l]$ *are used as local coordinates.*

Polar surface calculation: The problem is to calculate intersected surface dA of ball subsurface defined by radius r, horizontal span ϕ and vertical span θ. From classical mechanics one can formulate problem as given by (fig. 3.2a). The intersection plot is in (fig. 3.2b).



(a) Notation (b) Plot

Figure 3.2: Polar surface calculation notation and plot

One can use first fundamental form to determine the surface area element. Recall that this is the metric tensor, whose components are obtained by taking the inner product of two tangent vectors in polar space $g_{i,j} = X_i \cdot X_j$, for tangent vectors X_i, X_j. Following identification for the components of metric tensor will be used:

$$g_{ij} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} \tag{3.56}$$

Where $E =< X_u, X_u >$, $F =< X_u, X_v >$, and $G =< X_v, X_v >$. Lagrange's identity can be used , which tells us that the squared area of a parallelogram in space is equal to the sum of the squares of its projections onto the Cartesian plane:

$$|X_u \times X_v|^2 = |X_u|^2 |X_v|^2 - (X_u \cdot X_v)^2 \tag{3.57}$$

Given example is displayed in (fig. 3.2b). The area element is given as:

$$\begin{aligned} \mathrm{d}A &= |X_u \times X_v| \quad \mathrm{d}u\mathrm{d}v \\ &= \sqrt{\left| |X_u|^2 |X_v|^2 - (X_u \cdot X_v)^2 \right|} \quad \mathrm{d}u\mathrm{d}v \\ &= \sqrt{EG - F^2} \quad \mathrm{d}u\mathrm{d}v \end{aligned} \tag{3.58}$$

We will find tangent vectors via the usual parametrization which give, $X(\phi, \theta) = [r\cos\phi\sin\theta,\, r\sin\phi\sin\theta,\, r\cos\theta]$, so that tangent vectors are simply defined as:

$$\begin{aligned} X_\phi &= [-r\sin\phi\cos\theta, r\cos\phi\sin\theta, 0] \\ X_\theta &= [-r\cos\phi\sin\theta, r\sin\phi\cos\theta, -r\sin\theta] \end{aligned} \tag{3.59}$$

Computing the elements of the first fundamental form gives us:

$$E = r^2\cos^2\theta, \quad F = 0, \quad G = r^2 \tag{3.60}$$

Thus final difference is given as:

$$\mathrm{d}A = \sqrt{r^4\cos^2} \quad \mathrm{d}\theta\mathrm{d}\phi = r^2\cos\theta \quad \mathrm{d}\theta\mathrm{d}\phi \tag{3.61}$$

Note. Polar Surface is used in *Detected Obstacle Rating Calculation* (sec. **??**). The final formula used in *surface integral calculation* in *non-compact notation* is given as follow:

$$\mathrm{d}A = r^2\cos(horizontal^\circ) \quad \mathrm{d}horizontal^\circ\mathrm{d}vertical^\circ \tag{3.62}$$

Bibliography

[1] Thomas P Spriesterbach, Kelly A Bruns, Lauren I Baron, and Jason E Sohlke. Unmanned aircraft system airspace integration in the national airspace using a ground-based sense and avoid system. *Johns Hopkins APL Technical Digest*, 32(3):572–583, 2013.

[2] Adrian Muraru. A critical analysis of sense and avoid technologies for modern uavs. In *Mechanical, Industrial, and Manufacturing Engineering–Proceedings of 2011 International Conference on Mechanical, Industrial, and Manufacturing Engineering (MIME 2011)*, 2011.

[3] John Lai, Jason J Ford, Luis Mejias, Peter O'Shea, and Rod Walker. See and avoid using onboard computer vision. *Sense and Avoid in UAS Research and Applications, Plamen Angelov (ed.), John Wiley and Sons, West Sussex, UK*, 2012.

[4] Roberto Sabatini, Alessandro Gardi, and Mark A Richardson. Lidar obstacle warning and avoidance system for unmanned aircraft. *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, 8(4):702–713, 2014.

[5] Mykel J Kochenderfer, Leo P Espindle, J Daniel Griffith, and James K Kuchar. Encounter modeling for sense and avoid development. In *2008 Integrated Communications, Navigation and Surveillance Conference*, pages 1–10. IEEE, 2008.

[6] JARUS regulations. `http://jarus-rpas.org/regulations`. Accessed: 2018-10-28.

[7] Edward A Lee. *Structure and interpretation of signals and systems*. Lee & Seshia, 2011.

[8] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. Wiley-Interscience, 1987.

[9] Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

[10] Nikolai Nikolaevich Krasovskij, Andrei Izmailovich Subbotin, and Samuel Kotz. *Game-theoretical control problems*. Springer-Verlag New York, Inc., 1987.

[11] NN Krasovskii and AI Subbotin. Game-theoretical control problems. translated from the russian by samuel kotz, 1988.

[12] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool kronos. In *Hybrid Systems III*, pages 208–219. Springer, 1996.

[13] Thomas A Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and computation*, 111(2):193–244, 1994.

[14] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: the next generation. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 56–65. IEEE, 1995.

[15] G Leitmann. Optimality and reachability via feedback controls. *Dynamic systems and mycrophysics*, pages 119–141, 1982.

[16] LS Pontryagin, VG Boltyanskii, and RV Gamkrelidze. Ef mischenko the mathematical theory of optimal processes (english translation by k. n. trirogoff). *Interscience, New York*, 1962.

[17] Igor Vladimirovich Girsanov. *Lectures on mathematical theory of extremum problems*, volume 67. Springer Science & Business Media, 2012.

[18] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[19] Ondřej Šantin and Vladimir Havlena. Combined partial conjugate gradient and gradient projection solver for mpc. In *Control Applications (CCA), 2011 IEEE International Conference on*, pages 1270–1275. IEEE, 2011.

[20] Frangois G Pin and Hubert A Vasseur. Autonomous trajectory generation for mobile robots with non-holonomic and steering angle constraints. Technical report, Oak Ridge National Lab., 1990.

[21] Ralph G Andrzejak, G Widman, K Lehnertz, C Rieke, P David, and CE Elger. The epileptic process as nonlinear deterministic dynamics in a stochastic environment: an evaluation on mesial temporal lobe epilepsy. *Epilepsy research*, 44(2-3):129–140, 2001.

[22] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

[23] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001.

[24] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Trajectory tracking control design for autonomous helicopters using a backstepping algorithm. In *American Control Conference, 2000. Proceedings of the 2000*, volume 6, pages 4102–4107. IEEE, 2000.

[25] Alojz Gomola, João Borges de Sousa, Fernando Lobo Pereira, and Pavel Klang. Obstacle avoidance framework based on reach sets. In *Iberian Robotics conference*, pages 768–779. Springer, 2017.

[26] Mircea Lazar. Model predictive control of hybrid systems: Stability and robustness, 2006.

[27] Francesco Borrelli, Alberto Bemporad, Michael Fodor, and Davor Hrovat. An mpc/hybrid system approach to traction control. *IEEE Transactions on Control Systems Technology*, 14(3):541–552, 2006.

[28] Thor I Fossen. Mathematical models for control of aircraft and satellites. *Department of Engineering Cybernetics Norwegian University of Science and Technology*, 2011.

[29] Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.

[30] S Martin, J Bange, and F Beyrich. Meteorological profiling of the lower troposphere using the research uav" m 2 av carolo". *Atmospheric Measurement Techniques*, 4(4):705–716, 2011.

[31] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Jing Liu. Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55:344–358, 2016.

[32] Qi Chen. Airborne lidar data processing and information extraction. *Photogrammetric engineering and remote sensing*, 73(2):109, 2007.

[33] Theodore C Yapo, Charles V Stewart, and Richard J Radke. A probabilistic representation of lidar range data for efficient 3d object detection. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.

[34] Leonhard Euler. Formulae generales pro translatione quacunque corporum rigidorum. *Novi Acad. Sci. Petrop*, 20:189–207, 1775.

[35] Hanspeter Schaub and John L Junkins. *Analytical mechanics of space systems*. Aiaa, 2003.

[36] Manuel Kramer and Douglas J Dapprich. Gyro stabilized inertial reference system with gimbal lock prevention means, October 4 1977. US Patent 4,052,654.