6.2 UAS Model and Control

The key feature of *Movement Automaton* is to interface *continuous-control signal* as the *discrete command chain*. Following topics are introduced in this section:

1. *Movement Automaton Background* (sec. 6.2.1) - the listing of related work and similar approaches to ours.

2. *Specialization of Hybrid Automaton* (sec. 6.2.2) - the specialization of the hybrid automaton to fulfill control/approximation roles in our approach.

3. *Formal Movement Automaton Definition* (sec. 6.2.3) - the formal definition of *movement automaton* used in our approach.

4. *Used UAS Nonlinear Model* (sec. 6.2.4) - simple plane model used in this work as *controlled plant*.

5. *Used Movement Automaton* (sec. 6.2.5) - movement automaton for *UAS Nonlinear Model* constructed from scratch.

6. *Segmented Movement Automaton* (sec. 6.2.6) - for more complex systems the *State Space* can be *separated into Segments* and *segment movement automaton* is used to generate *thick reference trajectory*.

7. *Reference Trajectory Generator* (sec. 6.2.7) - other use of *Movement Automaton* as predictor for *reference trajectory calculation*.

6.2.1 Movement Automaton Background

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 [1].

2. Non-holistic methods for trajectory generation [2].

3. Stochastic approach to elliptic trajectories generation [3].

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

1. Control of traffic flow [4].
2. Complex air traffic collision situation resolution system [5, 6].
3. SAA/DAA capable avoidance system [7].

6.2.2 Specialization of Hybrid Automaton

Idea: There is a need for *fast trajectory approximation* method. The basic idea is taken from pilot steering an plane. The pilot is issued a commands from an navigator in very short and precise manner. The movement has its primitive phase when steering is static and its transition phase when steering is moving from one static position to another.

Imagine having vertical and horizontal flaps on airplane (fig. 6.1). The *navigator* is issuing a command every second to a pilot. Each command is translated by hands of the pilot to an input signal (blue line). The command validity period (black frame) is split into *transition period* (red frame) when input signal is changing and primitive period (magenta frame) when the position of input signal is static.

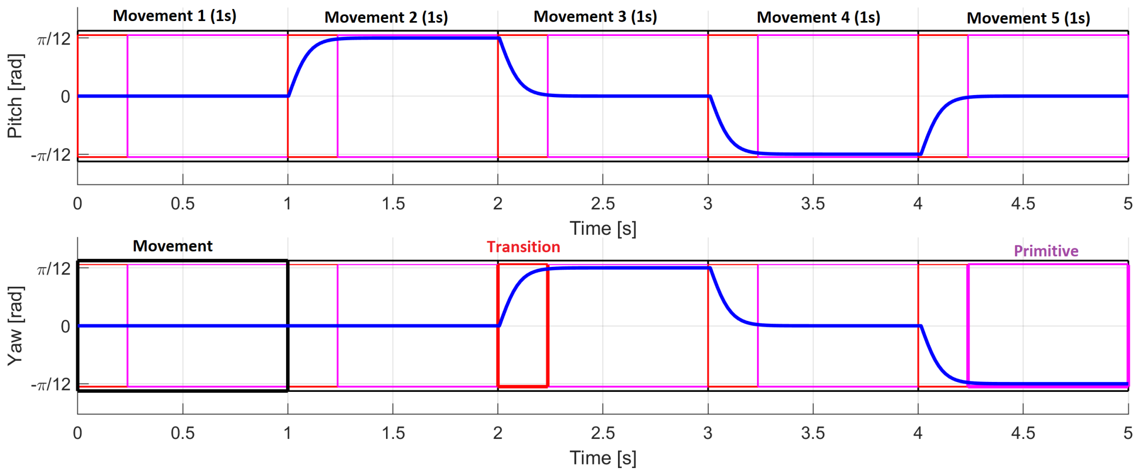

Figure 6.1: Example of input signal segmentation to movements.

Note. The hybrid automaton (sec. **??**) can be used as the base for simple control mechanism imitating navigators command execution by pilot. The automaton states can be mapped to primitives and transitions. The reset map needs to be replaced with external order issuer to ensure smooth execution of commands.

The future commands will be stacked in the buffer from which they will be picked for an execution.

Definition 1. *Movement Primitive:*
States *from* Hybrid automaton *can be taken as* Movements *in* Movement Automaton. MovementPrimitive *(eq. 6.1) is describing the* Movement *behaviour as transfer function* VectorField *enriched with parameters.*

$$\begin{aligned} &MovementPrimitive(vectorField, minimalDuration, parameters) \\ &VectorField : SystemState \times parameters \rightarrow SystemState \end{aligned} \tag{6.1}$$

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 $position = \Theta(\dots)$

$$position(t) = \Theta(\dots) \begin{cases} t \in [0,5] & : 10 \times t + position(0) \\ t \in (5,15] & : 0 \times (t-5) + position(5) \\ t \in (15,22] & : 4 \times (t-15) + position(15) \end{cases} \tag{6.2}$$

The *example* given by (eq. 6.2) is fairly primitive, but imagine UAS system given by nonlinear dynamics [8]. Then defining transfer function for given command chain can be impossible.

Definition 2. *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. 6.3).*

$$Transition : MovementPrimitive \times SystemState \rightarrow MovementPrimitive \tag{6.3}$$

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. 6.2) can be rewritten as combination of *MovementPrimitives* (eq. 6.1) and *Transitions* (eq. 6.3):

$$\begin{aligned} &Transition(Stay, Move), Move(5s, 10m/s), \\ &\qquad Transition(Move, Stay), Stay(10s), \\ &\qquad\qquad Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{6.4}$$

.

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. 6.3) can be mapped into movements (eq. 6.5).

$$\begin{aligned} Move(5s, 10m/s) &: Transition(Stay, Move), Move(5s, 10m/s), \\ Stay(10s) &: Transition(Move, Stay), Stay(10s), \\ Move(7s, 4m/s) &: Transition(Stay, Move), Move(7s, 4m/s) \end{aligned} \tag{6.5}$$

Definition 3. *Movement:*
Movement can consist from multiple Transitions *(eq. 6.3) and one* MovementPrimitive *(eq. 6.1), the duration of* MovementPrimitive *can be shortened by* Transitions *duration.* Movement *is defined as follows:*

$$Movement \begin{pmatrix} initialState, \\ initialTime[0..1], \\ duration, \\ parameters[0..1] \end{pmatrix} = Chain \begin{pmatrix} InitialTransition(\dots)[0..*], \\ MovementPrimitive \begin{pmatrix} transitionState, \\ remainingDuration, \\ parameters \end{pmatrix} \\ LeaveTransition(\dots)[0..*], \end{pmatrix} \tag{6.6}$$

Chain function *connects multiple* initial Transitions *which are appliead at* initialState *at* initialTime*. Then own* MovementPrimitive *(eq. 6.1) 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. 6.7) for *movement* is given as sum of *MovementPrimitive* (eq. 6.1) minimal time, and *Transition* (eq. 6.3) in/out combined minimal time.

$$minimalTime(Movement) = \begin{matrix} minimalTime(MovementPrimitive) + \\ \max_{in/out}\{time(Transition)\} \end{matrix} \tag{6.7}$$

Movement Chaining: *Movements* can be *chained* and applied to initial *system state* to generate *system trajectory*. Example of trajectory is given by (eq. 6.2). 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. 6.8).

$$\begin{aligned}
&Trajectory(t_0) = State(t_0)\\
&Trajectory(t_0, t_1] = Movement_1(Trajectory(t_0), t_0, duration_1, parameters_1)\\
&Trajectory(t_1, t_2] = Movement_2(Trajectory(t_1), t_1, duration_2, parameters_2)\\
&Trajectory(t_2, t_3] = Movement_3(Trajectory(t_2), t_2, duration_3, parameters_3)\\
&\vdots\\
&Trajectory(t_{n-1}, t_n] = Movement_n(Trajectory(t_{n-1}), t_{n-1}, duration_n, parameters_n)
\end{aligned} \tag{6.8}$$

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 4. *Movement Buffer:*
Movements *can be chained into* Buffer *with assumption of* continuous movement execution. Continuous movement executions *each movement in chain (eq. 6.8) 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. 6.9) with parameters* t_{i-1}, t_i *omitted, due* t_0 *and* $duration_i$ *dependency.*

$$Buffer = \{Movement_i(duration_i, parameters_i)\}\ i \in \mathbb{N}^+ \tag{6.9}$$

Definition 5. *Movement Automaton Trajectory:*
Let say system State$\in \mathbb{R}^n$ *which* Trajectory *is defined by movement chaining (eq. 6.8), 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. 6.9) is given as* Trajectory *(eq. 6.10).*

$$Trajectory(t_0, State(t_0), Buffer)\ or\ Trajectory(State_0, Buffer)\ if\ t_0 = 0 \tag{6.10}$$

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. 6.2) to *trajectory* (eq. 6.10) is natural, only set of *Movement primitives* (eq. 6.1) and set of *Transitions* (eq. 6.3) is required.

State Projection: *Trajectory* (eq. 6.10)is naturally evolution of space over time, then there exists *StateProjection* function (eq. 6.11) which returns *State* for specific *Time.*

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{6.11}$$

6.2.3 Formal Movement Automaton Definition

Definition 6. *Movement Automaton is given as follow:*

$$InitialState :\in \mathbb{R}^h, h \in \mathbb{N}^+ \tag{6.12}$$

$$System : \dot{State} = f(Time, State, Input) \text{ } or \text{ } vectorField \tag{6.13}$$

$$Primitives = \left\{ MovementPrimitive_i \begin{pmatrix} vectorField, \\ minimalDuration, \\ parameters \end{pmatrix} \right\} i \in \mathbb{N}^+ \tag{6.14}$$

$$Transitions = \left\{ Transition_j \begin{pmatrix} MovementPrimitive_l, \\ MovementPrimitive_k \end{pmatrix}_{k \neq l} \right\} j \in N^+ \tag{6.15}$$

$$Movements = \left\{ Movement_m \begin{bmatrix} Transition_o[0..*], \\ MovementPrimitive_p \\ Transition_r[0..*], \end{bmatrix}_{o \neq r} \right\} m \in N^+ \tag{6.16}$$

$$Buffer = \{Movement_s(duration_s, parameters_s)\} \text{ } s \in \mathbb{N}^+ \tag{6.17}$$

$$Executed = \{Movement_s(duration_s, parameters_t)\} \text{ } t \in \mathbb{N}^+ \tag{6.18}$$

$$Builder : Movement \times MovementPrimitive \rightarrow Movement \tag{6.19}$$

$$Trajectory : InitialState \times Movement^u \rightarrow State \times Time, u \in N^+ \tag{6.20}$$

$$StateProjection : Trajectory \times Time \rightarrow State(Time) \tag{6.21}$$

System *(eq. 6.13) is given in form of* differential equations $\dot{x} = f(t, x, u)$ *or* other transformable equivalent*, with* initial state *(eq. 6.12).*

Movements *(eq. 6.8) are defined as sequence of necessary* initial transitions *(eq. 6.15),* movement primitive *(eq. 6.14), and,* leave transitions *(6.15).*

Buffer *contains a set of* movement primitives *(eq. 6.14) to be executed in order to achieve desired goal.* Builder *(eq. 6.19) assures that first* movement primitive *(eq. 6.1)from* Buffer *(eq. 6.17) is transformed into* next movement *(eq. 6.16) based on* current movement *(eq.6.16).*

The system trajectory *(eq. 6.20) is defined in (eq. 6.10).* State projection *(eqs. 6.11,6.21) is giving* State *variable for time $t \in [t_0, t_{max}]$ where t_max is given by:*

$$t_{max} = t_0 + \sum_{i=1,u} Buffer.Movement(i).movementDuration \tag{6.22}$$

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

The reach set R (6.23) for system d/dt 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 $t_1 > t_0$.

$$R(state_0, t_0, t_1) = \bigcup \{state(s) : input(s) \in ControlStrategy(s), s \in (t_0, t_1]\} \tag{6.23}$$

The reach set $\mathcal{R}$ (6.24) 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 $[t_i, t_{i+1}[$.

$$\begin{aligned} &ReachSet(state_0, t_i, t_{i+1}) = \\ &\qquad \{Trajectory(state_0, buffer) : \text{duration}(buffer) \le (t_{i+1} - t_i)\} \end{aligned} \tag{6.24}$$

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 d/dt 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 d/dt 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 $State(t) \in SafeSpace$ [9].

6.2.4 Used UAS Nonlinear Model

Motivation: Simplified rigid body kinematic model will be used. This model have decoupled roll, yaw and pitch angles. The focus is on *reach set approximation methods*, therefore *UAS model* is simplified.

State Vector (eq. 6.25) defined as positional state in euclidean position in right-hand euclidean space, where *x, y, z* can be abstracted as latitude, longitude, altitude.

$$state = [x, y, z, roll, pitch, yaw]^T \tag{6.25}$$

Input Vector (eq. 6.26) is defined as linear velocity of UAS v and angular speed of rigid body $\omega_{roll}, \omega_{pitch}, \omega_{yaw}$.

$$input = [v, \omega_{roll}, \omega_{pitch}, \omega_{yaw}]^T \tag{6.26}$$

Velocity distribution function (eq. 6.27) is is defined trough standard rotation matrix and linear velocity v, oriented velocity $[v_x,\ v_y,\ v_z]$ given by (eq. 6.28).

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v\cos(pitch)\cos(yaw) \\ v\cos(pitch)\sin(yaw) \\ -v\sin(pitch) \end{bmatrix} \tag{6.27}$$

UAS Nonlinear Model (eq. 6.28) is given by *first order equations:*

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}time} &= v\cos(pitch)\cos(yaw); & \frac{\mathrm{d}roll}{\mathrm{d}time} &= \omega_{roll}; \\ \frac{\mathrm{d}y}{\mathrm{d}time} &= v\cos(pitch)\sin(yaw); & \frac{\mathrm{d}pitch}{\mathrm{d}time} &= \omega_{pitch}; \\ \frac{\mathrm{d}z}{\mathrm{d}time} &= -v\sin(pitch); & \frac{\mathrm{d}yaw}{\mathrm{d}time} &= \omega_{yaw}; \end{aligned} \tag{6.28}$$

6.2.5 Used Movement Automaton for UAS Model

Motivation: An *UAS Nonlinear Model* (eq. 6.28) can be modeled by *Movement Automaton* (def. 6).

Movement Primitives by (def. 1) are given as (eq. 6.1). To define primitives the *minimal time* is $1s$. The *maximal duration* is also $1s$.

Assumption 1. *Let assume that* transition time *of* roll, pitch, yaw, linear velocity *is* $0s$.

Under the assumption (as. 1) the *movement transitions* (def. 2) have 0 duration.

Note. The assumption (as. 1) can be relaxed under condition that *path tracking controller exists.*

Movements (def. 3) for *fixed step k* we start with discretization of the input variables. The *linear velocity* in text step is given:

$$v(k+1) = v(k) + \delta v(k) \tag{6.29}$$

The *roll, pitch, yaw* for next step are given

$$\begin{aligned} roll(k+1) &= roll(k) + \delta roll(k) \\ pitch(k+1) &= pitch(k) + \delta pitch(k) \\ yaw(k+1) &= yaw(k) + \delta yaw(k) \end{aligned} \tag{6.30}$$

The $\delta v(k)$ is *velocity change*, $\delta roll(k)$, $\delta pitch(k)$, $\delta yaw(k)$, are *orientation changes* for current discrete step k. If the duration of *transition* is $0s$ (as. 1) then 3D trajectory evolution in discrete time is given as:

$$\begin{aligned} x(k+1) &= x(k) + v(k+1)\cos(pitch(k+1))\cos(yaw(k+1)) && = \delta x(k) \\ y(k+1) &= y(k) + v(k+1)\cos(pitch(k+1))\sin(yaw(k+1)) && = \delta y(k) \\ z(k+1) &= z(k) - v(k+1)\sin(pitch(k+1)) && = \delta z(k) \\ time(k+1) &= time(k) + 1 && = \delta time(k) \end{aligned} \tag{6.31}$$

The $\delta x(k)$, $\delta y(k)$, $\delta z(k)$ are positional differences depending on *input vector* for given discrete time k:

$$input(k) = \begin{bmatrix} \delta x(k), \delta y(k), \delta z(k), \delta v(k), \\ \delta roll(k), \delta pitch(k), \delta yaw(k), \delta time(k) \end{bmatrix}^T \tag{6.32}$$

The *state vector* for discrete time is given:

$$state(k) = \begin{bmatrix} x(k), y(k), z(k), v(k), \\ roll(k), pitch(k), yaw(k), time(k) \end{bmatrix}^T \tag{6.33}$$

The nonlinear model (eq. 6.28) is then reduced to *linear discrete model* (eq. 6.34) given by *apply movements* function (eq. 6.29, 6.30, 6.31).

$$state(k+1) = applyMovement(state(k), input(k)) \tag{6.34}$$

Movement Set for linear discrete model (eq. 6.34) is defined as set of extreme unitary movements on main axes (tab. 6.1) and diagonal axes (tab. 6.2).

$input(movement)$	Straight	Down	Up	Left	Right
$\delta x(k)[m]$	1.00	0.98	0.98	0.98	0.98
$\delta y(k)[m]$	0	0	0	0.13	-0.13
$\delta z(k)[m]$	0	-0.13	0.13	0	0
$\delta roll(k)[°]$	0	0	0	0	0
$\delta pitch(k)[°]$	0	15°	-15°	0	0
$\delta yaw(k)[°]$	0	0	0	15°	-15°

Table 6.1: Input values for main axes movements.

$input(movement)$	Down-Left	Down-Right	Up-Left	Up-Right
$\delta x(k)[m]$	0.76	0.76	0.76	0.76
$\delta y(k)[m]$	-0.13	0.13	0.13	-0.13
$\delta z(k)[m]$	-0.13	-0.13	0.13	0.13
$\delta roll(k)[°]$	0	0	0	0
$\delta pitch(k)[°]$	-15°	-15°	15°	15°
$\delta yaw(k)[°]$	15°	-15°	15°	-15°

Table 6.2: Input values for diagonal axes movements.

Note. Movement set in shorten form is given as

$$MovementSet = \left\{\begin{matrix} Straight, Left, Right, Up, Down, \\ DownLeft, DownRight, UpLeft, UpRight \end{matrix}\right\} \tag{6.35}$$

Trajectory by (def. 5) for initial time $time = 0$, initial state $state(0)$ and *Movement Buffer* (from def. 4):

$$Buffer \in MovementSet^{*}(eq.6.35), \quad |Buffer| \in \mathbb{N} \tag{6.36}$$

Trajectory (eq. 6.37) is then given as the time-series of discrete states:

$$Trajectory(state(0), Buffer) = \left\{\begin{matrix} state(0) + \sum_{j=0}^{i-1} input(movement(j)) : \\ i \in \{1 \ldots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{matrix}\right\} \tag{6.37}$$

Trajectory (eq. 6.37) is ordered set of states bounded to discrete time $0 \ldots n$, where n is member count of *Buffer*. Trajectory set has $n + 1$ members:

$$Trajectory(state(0), Buffer) = \\ \left\{\begin{matrix} state(0) = state(0) + \{\} \\ state(1) = state(0) + input(movement(1)) \\ state(2) = state(0) + input(movement(1)) + input(movement(2)) \\ \vdots = \vdots \\ state(n) = state(0) + input(movement(1)) + \cdots + input(movement(n)) \end{matrix}\right\} \tag{6.38}$$

State Projection (eq. 6.39) for the *Trajectory* (eq. 6.37) is given as follow:

$$StateProjection(Trajectory, time) = Trajectory.getMemberByIndex(time + 1) \tag{6.39}$$

Note. Movement Automaton for system (eq. 6.28) with given (as. 1) is established with all related properties (sec. 6).

6.2.6 Segmented Movement Automaton

Motivation: Constructing *Movement Automaton* for more complex system can be tedious. Used *Movement Automaton* for *UAS system* (6.28) has decoupled control which is not true for most of the copters/planes [8].

Partitioning UAS State Space: Proposed movement automaton is defined by its Movement set (tab. 6.1,6.2). Those can be scaled depending on maneuverability in the *Initial state* $state(0)$:

1. *Climb/Descent Rate* $\delta pitch_{max}(k)$ - the maximal climb or descent rate for Up/Down movements.
2. *Turn Rate* $\delta yaw_{max}(k)$ - the maximal turn rate for Left/Right movement.
3. *Acceleration* $\delta v_{max}(k)$ - the maximal acceleration in cruising speed range.

Definition 7. *State Space partition* Maneuverability *is depending on* Initial State*. There can not be the infinite count of* Movement Automatons.

The state space $StateSpace \in \mathbb{R}^n$ *can be separated into two exclusive subsets:*

$$StateSpace = [ImpactStates, NonImpactingStates] \tag{6.40}$$

The Impacting states *are states which bounds the* Maneuverability*:* $\delta pitch_{max}(k)$, $\delta yaw_{max}(k)$, $\delta v_{max}(k)$. *For each* impact state *is possible to define upper and lower boundary:*

$$\begin{aligned}&\forall impactState \in ImpactStates, \exists : \\ &\qquad lower(impactState) \leq value(impactState) \leq upper(impactState)\end{aligned} \tag{6.41}$$

The bounded interval of impact state can be separated into distinctive impact state segments *like follow:*

$$\begin{aligned}&impactState \in [lower, upper] : \\ &\qquad \{[lower, separator_1[\cdots \cup \ldots [separator_i, separator_{i+1}[\cdots \cup \ldots \\ &\qquad\qquad \cdots \cup \ldots [separator_n, upper]\} = \\ &\qquad\qquad\qquad = impactStateIntervals(impactState)\end{aligned} \tag{6.42}$$

Note. The interval length depends on model dynamics. The rule of thumb is to keep maximal climb/descend/turn/acceleration rates near constant value.

When partitioning of all impact States *finishes, the count of partitions is given as product of* count of partitions *for each member of* Impact States*:*

$$partitionCount = \prod_{impactState \in}^{ImpactStates} |impactStateIntervals(impactState)| \tag{6.43}$$

Note. Try to keep the count of partitions to minimum, each new interval increases the count of partitions geometrically.

There is finite number n of Impacting States*, these are separated into impactState−Intervals$_i$ with respective index $i \in 1 \dots n$. The* segment *with index defining position used* impacting state *intervals is given as* constrained space*:*

$$Segment(index) = \begin{bmatrix} impactState_1 \in impactStateIntervals_1[index_1], \\ \vdots \\ impactState_n \in impactStateIntervals_n[index_n], \\ \vdots \\ NonImpactingStates \end{bmatrix} \tag{6.44}$$

Each Segment *covers one of impacting state intervals combination, because the original intervals are exclusive, also* Segments *are exclusive. The* union *of all segments covers* State Space*:*

$$StateSpace = \bigcup_{\forall \quad index \in |impactStateIntervals|^n} Segment(index) \tag{6.45}$$

Segmented Movement Automaton: The segmentation of *state space* is done in (def. 7) any *state* belongs exactly to *Segment* of *State Space*. For each *Segment* in *State Space*it is possible to assess: *Climb/Descent Rate* $\delta pitch_{max}(k)$, *Turn Rate* $\delta yaw_{max}(k)$, and, *Acceleration* $\delta v_{max}(k)$.

Definition 8. *Movement Automaton for Segment(index)*
For for Model(eq. 6.34) with State (eq. 6.33) the input vector (eq. 6.32) is for position $[x, y, z]$ *and velocity defined like:*

$$\begin{aligned} \delta x(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \cos(\delta yaw(k)) \\ \delta y(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \sin(\delta yaw(k)) \\ \delta z(k) &= -(v(k) + \delta v(k)) \cos(\delta pitch(k)) \\ \delta v(k) &\in [-\delta v(k)_{max}, \delta v(k)_{max}] \end{aligned} \tag{6.46}$$

The acceleration $\delta v(k)$ is in interval $[-\delta v(k)_{max}, \delta v(k)_{max}]$, usually set to 0 ms^{-1}. The change of the orientation angles for *Movement Set* (eq. 6.35) is given in (tab. 6.3,6.4).

$input(movement)$	Straight	Down	Up	Left	Right
$\delta roll(k)[^\circ]$	0	0	0	0	0
$\delta pitch(k)[^\circ]$	0	$\delta pitch_{max}$	-$\delta pitch_{max}$	0	0
$\delta yaw(k)[^\circ]$	0	0	0	δyaw_{max}	-δyaw_{max}

Table 6.3: Orientation input values for main axes movements.

$input(movement)$	Down-Left	Down-Right	Up-Left	Up-Right
$\delta roll(k)[^\circ]$	0	0	0	0
$\delta pitch(k)[^\circ]$	-$\delta pitch_{max}$	-$\delta pitch_{max}$	$\delta pitch_{max}$	$\delta pitch_{max}$
$\delta yaw(k)[^\circ]$	δyaw_{max}	-δyaw_{max}	δyaw_{max}	-δyaw_{max}

Table 6.4: Orientation input values for diagonal axes movements.

Note. The *Trajectory* is calculated same as in (eq. 6.37). The *State Projection* is given as in (eq. 6.39).

Then the *Movement Automaton* for *Segment* $\in$ *State Space* is defined.

Definition 9. *Segmented Movement Automaton For system with segmented state space (eq. 6.45) there is for each state(k) in StateSpace injection function:*

$$ActiveMovementAutomaton : StateSpace \rightarrow MovementAutomaton \tag{6.47}$$

Selecting appropriate movement automaton *implementation (def. 8) for* state(k) $\in$ Segment $\subset$ State Space. *The mapping function (eq. 6.47) is injection mapping every state(k) to Segment then* Movement Automaton Implementation. *The trajectory generated is then given:*

$$Trajectory\begin{pmatrix} state(0), \\ Buffer \end{pmatrix} = \left\{ \begin{matrix} state(0) + \dots \\ \sum_{j=0}^{i-1} \begin{matrix} ActiveMovementAutomaton(state(j-1)). \\ .input(movement(j)) \end{matrix} : \\ i \in \{1 \dots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{matrix} \right\} \tag{6.48}$$

6.2.7 Reference Trajectory Generator

Reference Trajectory Generator: Segmented Movement Automaton (def. 9) with *trajectory function* (eq. 6.48) is used as *reference trajectory generator* for *complex systems.*

There is assumption that precise *path tracking* implementation exist for such system which with *thick reference trajectory* gives similar results to *plain movement automaton control.*

The *Reference trajectory* (eq. 6.49) for *Planned* movement set is given as projection of *Trajectory* time series to position time series $[x, y, z, t]$:

$$ReferenceTrajectory : Trajectory\begin{pmatrix} state(now), \\ Planned \end{pmatrix} \rightarrow \begin{bmatrix} x_{ref} \in \mathbb{R}^{|Planned|} \\ y_{ref} \in \mathbb{R}^{|Planned|} \\ z_{ref} \in \mathbb{R}^{|Planned|} \\ t_{ref} \in \mathbb{R}^{|Planned|} \end{bmatrix} \tag{6.49}$$

Predictor: The *Reference Trajectory Generator* (eq. 6.49) can be also used as predictor.

Note. The *Segmented Movement Automaton* (def. 9) is used in this work with one Segment equal to State space with input function given by (6.1, 6.2). The predictor used in *Reach set computation* is given by (eq. 6.49).

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

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

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