**Appendix C**

**Movement Automaton Theory**

This appendix covers theory related to *Movement Automaton*.

1. *Specialization of Hybrid Automaton* (sec. C.1) - the specialization of the hybrid automaton to fulfill control/approximation roles in our approach.
2. *Formal Movement Automaton Definition* (sec. C.2) - the formal definition of *movement automaton* used in our approach.
3. *Segmented Movement Automaton* (sec. C.3) - for more complex systems the *State Space* can be *separated into Segments,* and *segment movement automaton* is used to generate a *thick reference trajectory*.
4. *Reference Trajectory Generator* (sec. C.4) - other use of *Movement Automaton* as the predictor for *reference trajectory calculation*.

# C.1 Specialization of Hybrid Automaton

**Idea:** There is a need for *fast trajectory approximation* method. The basic idea is taken from the pilot steering a plane. The pilot has issued commands from a navigator in a 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 an airplane (fig. C.1). The *navigator* is issuing a command every second to a pilot. Hands of the pilot translate each command to an input signal (blue line). The command validity period (black frame) is split into *transition period* (red frame) when the input signal is changing and primitive period (magenta frame) when the position of the input signal is static.

1

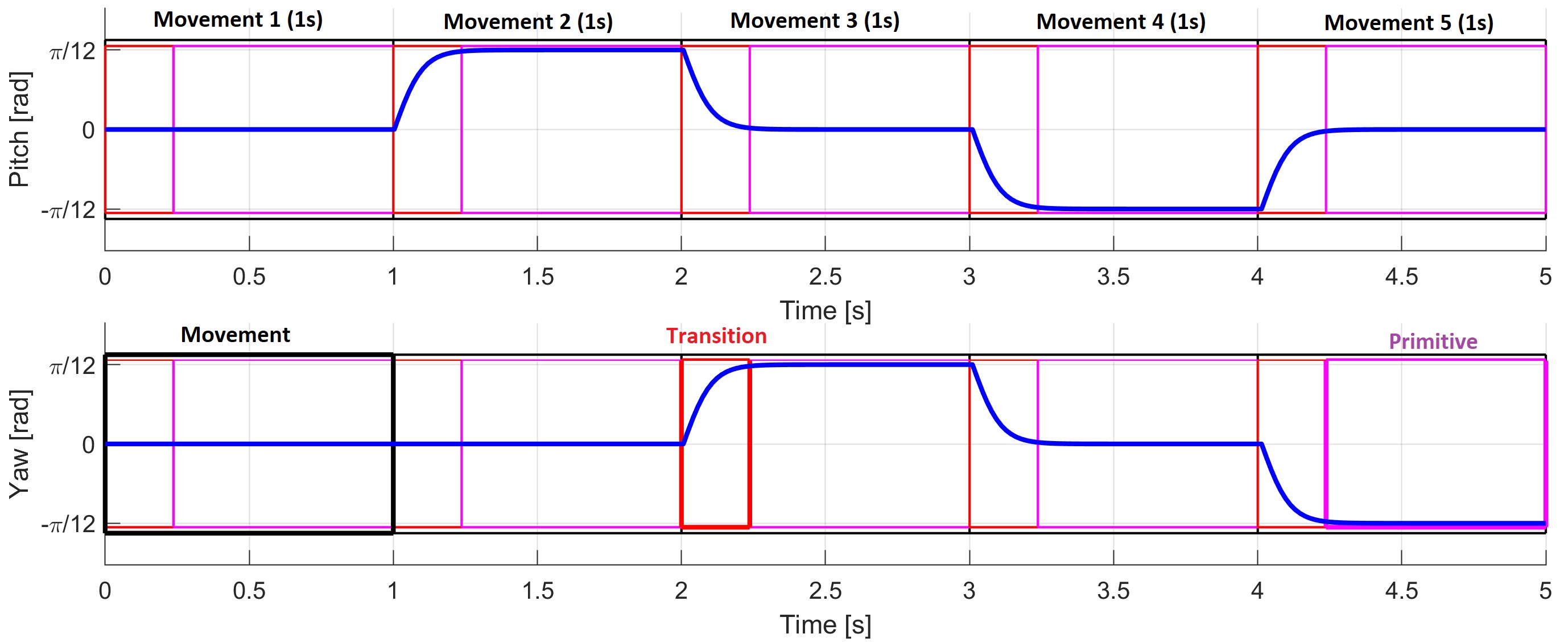


Figure C.1: Example of input signal segmentation to movements.

*Note.* The hybrid automaton (sec. **??**) can be used as the base for simple control mechanism imitating navigators command execution by the 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 execution.

**Definition 1.** *Movement Primitive:*

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

*MovementPrimitive*(*vectorField,minimalDuration,parameters*)

(C.1) *V ectorField* : *SystemState* × *parameters* → *SystemState*

**Example:** Let say that *UAS* system is given as *position*˙ = *velocity*, then let us have two *MovementPrimitives*:

1. *Stay* - *minimalTime* = 1*s*, *parameters* = {}, *V ectorField* : *position*˙ = 0.
2. *Move* - *minimalTime* = 1*s*, *parameters* = {*velocity*}, *V ectorField* : *position*˙ = *velocity*.

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

|  |  |  |
| --- | --- | --- |
|   *t* ∈ [0*,*5]  *position*(*t*) = Θ(*...*) *t* ∈ (5*,*15]  *t* ∈ (15*,*22] | : 10 × *t* + *position*(0)  : 0 × (*t* − 5) + *position*(5)  : 4 × (*t* − 15) + *position*(15) | (C.2) |

C.1. Specialization of Hybrid Automaton

The *example* given by (eq. C.2) is fairly primitive, but imagine UAS system given by nonlinear dynamics [1]. Then defining transfer function for a given command chain can be impossible.

**Definition 2.** *Movement Transition:*

System state *can be different from intended movement application, the notion of* Transition *is therefore introduced as stabilizing element in movement chaining (eq. C.3).*

*Transition* : *MovementPrimitive* × *SystemState* → *MovementPrimitive* (C.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. C.2) can be rewritten as combination of *MovementPrimitives* (eq. C.1) and *Transitions* (eq. C.3):

*Transition*(*Stay,Move*)*,Move*(5*s,*10*m/s*)*,*

*Transition*(*Move,Stay*)*,Stay*(10*s*)*,*

*Transition*(*Stay,Move*)*,Move*(7*s,*4*m/s*) (C.4)

.

*Note.* There are two types of *movement primitives*:

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

**Movement Mapping Example:** Transition/MovementPrimitive pairs (eq. C.3) can be mapped into movements (eq. C.5).

*Move*(5*s,*10*m/s*) : *Transition*(*Stay,Move*)*,Move*(5*s,*10*m/s*)*,*

*Stay*(10*s*) : *Transition*(*Move,Stay*)*,Stay*(10*s*)*,* (C.5)

*Move*(7*s,*4*m/s*) : *Transition*(*Stay,Move*)*,Move*(7*s,*4*m/s*)

**Definition 3.** *Movement:*

*Movement can consist from multiple* Transitions *(eq. C.3) and one* MovementPrimitive *(eq. C.1), the duration of* MovementPrimitive *can be shortened by* Transitions *duration.*

Movement *is defined as follows:*

 

*InitialTransition*(*...*)[0*..*∗]*,*

 

*initialState,*   

 *transitionState,* 

   

*initialTime*[0*..*1]*,*   

*Movement*  = *Chain**MovementPrimitive**remainingDuration,* (C.6)

    

 *duration,*        *parameters* 

  *parameters*[0*..*1]  

*LeaveTransition*(*...*)[0*..*∗]*,*

Chain function *connects multiple* initial Transitions *which are applied at* initialState *at* initialTime*. Then own* MovementPrimitive *(eq. C.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. C.7) for *movement* is given as the sum of *MovementPrimitive* (eq. C.1) minimal time, and *Transition* (eq. C.3) in/out combined minimal time.

*minimalTime*(*MovementPrimitive*)+

*minimalTime*(*Movement*) = (C.7) max*in/out* {*time*(*Transition*)}

**Movement Chaining:** *Movements* can be *chained* and applied to initial *system state* to generate *system trajectory*. Example of the trajectory is given by (eq. C.2). Movements are reversibly obtained by participation such a *trajectory* into *Movement primitives* and *Transitions*. Then sample *Trajectory* for *n* ∈ N+ movements looks like (eq. C.8).

*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)

...

*Trajectory*(*tn*−1*,tn*] = *Movementn*(*Trajectory*(*tn*−1)*,tn*−1*,durationn,parametersn*)

## (C.8)

Given *Trajectory* at time *t*0 is given as the 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 behavior continues for movements 2*,...,n*.

**Definition 4.** *Movement Buffer:*

Movements *can be chained into* Buffer *with the assumption of* continuous movement execution*.* Continuous movement executions *each movement in the chain (eq. C.8) is executed in time interval τi* = (*ti*−1*,ti*] *where i is movement order and* ∀ *Movementi starting time is t*0 *or ti*−1 *from the previous movement. With given assumption* Buffer *is given as (eq. C.9) with parameters ti*−1*,ti omitted, due t*0 *and durationi dependency.*

C.1. Specialization of Hybrid Automaton

*Buffer* = {*Movementi*(*durationi,parametersi*)}*i* ∈ N+ (C.9)

**Definition 5.** *Movement Automaton Trajectory:*

*Let say system* State∈ R*n which* Trajectory *is defined by movement chaining (eq. C.8), applied on some* initial time *t*0 ∈ R+ *and final time* *, with movements contained in* Buffer *(eq. C.9) is given as* Trajectory *(eq. C.10).*

*Trajectory*(*t*0*,State*(*t*0)*,Buffer*) *or Trajectory*(*State*0*,Buffer*) *if t*0 = 0 (C.10)

*Note.* The space dimension of *Trajectories* is R*n*+1 if the space dimension of state *Space* is *Rn*, because *Trajectory space* contains evolution of *Space* in time interval *T*[*t*0*,tf*]. The transformation from *transfer function* (eq. C.2) to *trajectory* (eq. C.10) is natural, only set of *Movement primitives* (eq. C.1) and set of *Transitions* (eq. C.3) is required.

**State Projection:** *Trajectory* (eq. C.10) is natural evolution of space over time, then there exists *StateProjection* function (eq. C.11) which returns *State* for specific *Time*.

*StateProjection* : *Trajectory* × *Time* → *State*(*Time*) (C.11)

# C.2 Formal Movement Automaton Definition

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

|  |  |
| --- | --- |
| *InitialState* :∈ R*h,h* ∈ N+ | (C.12) |
| *System* : *State*˙ = *f*(*Time,State,Input*) *or vectorField* | (C.13) |
|    *vectorField,*     ∈ N+  *Primitives* = *MovementPrimitivei* *minimalDuration,* *i*   *parameters*  | (C.14) |
|  | (C.15) |
|        *Transitiono*[0*..*∗]*,*  *m* ∈ *N*+  *Movements* = *Movementm* *MovementPrimitivep*   *Transitionr*[0*..*∗]*,* *o*6=*r* | (C.16) |
| *Buffer* = {*Movements*(*durations,parameterss*)}*s* ∈ N+ | (C.17) |
| *Executed* = {*Movements*(*durations,parameterst*)}*t* ∈ N+ | (C.18) |
| *Builder* : *Movement* × *MovementPrimitive* → *Movement* | (C.19) |
| *Trajectory* : *InitialState* × *Movementu* → *State* × *Time,u* ∈ *N*+ | (C.20) |
| *StateProjection* : *Trajectory* × *Time* → *State*(*Time*) | (C.21) |
| System *(eq. C.13) is given in form of* differential equations ˙*x* = *f*(*t,x,u*) *or* other | |

transformable equivalent*, with* initial state *(eq. C.12).*

Movements *(eq. C.8) are defined as sequence of necessary* initial transitions *(eq. C.15),* movement primitive *(eq. C.14), and,* leave transitions *(C.15).*

The buffer *contains a set of* movement primitives *(eq. C.14) to be executed in order to achieve the desired goal.* Builder *(eq. C.19) assures that first* movement primitive *(eq. C.1)from* Buffer *(eq. C.17) is transformed into* next movement *(eq. C.16) based on* current movement *(eq.C.16).*

*The system* trajectory *(eq. C.20) is defined in (eq. C.10).* State projection *(eqs. C.11,C.21) is giving* State *variable for time t* ∈ [*t*0*,tmax*] *where tmax is given by:*

*tmax* = *t*0 + X *Buffer.Movement*(*i*)*.movementDuration* (C.22)

*i*=1*,u*

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

*The reach set R* (C.23) for system d*/*dt state = *model*(*state,input*) with initial state *state*0 = *state*(*ti*) in time interval [*ti,ti*+1[ is with existing control strategy *input*(*t*) ∈

C.3. Segmented Movement Automaton

*ControlStrategy*(*t*). The reach set *R*(*state*0*,t*0*,t*1) where *t*1 *> t*0.

*R*(*state*0*,t*0*,t*1) = [{*state*(*s*) : *input*(*s*) ∈ *ControlStrategy*(*s*)*,s* ∈ (*t*0*,t*1]} (C.23)

*The reach set* R (C.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 [*ti,ti*+1[.

*ReachSet*(*state*0*,ti,ti*+1) =

{*Trajectory*(*state*0*,buffer*) : duration(*buffer*) ≤ (*ti*+1 − *ti*)} (C.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 UAS will not intersect any threat. This means that the controlled system d*/*dt state = *model*(*state,input*) is *weakly invariant* concerning the complement of the threats, and 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 ∈ *SafeSpace* remains inside *State*(*t*) ∈ *SafeSpace* [2].

# C.3 Segmented Movement Automaton

**Motivation:** Constructing *Movement Automaton* for the more complex system can be tedious. Used *Movement Automaton* for *UAS system* (**??**) 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. **??**,**??**). Those can be scaled depending on maneuverability in the

*Initial state state*(0):

1. *Climb/Descent Rate δpitchmax*(*k*) - the maximal climb or descent rate for Up/Down movements.
2. *Turn Rate δyawmax*(*k*) - the maximal turn rate for Left/Right movement.
3. *Acceleration δvmax*(*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* ∈ R*n can be separated into two exclusive subsets:*

*StateSpace* = [*ImpactStates,NonImpactingStates*] (C.25)

*The* Impacting states *are states which bounds the* Maneuverability*: δpitchmax*(*k*)*, δyawmax*(*k*)*, δvmax*(*k*)*. For each* impact state *is possible to define upper and lower boundary:*

∀*impactState* ∈ *ImpactStates,*∃ :

*lower*(*impactState*) ≤ *value*(*impactState*) ≤ *upper*(*impactState*) (C.26)

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

*impactState* ∈ [*lower,upper*] :

{[*lower,separator*1[··· ∪ *...*[*separatori,separatori*+1[··· ∪ *...*

··· ∪ *...*[*separatorn,upper*]} =

= *impactStateIntervals*(*impactState*) (C.27)

*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 the product of* count of partitions *for each member of* Impact States*:*

*ImpactStates*

*partitionCount* = Y |*impactStateIntervals*(*impactState*)| (C.28)

*impactState*∈

*Note.* Try to keep the count of partitions to a minimum; each new interval increases the count of partitions geometrically.

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

  *impactState*1 ∈ *impactStateIntervals*1[*index*1]*,*

 *...* 

 

 

*Segment*(*index*) = *impactStaten* ∈ *impactStateIntervalsn*[*indexn*]*,* (C.29)

 

 *...* 



 

*NonImpactingStates*

*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*:*

C.3. Segmented Movement Automaton

*StateSpace* = [ *Segment*(*index*) (C.30)

∀ *index*∈|*impactStateIntervals*|*n*

**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 δpitchmax*(*k*), *Turn Rate δyawmax*(*k*), and, *Acceleration δvmax*(*k*).

**Definition 8.** *Movement Automaton for Segment(index)*

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

*δx*(*k*) = (*v*(*k*) + *δv*(*k*))cos(*δpitch*(*k*))cos(*δyaw*(*k*))

*δy*(*k*) = (*v*(*k*) + *δv*(*k*))cos(*δpitch*(*k*))sin(*δyaw*(*k*))

## (C.31)

*δz*(*k*) = −(*v*(*k*) + *δv*(*k*))cos(*δpitch*(*k*)) *δv*(*k*) ∈ [−*δv*(*k*)*max,δv*(*k*)*max*]

The acceleration *δv*(*k*) is in interval [−*δv*(*k*)*max,δv*(*k*)*max*], usually set to 0 *ms*−1. The change of the orientation angles for *Movement Set* (eq. **??**) is given in (tab. C.1,C.2).

*input*(*movement*) Straight Down Up Left Right

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *δroll*(*k*)[◦] | 0 | 0 | 0 | 0 | 0 |
| *δpitch*(*k*)[◦] | 0 | *δpitchmax* | -*δpitchmax* | 0 | 0 |
| *δyaw*(*k*)[◦] | 0 | 0 | 0 | *δyawmax* | -*δyawmax* |

Table C.1: Orientation input values for main axes movements. *input*(*movement*) Down-Left Down-Right Up-Left Up-Right

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *δroll*(*k*)[◦] | 0 | 0 | 0 | 0 |
| *δpitch*(*k*)[◦] | -*δpitchmax* | -*δpitchmax* | *δpitchmax* | *δpitchmax* |
| *δyaw*(*k*)[◦] | *δyawmax* | -*δyawmax* | *δyawmax* | -*δyawmax* |

Table C.2: Orientation input values for diagonal axes movements.

*Note.* The *Trajectory* is calculated the same as in (eq. C.33). The *State Projection* is given as in (eq. C.36).

Then the *Movement Automaton* for *Segment* ∈ *State Space* is defined.

**Definition 9.** *Segmented Movement Automaton For system with segmented state space (eq. C.30) there is for each state*(*k*) *in StateSpace injection function:*

*ActiveMovementAutomaton* : *StateSpace* → *MovementAutomaton* (C.32) *Trajectory of non-segmented movement automaton (eq. C.33) is then given as the timeseries of discrete states:*

|  |  |
| --- | --- |
|  *i*−1   *state*(0) + X*j*=0 *input*(*movement*(*j*)) :  *Trajectory*(*state*(0)*,Buffer*) =   *imovement*∈ {1*...*|*Buffer*(·) ∈ *Buffer*| + 1}*,*  | (C.33) |

*Selecting appropriate* movement automaton *implementation (def. 8) for* state(k) ∈ Segment ⊂ State Space*. The mapping function (eq. C.32) is injection mapping every state(k) to Segment then* Movement Automaton Implementation*. The trajectory generated is then given:*

|  |  |
| --- | --- |
| ! |   *state*(0) + *...*  X*i*−1 *ActiveMovementAutomaton*(*state*(*j* − 1))*.* :  = *j*=0 *.input*(*movement*(*j*))   *imovement*∈ {1*...*|*Buffer*(·) ∈ *Buffer*| + 1}*,*  |

(C.34)

# C.4 Reference Trajectory Generator

**Reference Trajectory Generator:** Segmented Movement Automaton (def. 9) with *trajectory function* (eq. C.34) is used as a *reference trajectory generator* for *complex systems*.

There is an 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. C.35) for *Planned* movement set is given as projection of *Trajectory* time series to position time series [*x,y,z,t*]:

R

*xref* ∈ |*Planned*|

*ReferenceTrajectory* : *Trajectory* *state*(*now*)*,*! → *yref* ∈ R||*PlannedPlanned*|| (C.35)

*Planned* *zref* ∈ R 

  *tref* ∈ R|*Planned*|

**Predictor:** The *Reference Trajectory Generator* (eq. C.35) can also be used as a predictor.

*Note.* The *Segmented Movement Automaton* (def. 9) is used in this work with one

C.4. Reference Trajectory Generator

Segment equal to State space with input function given by (**??**, **??**). The predictor used in *Reach set computation* is given by (eq. C.35).

**State Projection** (eq. C.36) for the *Trajectory* (eq. C.33) is given as follow:

*StateProjection*(*Trajectory,time*) = *Trajectory.getMemberByIndex*(*time* + 1)

## (C.36)

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

12 Appendix C. Movement Automaton Theory

**Bibliography**

1. Thor I Fossen. Mathematical models for control of aircraft and satellites. *Department of Engineering Cybernetics Norwegian University of Science and Technology*, 2011.
2. Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999.

13