

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. *UAS Nonlinear Model* (sec. 6.2.1) - simple plane model used in this work as *controlled plant*.
2. *Movement Automaton* (sec. 6.2.2) - movement automaton for *UAS Nonlinear Model* constructed from scratch.
3. *Segmented Movement Automaton* (sec. 6.2.3) - for more complex systems the *State Space* can be *separated into Segments* and *segment movement automaton* is used to generate *thick reference trajectory*.
4. *Reference Trajectory Generator* (sec. 6.2.4) - other use of *Movement Automaton* as predictor for *reference trajectory calculation*.

6.2.1 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. 6.1) defined as positional state in euclidean position in right-hand euclidean space, where x , y , z can be abstracted as latitude, longitude, altitude.

$$state = [x, y, z, roll, pitch, yaw]^T \quad (6.1)$$

Input Vector (eq. 6.2) is defined as linear velocity of UAS v and angular speed of rigid body $\omega_{roll}, \omega_{pitch}, \omega_{yaw}$.

$$input = [v, \omega_{roll}, \omega_{pitch}, \omega_{yaw}]^T \quad (6.2)$$

Velocity distribution function (eq. 6.3) is defined trough standard rotation matrix and linear velocity v , oriented velocity $[v_x, v_y, v_z]$ given by (eq. 6.4).

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v \cos(pitch) \cos(yaw) \\ v \cos(pitch) \sin(yaw) \\ -v \sin(pitch) \end{bmatrix} \quad (6.3)$$

UAS Nonlinear Model (eq. 6.4) is given by *first order equations*:

$$\begin{aligned}\frac{\partial x}{\partial time} &= v \cos(pitch) \cos(yaw); & \frac{\partial roll}{\partial time} &= \omega_{roll}; \\ \frac{\partial y}{\partial time} &= v \cos(pitch) \sin(yaw); & \frac{\partial pitch}{\partial time} &= \omega_{pitch}; \\ \frac{\partial z}{\partial time} &= -v \sin(pitch); & \frac{\partial yaw}{\partial time} &= \omega_{yaw};\end{aligned}\tag{6.4}$$

6.2.2 Movement Automaton for UAS Model

Motivation: An *UAS Nonlinear Model* (eq. 6.4) can be modeled by *Movement Automaton* (def. ??).

Movement Primitives by (def. ??) are given as (eq. ??). 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. ??) have 0 duration.

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

Movements (def. ??) for *fixed step k* we start with discretization of the input variables.

The *linear velocity* in text step is given:

$$v(k+1) = v(k) + \delta v(k)\tag{6.5}$$

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

$$\begin{aligned}roll(k+1) &= roll(k) + \delta roll(k) \\ pitch(k+1) &= pitch(k) + \delta pitch(k) \\ yaw(k+1) &= yaw(k) + \delta yaw(k)\end{aligned}\tag{6.6}$$

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:

$$\begin{aligned}x(k+1) &= x(k) + v(k+1) \cos(pitch(k+1)) \cos(yaw(k+1)) &= \delta x(k) \\ y(k+1) &= y(k) + v(k+1) \cos(pitch(k+1)) \sin(yaw(k+1)) &= \delta y(k) \\ z(k+1) &= z(k) - v(k+1) \sin(pitch(k+1)) &= \delta z(k) \\ time(k+1) &= time(k) + 1 &= \delta time(k)\end{aligned}\tag{6.7}$$

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

$$input(k) = \begin{bmatrix} \delta x(k), \delta y(k), \delta z(k), \delta v(k), \\ \delta roll(k), \delta pitch(k), \delta yaw(k), \delta time(k) \end{bmatrix}^T \quad (6.8)$$

The *state vector* for discrete time is given:

$$state(k) = \begin{bmatrix} x(k), y(k), z(k), v(k), \\ roll(k), pitch(k), yaw(k), time(k) \end{bmatrix}^T \quad (6.9)$$

The nonlinear model (eq. 6.4) is then reduced to *linear discrete model* (eq. 6.10) given by *apply movements* function (eq. 6.5, 6.6, 6.7).

$$state(k+1) = applyMovement(state(k), input(k)) \quad (6.10)$$

Movement Set for linear discrete model (eq. 6.10) 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
$\delta roll(k)[^\circ]$	0	0	0	0	0
$\delta pitch(k)[^\circ]$	0	15°	-15°	0	0
$\delta yaw(k)[^\circ]$	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
$\delta roll(k)[^\circ]$	0	0	0	0
$\delta pitch(k)[^\circ]$	-15°	-15°	15°	15°
$\delta yaw(k)[^\circ]$	15°	-15°	15°	-15°

Table 6.2: Input values for diagonal axes movements.

Note. *Movement set* in shorten form is given as

$$MovementSet = \left\{ \begin{array}{l} Straight, Left, Right, Up, Down, \\ DownLeft, DownRight, UpLeft, UpRight \end{array} \right\} \quad (6.11)$$

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

$$Buffer \in MovementSet^*(eq.6.11), \quad |Buffer| \in \mathbb{N} \quad (6.12)$$

Trajectory (eq. 6.13) is then given as the time-series of discrete states:

$$Trajectory(state(0), Buffer) = \left\{ \begin{array}{l} state(0) + \sum_{j=0}^{i-1} input(movement(j)) : \\ i \in \{1 \dots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{array} \right\} \quad (6.13)$$

Trajectory (eq. 6.13) is ordered set of states bounded to discrete time $0 \dots n$, where n is member count of *Buffer*. Trajectory set has $n + 1$ members:

$$Trajectory(state(0), Buffer) = \left\{ \begin{array}{l} state(0) = state(0) + \{\} \\ state(1) = state(0) + input(movement(1)) \\ state(2) = state(0) + input(movement(1)) + input(movement(2)) \\ \vdots = \vdots \\ state(n) = state(0) + input(movement(1)) + \dots + input(movement(n)) \end{array} \right\} \quad (6.14)$$

State Projection (eq. 6.15) for the *Trajectory* (eq. 6.13) is given as follow:

$$StateProjection(Trajectory, time) = Trajectory.getMemberByIndex(time + 1) \quad (6.15)$$

Note. *Movement Automaton* for system (eq. 6.4) with given (as. 1) is established with all related properties (sec. ??).

6.2.3 Segmented Movement Automaton

Motivation: Constructing *Movement Automaton* for more complex system can be tedious. Used *Movement Automaton* for *UAS system* (6.4) has decoupled control which is not true

for most of the copters/planes [1].

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 1. *State Space partition Maneuverability is depending on Initial State. There can not be the infinite count of Movement Automations.*

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

$$StateSpace = [ImpactStates, NonImpactingStates] \quad (6.16)$$

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:

$\forall impactState \in ImpactStates, \exists :$

$$lower(impactState) \leq value(impactState) \leq upper(impactState) \quad (6.17)$$

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

$impactState \in [lower, upper] :$

$$\begin{aligned} \{[lower, separator_1[\cdots \cup \dots [separator_i, separator_{i+1}[\cdots \cup \dots \\ \cdots \cup \dots [separator_n, upper]]\} = \\ = impactStateIntervals(impactState) \end{aligned} \quad (6.18)$$

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:

$$partitionCount = \prod_{impactState \in ImpactStates} |impactStateIntervals(impactState)| \quad (6.19)$$

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 *impactStateIntervals_i* with respective index $i \in 1 \dots n$. The segment with index defining position used impacting state intervals is given as constrained space:

$$Segment(index) = \begin{bmatrix} impactState_1 \in impactStateIntervals_1[index_1], \\ \vdots \\ impactState_n \in impactStateIntervals_n[index_n], \\ \vdots \\ NonImpactingStates \end{bmatrix} \quad (6.20)$$

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:

$$StateSpace = \bigcup_{\forall \quad index \in |impactStateIntervals|^n} Segment(index) \quad (6.21)$$

Segmented Movement Automaton: The segmentation of *state space* is done in (def. 1) 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 2. *Movement Automaton for Segment(index)*

For for Model(eq. 6.10) with State (eq. 6.9) the input vector (eq. 6.8) is for position $[x, y, z]$ and velocity defined like:

$$\begin{aligned} \delta x(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \cos(\delta yaw(k)) \\ \delta y(k) &= (v(k) + \delta v(k)) \cos(\delta pitch(k)) \sin(\delta yaw(k)) \\ \delta z(k) &= -(v(k) + \delta v(k)) \cos(\delta pitch(k)) \\ \delta v(k) &\in [-\delta v(k)_{max}, \delta v(k)_{max}] \end{aligned} \quad (6.22)$$

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. 6.11) 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}	$-\delta yaw_{max}$

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}	$-\delta yaw_{max}$	δyaw_{max}	$-\delta yaw_{max}$

Table 6.4: Orientation input values for diagonal axes movements.

Note. The *Trajectory* is calculated same as in (eq. 6.13). The *State Projection* is given as in (eq. 6.15).

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

Definition 3. *Segmented Movement Automaton* For system with segmented state space (eq. 6.21) there is for each state(k) in StateSpace injection function:

$$ActiveMovementAutomaton : StateSpace \rightarrow MovementAutomaton \quad (6.23)$$

Selecting appropriate movement automaton implementation (def. 2) for state(k) \in Segment \subset State Space. The mapping function (eq. 6.23) is injection mapping every state(k) to Segment then Movement Automaton Implementation. The trajectory generated is then given:

$$Trajectory \left(\begin{matrix} state(0), \\ Buffer \end{matrix} \right) = \left\{ \begin{matrix} state(0) + \dots \\ \sum_{j=0}^{i-1} ActiveMovementAutomaton(state(j-1)). \\ input(movement(j)) \\ i \in \{1 \dots |Buffer| + 1\}, \\ movement(\cdot) \in Buffer \end{matrix} \right\} \quad (6.24)$$

6.2.4 Reference Trajectory Generator

Reference Trajectory Generator: Segmented Movement Automaton (def. 3) with *trajectory function* (eq. 6.24) 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. 6.25) for *Planned* movement set is given as projection of *Trajectory* time series to position time series $[x, y, z, t]$:

$$ReferenceTrajectory : Trajectory \left(\begin{matrix} state(now), \\ Planned \end{matrix} \right) \rightarrow \begin{bmatrix} x_{ref} \in \mathbb{R}^{|Planned|} \\ y_{ref} \in \mathbb{R}^{|Planned|} \\ z_{ref} \in \mathbb{R}^{|Planned|} \\ t_{ref} \in \mathbb{R}^{|Planned|} \end{bmatrix} \quad (6.25)$$

Predictor: The *Reference Trajectory Generator* (eq. 6.25) can be also used as predictor.

Note. The *Segmented Movement Automaton* (def. 3) 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. 6.25).

Bibliography

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