J Intell Robot Syst (2010) 58:69–94
DOI 10.1007/s10846-009-9344-8

Transition Management for the Smooth Flight of a Small Autonomous Helicopter

Andres Y. Agudelo-Toro · Carlos M. Velez

Received: 25 August 2008 / Accepted: 18 May 2009 / Published online: 12 June 2009

Abstract This work is centered in the definition of a transition management system for a small autonomous helicopter based on trajectory smoothing and a finite state machine (FSM). A smooth flight schedule decreases transients originated by direction changes and flight mode transitions (e.g., horizontal flight to hover mode). Although previous works have presented trajectory generation and FSM oriented controls, no previous studies have mixed these approaches in a single framework together with speed transitions. The proposed methods are validated in simulation with a realistic dynamic model of a small helicopter.

Keywords Helicopter · Autonomous air vehicle · Robotics · Trajectory · Smooth speed

Mathematics Subject Classifications (2000) 93C85 · 93C65 · 65D10 · 68Q45

Abbreviations

UAV	Unmanned aerial vehicles
FSM	Finite state machine
WP	Flight waypoints
PID	Proportional-integral-derivative
CPU	Central processing unit

A. Y. Agudelo-Toro (✉) · C. M. Velez
Basic Sciences Department, EAFIT University,
Cr 49 No. 7 sur 50, Medellin, Colombia
e-mail: aagudel6@eafit.edu.co

C. M. Velez
e-mail: cmvelez@eafit.edu.co

1 Introduction

Guidance of an autonomous helicopter requires transforming a predefined flight schedule, described as a series of discrete point locations and maneuvers, into a continuous smooth three dimensional trajectory. A smooth flight schedule reduces the control system error and decreases transients caused by directional and flight mode changes [9, 11, 12, 22]. Helicopter flight, differently from fixed wing flight, must be described by three dimensional position and flight mode changes (e.g., forward flight to hover). Reduced transients depend on continuity of the planned trajectory, speed transitions and adequate management of mode transfers.

Generation of a smooth trajectory can be done before flight by means of an n-dimensional regression method such as spline functions [2]. Speed transitions can also be parameterized creating progressive changes and controlling acceleration. Mode transfers can be handled with known control techniques such as bumpless transfer and gain scheduling. A small hobby helicopter and, in general, a helicopter is a nonlinear dynamic system [8, 18], so a single linear control system may not be effective for all flight modes [11, 13]. A mode-transition manager for this type of system requires a combination of discrete asynchronous event logic with continuous time sampled dynamics in a hybrid control scheme [3, 4, 28].

This paper is centered on the mathematical definition and testing in simulation of a hybrid control system that combines spline functions and a finite state machine for an autonomous small helicopter [25, 27]. An interesting approach to mix position and speed smoothing in a single numerical method is also presented. While it is simple to smooth position and speed independently, it is not simple to smooth 2D position and speed simultaneously. If the two-dimensional position of a particle is described by $\langle x, y\rangle = f(t)$, its speed is implicitly declared by $f'(t)$. If $f(t)$ was designed to traverse a set of points, it is harder at the same time to make it reach expected speeds at these points.

The proposed method is two-dimensional (x, y) and considers altitude changes (z) only during the takeoff sequence. Some of the methods here proposed can be easily extended to three-dimensional space, but still, full 3D flight such as the one in acrobatic maneuvers, requires a more detailed study of attitude changes, and is beyond the purpose of this paper. This method also assumes that control of the system is an independent problem and only deals with the effects of the improved trajectory in the final state of the vehicle. The trajectory generation processes does not produce an error by itself because it exactly passes through the set of data points. The proposed methods are validated in simulation with the dynamic model of a small helicopter using the control error as a benefit indicator. The system is also tested on a real flight computer to verify it satisfies computational constraints.

In the remaining sections of this paper related work will be presented and the helicopter simulation model and control strategy will be described. The smoothing algorithms are then provided and the finite state machine is formally defined. Finally, the benefits are measured, following with conclusions and proposed future work.

2 Related Work

Linear programming and mixed integer lineal programming are common used techniques for path planning and route optimization [17, 20]. One clear disadvantage of

linear programming is its computational expensiveness (even offline) which grows with the number of waypoints.

The use of FSM and hybrid controllers for unmanned helicopters has also been explored before [6, 7, 11, 24]. In [6] emphasis is made on the use of a state machine for the mission control of a helicopter and the management of discrete events. The automaton defined was designed specifically to fulfill a single static mission and there is not a clear description of the flight modes or trajectory generation. In [7] a distributed architecture for an unmanned helicopter UAV is discussed. The flight modes of the state automaton are described but experimental results do not appear in the paper. The specific use of a tool for the design of hybrid control systems with a hybrid control architecture appears in [11] and [24]. The study is nevertheless focused to the architecture and benefits from the framework, and not in its implementation.

The use of a FSM for an autonomous helicopter appears in [23], but this work is limited to the landing problem. In [14] a formal method for the design of reactive systems is described and applied to the flight control system of a helicopter. This work is mostly design oriented and does not include smoothing.

The work in [22] describes a FSM for autonomous helicopter flight and provides a detailed description of each state and its integration with the system architecture. This work lacks description of the transitions or trajectory generation. In [9] a type of Bezier splines is used for obstacle avoidance in simulation of a laser guided helicopter. Some results are shown but individual state error data was not found. Finally, the work in [12] describes a smoothing strategy based on Catmull–Rom splines for helicopter flight. The work covers trajectory smoothing but not speed or mode management.

3 Helicopter Model

Trajectory optimization and transition management methods presented in this work were tested on the dynamic model of a Miniature Aircraft X-Cell Gas Graphite helicopter with rotor diameter of 155 cm. The dynamic model [25, 26] is an improved implementation of the X-Cell model [8, 18], validated in real flight.

Helicopter flight is governed by four basic forces: lift, drag, thrust and weight. Lift and drag rely on fuselage and blade shapes and interact mainly with external wind conditions and flow produced by the main and tail rotors. Thrust and weight counteract to maintain the helicopter in flight and are related to interactions of the helicopter blades and body. The model has nine inputs (including three wind disturbances) and 12 state variables. State variables evolve in the vehicle frame of reference as presented in Fig. 1. The dynamic model input vector is,

$$\left[\delta_{col}, \delta_{lon}, \delta_{lat}, \delta_{ped}, \delta_t, u_w, v_w, w_w\right]^T$$

Where δ values represent collective, longitudinal, lateral, pedal and throttle input confined to the interval $[-1, 1]$ to represent saturation; and (u_w, v_w, w_w) values represent the wind speed vector in the body frame of reference.

State variables are,

$$\left[u, v, w, p, q, r, \varphi, \theta, \psi, a_1, b_1, \Omega\right]^T$$

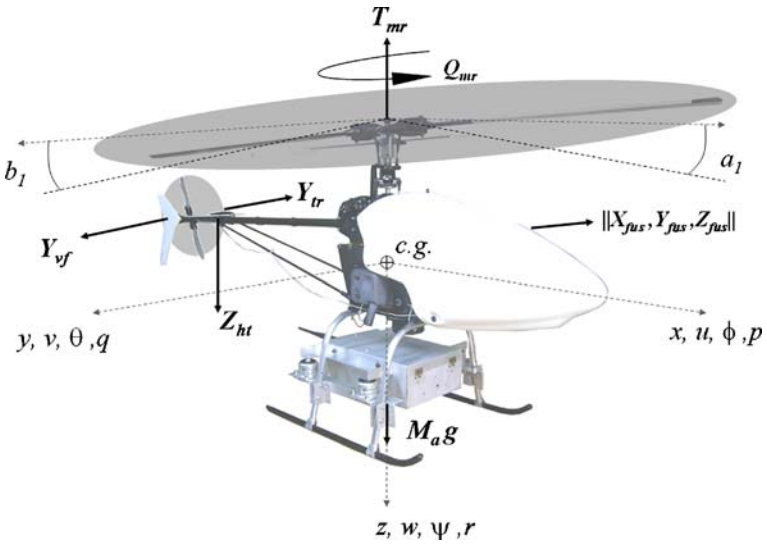

Fig. 1 Helicopter body axes and forces

where (u, v, w) are the linear speeds in body frame reference; (p, q, r) are the angular rates in body frame; (φ, θ, ψ) are roll, pitch and yaw Euler angles in reference frame (fixed to Earth); (a_1, b_1) are the longitudinal and lateral flapping angles; and Ω is the main rotor angular speed. Important model parameters include: mass (M_a), which varies from M_{full} to M_{empty} as fuel is consumed, and moments of inertia (I) which also vary linearly as fuel is consumed.

The combined Newton's Second Law and Euler's Equations for linear and angular motion of a rigid body around its center of gravity, together with the flapping and main rotor speed dynamics that model the vehicle, are presented in Eq. 1. In this equation, (X, Y, Z) represent forces and (L, M, N) moments around the x, y and z axes. Sub- and super-scripts *mr*, *tr*, *vf*, *ht* and *e* refer respectively to main rotor, tail rotor, vertical fin, horizontal stabilizer and engine. Parameter g is the gravity constant and Q_e is the engine torque.

$$
\begin{aligned}
\dot{u} &= vr - wq - g\sin\theta + \left(X_{mr} + X_{fus}\right)/M_a,\\
\dot{v} &= pw - ru + g\sin\varphi\cos\theta + \left(Y_{mr} + Y_{fus} + Y_{tr} + Y_{vf}\right)/M_a,\\
\dot{w} &= uq - vp + g\cos\varphi\cos\theta + \left(Z_{mr} + Z_{fus} + Z_{ht}\right)/M_a,\\
\dot{p} &= qr\left(I_y - I_z\right)/I_x + \left(L_{mr} + L_{tr} + L_{vf}\right)/I_x,\\
\dot{q} &= pr(I_z - I_x)/I_y + (M_{mr} + M_{ht})/I_y,\\
\dot{r} &= pq\left(I_x - I_y\right)/I_z + \left(-Q_e + N_{tr} + N_{vf}\right)/I_z,\\
\dot{\phi} &= p + q\tan\theta\sin\phi + r\tan\theta\cos\phi,\\
\dot{\theta} &= q\cos\phi - r\sin\phi,\\
\dot{\psi} &= q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta},\\
\dot{a}_1 &= f_{a_1}\left(q, a_1, u, w, \Omega\right),\\
\dot{b}_1 &= f_{b_1}\left(p, b_1, v, \Omega\right),\\
\dot{\Omega} &= f_{\Omega}\left(\dot{r}, Q_e, Q_{mr}, Q_{tr}\right).
\end{aligned}
\tag{1}
$$

Equation 1 describes a non-linear time invariant system, where X, Y, Z, L, M, N and Q are also non linear functions of other states. The equations for linear acceleration $(\dot{u}, \dot{v}, \dot{w})$ rely on the components of acceleration determined by gravity, roll, pitch and yaw angular velocities, and the effects of axial forces exerted by the main rotor, air pressure on fuselage, stabilizers and tail. Equations for angular acceleration $(\dot{p}, \dot{q}, \dot{r})$ in Eq. 1 are a simplification of the inertia tensor products implemented in the final model. Angular accelerations depend on moments produced by the same forces in linear acceleration plus the engine drive.

The four basic forces of flight: lift, drag, thrust and weight are represented in Eq. 1 by forces (X, Y, Z) and moments (L, M, N), and their magnitudes depend on both internal properties and external effects acting upon the vehicle. Lift and drag are mainly related to wind and aerodynamic characteristics of the helicopter surfaces. Thrust and weight depend on the engine, rotor and helicopter mass. For this model, it is assumed that the fuselage center of pressure coincides with the center of gravity; therefore moments created by the fuselage aerodynamic forces are neglected. Equations for forces related to the main rotor and engine and a description of fuselage and tail forces and moments have their own expressions and are detailed in [1, 8]. Higher-order effects (lateral and longitudinal tip-path-plane flapping) are taken into account in equations for a_1 and b_1 to improve the rigid-body model accuracy. The coupled rotor and stabilizer bar equations are lumped into one first-order equation of motion. Further details about these equations can be found in [8, 18].

4 Control Architecture

Complex systems typically possess a hierarchical structure, characterized by a mixture of both continuous dynamics at the lower levels and logical decision-making at the higher levels. An autonomous helicopter is hybrid because it exhibits continuous and discrete dynamics in the form of events determined by internal and external sources. A hybrid controller depends on discrete phenomena and each discrete state may correspond to independent continuous states, dynamics and controllers. For the problem of autonomous control of the helicopter, the entire system will be treated as a finite automaton and the continuous state space will be partitioned in different operation modes. The transitions are started when one continuous state satisfies certain conditions as, for example, reaching a waypoint.

The control system architecture for the autonomous small helicopter is presented in Fig. 2. Every flight of the autonomous vehicle starts from a mission assigned by a human operator who indicates, to a computer mission planner, a route to follow (**X**). The mission planner converts the waypoint-defined trajectory and actions into a smooth description with a set of properties for each waypoint. The mission planner provides the finite state machine (that runs in the flight computer) with a routine to follow (**P**). The FSM will make decisions regarding the best way to execute each routine, selecting continuous controllers and discrete time actions. The control system carries out the tasks assigned by the FSM ($\mathbf{x}_{fsm}$), executing the final actions (δ) based on the current Kalman-filter estimated state $\hat{\mathbf{x}}$ [21]. At any moment a safety pilot can take control of the vehicle (δ_{man}) activating a switching signal (m_a).

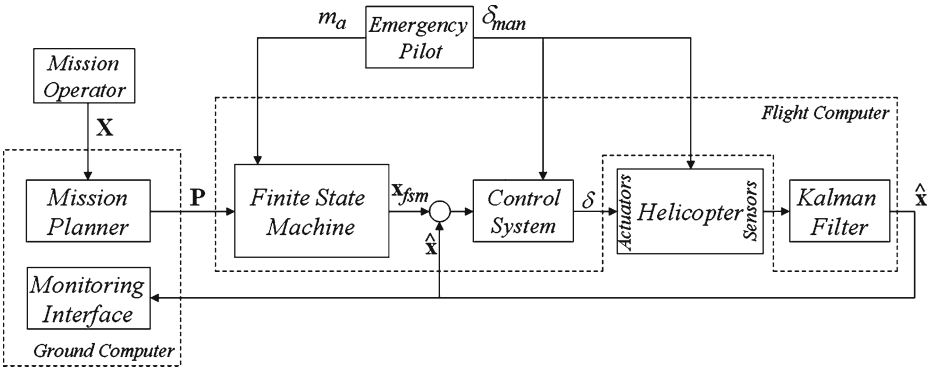

Fig. 2 Control system architecture

5 Trajectory Smoothing

The helicopter flight plan is transformed into a smooth trajectory in order to reduce direction and mode change transients. This trajectory is rendered in flight by a finite state machine, able to infer flight mode changes and, in general, to manage the autonomous flight. The trajectory is described as a sequence of points assigned by a flight operator (Fig. 3). The series of flight waypoints (WP) is referred as the vector sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i \ldots, \mathbf{x}_{n+1}\}$, where $\mathbf{x}_i = \langle x_i, y_i, z_i, v_i \rangle$ contains a three-dimensional point and a speed value v_i indicating speed at the WP. If two successive WPs have the same speed values, the trajectory within those points is traversed at constant speed. If two consecutive WPs have different speed values, the trajectory smoothing algorithm and FSM must provide a speed transition to meet the specified speeds from the starting to end WP. A hover WP is indicated as $\langle x_i, y_i, z_i, 0 \rangle$ and

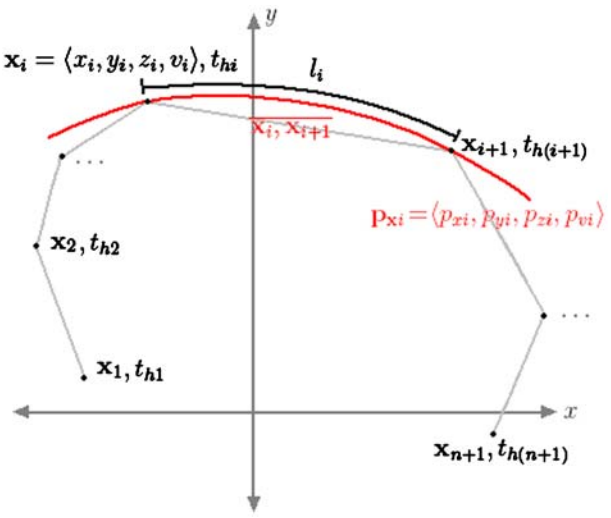

Fig. 3 Waypoint and trajectory notation

a separate sequence $T_h = \{t_{h1}, t_{h2}, \ldots, t_{hi} \ldots, t_{h(n+1)}\}$ corresponding to the hover time for each WP (zero if a speed waypoint) has to be specified. On this definition, two successive waypoints cannot be zero. The z_i component of $\mathbf{x}_i$ is constant, except at z_1: the startup ground position.

A piecewise polynomial vector describing the smooth trajectory and speed on segment i is denoted by $\mathbf{p}_{\mathbf{x}i}(u) = \langle p_{xi}(u), p_{yi}(u), p_{zi}(u), p_{vi}(u) \rangle$, where any of the elements is a cubic polynomial $\mathbf{p}_{\mathbf{x}i}(u) = \mathbf{a}_i u^3 + \mathbf{b}_i u^2 + \mathbf{c}_i u + \mathbf{d}_i$. The entire set of polynomial vectors is referred as:

$$\mathbf{P} = \{\mathbf{p}_{\mathbf{x}1}, \mathbf{p}_{\mathbf{x}2}, \ldots, \mathbf{p}_{\mathbf{x}i}, \mathbf{p}_{\mathbf{x}(i+1)}, \ldots, \mathbf{p}_{\mathbf{x}n}\}.$$

The length of arc segment i is denoted by l_i and the time to traverse it t_i. The fundamental sample time is denoted T_s.

A spline is a polynomial function defined by very simple disjoint subsets of its domain but having the characteristic of globally being smooth [2]. Of particular interests are the Catmull–Rom splines [5]: a subset of Cardinal splines and Hermite splines. Hermite splines pass through all data points, something desired in the helicopter route. Let $\mathbf{p}_{\mathbf{x}i}(u) = \mathbf{p}_{\mathbf{x}}(u)\big|_{u_i}^{u_{i+1}}$ be a Cardinal spline (Fig. 4) with point restrictions for u_i and u_{i+1}:

$$\begin{aligned} \mathbf{p}_{\mathbf{x}i}(u_i) &= \mathbf{x}_i, \\ \mathbf{p}_{\mathbf{x}i}(u_{i+1}) &= \mathbf{x}_{i+1}, \\ \mathbf{p}'_{\mathbf{x}i}(u_i) &= \alpha(\mathbf{x}_{i+1} - \mathbf{x}_{i-1}), \\ \mathbf{p}'_{\mathbf{x}i}(u_{i+1}) &= \alpha(\mathbf{x}_{i+2} - \mathbf{x}_i). \end{aligned} \tag{2}$$

Here α is a tension parameter, which determines how strained the curve will be. For simplicity let $u_i = 0$ and $u_{i+1} = 1$ (as it can be verified this does not affect the final result). The restrictions of the segment are now,

$$\begin{aligned} \mathbf{p}_{\mathbf{x}i}(0) &= \mathbf{x}_i, \\ \mathbf{p}_{\mathbf{x}i}(1) &= \mathbf{x}_{i+1}, \\ \mathbf{p}'_{\mathbf{x}i}(0) &= \alpha(\mathbf{x}_{i+1} - \mathbf{x}_{i-1}), \\ \mathbf{p}'_{\mathbf{x}i}(1) &= \alpha(\mathbf{x}_{i+2} - \mathbf{x}_i). \end{aligned}$$

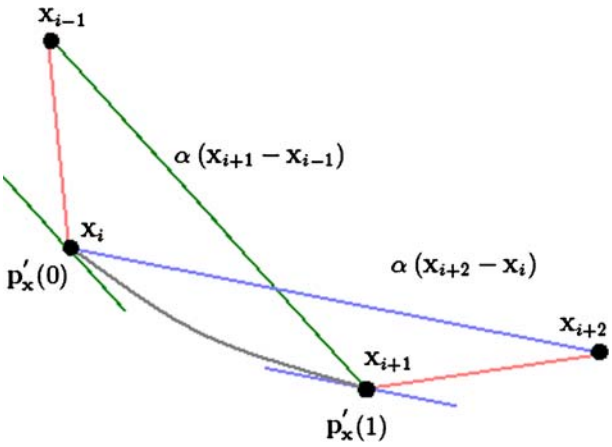

Fig. 4 Cardinal spline notation

Vector parameters $\mathbf{a}_i$, $\mathbf{b}_i$, $\mathbf{c}_i$, $\mathbf{d}_i$ can be found comparing the restrictions equation with the general equation of a cubic curve:

$$\begin{aligned}
\mathbf{p}_{\mathbf{x}i}(0) &= \mathbf{d}_i, \\
\mathbf{p}_{\mathbf{x}i}(1) &= \mathbf{a}_i + \mathbf{b}_i + \mathbf{c}_i + \mathbf{d}_i, \\
\mathbf{p}'_{\mathbf{x}i}(0) &= \mathbf{c}_i, \\
\mathbf{p}'_{\mathbf{x}i}(1) &= 3\mathbf{a}_i + 2\mathbf{b}_i + \mathbf{c}_i.
\end{aligned}$$

For the two dimensional case, as the tension parameter α approaches zero the tension in each knot is higher, and the spline becomes a polygon. Catmull–Rom splines are those Cardinal splines where $\alpha = 1/2$.

A particular property of the shape of $p_{vi}(u)$ is the behavior of the function near zero. Near zero speed values imply discontinuities in the time calculation integral described below. This problem was solved replacing zero speeds in the calculation with a small value ($v_\varepsilon = 0.01$ m/s $= 1$ cm/s), improving the stability on the time calculations and reducing unnecessarily long trajectory times. In the FSM, values near v_ε are treated as zero in any state and in consequence zero speed maneuvers (e.g. hover) do not accumulate error. A second property of $p_{vi}(u)$ is that it might not necessary share the same smoothness of the trajectory. For example it was found in simulation that a linear velocity transition ($a_{vi} = b_{vi} = 0$) was enough in most cases.

Two important attributes of the smooth trajectory are its length and traversal time. Length of the trajectory is required by the FSM to produce the final trajectory. Knowing the traversal time guarantees the helicopter will not surpass the maximum time allowed by fuel or battery constraints. The length o the two-dimensional trajectory for segment i can be found with the arc length function for parametric curves:

$$l_i = \int_0^1 \sqrt{p'_{xi}(u)^2 + p'_{yi}(u)^2}\,\mathrm{d}u. \tag{3}$$

Integral of Eq. 3 is in general not solvable analytically for second order polynomials, so the Gauss–Legendre integration method is suggested [10]. Gauss–Legendre integration is preferred due to its accuracy for polynomial functions and exactness for up to $(2N_{gl} - 1)$ degree polynomials, with N_{gl} the number of Gauss–Legendre points [15]. The general N_{gl}-point Gauss–Legendre rule, which is exact for polynomials of degree $\leq (2N_{gl} - 1)$, is:

$$F_{N_{gl}}(f) = w_1 f(u_1) + w_2 f(u_2) + \cdots + w_N f(u_N)$$

Values u_k and weights w_k have been tabulated and are available in the literature. The values of the eight-point method were used for a better approximation. Considering that the Gauss–Legendre formula is defined for the interval $[-1, 1]$, the mapping $u = (1 + u_k)/2$ is applied to convert the values to the interval $[0, 1]$. The length is then found with the approximation:

$$\begin{aligned}
l_i &= \int_0^1 \sqrt{p'_x(u)^2 + p'_y(u)^2}\,\mathrm{d}u \\
&\approx \frac{1}{2}\sum_{k=1}^{8} w_k \sqrt{p'_x\left(\frac{1+u_k}{2}\right)^2 + p'_y\left(\frac{1+u_k}{2}\right)^2}
\end{aligned}$$

Table 1 Numerical results of length for a waypoints trajectory with 360 points

Algorithm	Result
Gauss–Legendre	62.8318530735243
Numerical parallelogram	62.8318530622568
Real ($l = 2\pi r$)	62.8318530717959

Precision of l_i is fundamental for trajectory rendering as it reduces irregularities near the knots. Numerical tests were made to the Gauss–Legendre method with a circular trajectory of 10 m radius and 360 equally spaced points. Three different integration methods were compared, with the results shown in Table 1. The Gauss–Legendre formula reached excellent numerical precision with only eight iterations compared to the parallelogram method with 200 iterations.

Estimated traversal time can be obtained from the trajectory length, the speed function and the basic relation $t = x/v$. Intuitively one could argue that traversal time depends on the trajectory shape given by polynomials $p_x(u)$ and $p_y(u)$; however, as the speed vector is always tangent to the trajectory, the change in speed over the curve is equivalent to a particle following a line of length l_i. Trajectory traversal time is then estimated with:

$$t_i = \int_0^{l_i} \frac{1}{p_{vi}(u)} du. \tag{4}$$

As continuity of $1/p_{vi}(u)$ is an issue due to the initial and hover WPs, the constant $v_\varepsilon = 0.01$ was used ensuring that always $p_{vi}(u) \geq v_\varepsilon$. The Gauss–Legendre method was tested to calculate time but results were very inaccurate due to instability of the algorithm at low speeds. A simple numerical parallelogram method was used instead with excellent results (around 1 to 2 s of error for long trajectories of 15 min).

As described in the control architecture, a gain scheduler helps the FSM and control system to move smoothly from a set of controller gains to another. This feature was implemented in as an array of discrete time Proportional-Integral-Derivative (PID) controllers with online tuning parameters (K_p, T_i, T_d, K_d), where K_d is a relaxation coefficient for the derivative action. Once the gain scheduler detects a mode transition, values (K_p, T_i, T_d, K_d) for each controller are smoothly changed to new values, creating a new controller. The smoothing method used was a Hermite spline with zero tangents, equivalent to a Cardinal spline with tension value $\alpha = 0$. This spline was chosen due to a fewer number of operations and flat shape at the endpoints. Parameter transition needs to be performed for four parameters on six controllers during flight (24 polynomials need to be found).

6 The Finite State Machine

The main function of the Finite State Machine is to render the trajectory produced by the smoothing algorithm and to perform event driven actions during each operation mode change. Event driven actions include default behavior once a waypoint is reached, executing takeoff and landing maneuvers, maintaining hover for a defined

time, and performing safety operations on mode changes such as autonomous to manual transfer. The states of the FSM were deduced from observation of manned flights and the autonomous helicopter flight modes proposed in [22]. The FSM was implemented on Stateflow (http://www.mathworks.com/products/stateflow) and is presented on Fig. 5. A Discrete Event System FSM [28] is a 9-tuple:

$$M_d = \langle \Sigma, \Delta, S, \delta_{int}, \delta_{ext}, \lambda, s_0, \Gamma, t_a \rangle , \tag{5}$$

with Σ the set of input values; Δ the set of output values; S the set of states; $\delta_{int} : S \times \Gamma \rightarrow S \times \Gamma$ a function of internal transitions; $\delta_{ext} : S \times \Gamma \times \Sigma \rightarrow S \times \Gamma$ a function of external transitions; $\lambda : S \times \Gamma \times \Sigma \rightarrow \Delta$ the output function (triggered after entering any state); s_0 the initial state; Γ the set of internal variables; and $t_a : S \rightarrow \mathbb{R}_{0,\infty}$ the time advance function. In any given moment the system is in a state of $S \times \Gamma$. If an external event (e.g. changing from autonomous to manual mode) does not occur, the system will remain in the state s for time $t_a(s)$, $s \in S$. In this sense, $t_a(s)$ can be interpreted as the time the system remains in a given state s if there are no external events. When the waiting time finishes, that is, $t_a(s)$ reaches a predefined time T_a, the system produces the output $\lambda(s, \gamma)$, $\gamma \in \Gamma$ and proceeds to state $\delta_{int}(s, \gamma)$. If an external event $\sigma \in \Sigma$ occurs before the expiration time, the system changes to state $\delta_{ext}(s, \gamma, \sigma)$. This means that the external transition function determines the new state

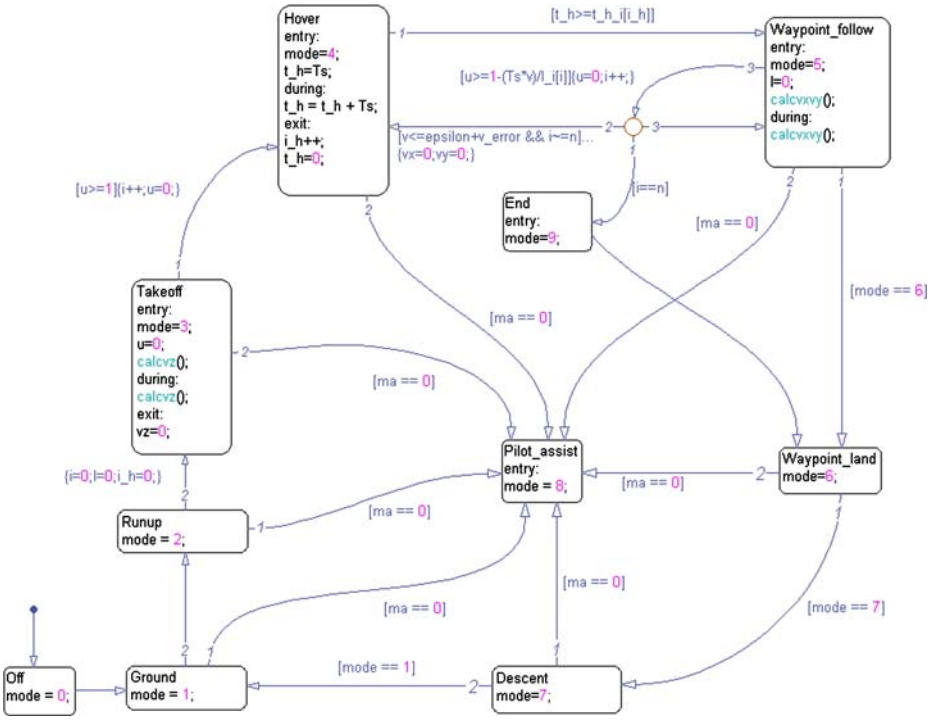

Fig. 5 Finite State Machine implementation

of the system when there is an external event. The sets for the FSM are formally defined as:

$$\Sigma = \left\{ \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi} \right\rangle \middle| \, m_a \in \{0, 1\}, \mathbf{p}_{\mathbf{x}i} \in \mathbf{P}, n \in \mathbb{Z}_{1,K}, l_i \in L, t_{hi} \in T_h \right\},$$

$$\Delta = \left\{ \begin{array}{c} \left\langle \mathbf{x}_{fsm}, \mathbf{v}_{fsm}, \psi, m_0 \right\rangle \middle| \, \mathbf{x}_{fsm} \in \mathbb{R}^3, \mathbf{v}_{fsm} \in \mathbb{R}^3_{v_\varepsilon, v_{\max}} \\ \psi \in [-\pi, \pi], m_0 \in \{0, 1, \ldots, 9\} \end{array} \right\},$$

$$S = \left\{ s \middle| \, s \in \left\{ \begin{array}{c} \text{Off, Ground, Runup, Takeoff, Hover,} \\ \text{WP\_follow, WP\_land,} \\ \text{Descent, Pilot\_assist, End} \end{array} \right\} \right\},$$

$$\Gamma = \left\{ \begin{array}{l} \langle i, l, t_h, u, v \rangle \middle| \, i \in \mathbb{Z}_{1,n}, l \in \mathbb{R}_{1,\max L}, \\ t_h \in \mathbb{R}_{1,\max T_h}, u \in [0, 1), v \in \mathbb{R}_{v_\varepsilon, v_{\max}} \end{array} \right\}.$$

Input values on Σ are the structure produced by the smoothing algorithm and notation corresponds exactly as it was specified on Section 5. Variable m_a is the manual/autonomous mode flag received from the pilot, n is the cardinality of $\mathbf{P}$, L is a set of arc lengths, and T_h is a set of hover times. Variable m_a is deactivated (0) by the safe pilot of the helicopter in case he or she wanted to take control of the vehicle. The subscript on the integers $\mathbb{Z}_{1,K}$ indicates a finite subset of $\mathbb{Z}$, in this case n can be any integer from 1 to a finite number K. Output values in the set Δ are the commanded control system position ($\mathbf{x}_{fsm} = \langle x, y, z \rangle$), 3D speed ($\mathbf{v}_{fsm} = \langle v_x, v_y, v_z \rangle$), orientation ψ, and an integer value m_0 that informs the control system the current operation mode.

Functions δ and λ are different for each state and will be presented in the next section. Function t_a is defined as $t_a(s) = T_s, \forall s \in S$, with T_s the constant sample time. The only source of internal transitions is then the system clock, agreeing with the continuous sampled nature of the system.

6.1 Off, Ground and Runup States

Off, *Ground* and *Runup* states are special initialization modes added to the machine to perform start-up routines prior to flight. Only the transitions of these states will be described. The *Off* state main goal is to reset the initial internal values of the machine. The *Off* state also acts as the initial state:

$$s_0 = \text{Off}.$$

Initial values are fundamental for initialization of the state estimation algorithm (i.e. Kalman filter). The output function defined for the *Off* state is:

$$\lambda\left(\text{Off}, \langle i, l, t_h, u, v \rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi} \right\rangle\right) = \left\langle \mathbf{p}_{\mathbf{x}1}(0), \mathbf{0}, \arctan\left(\left. \frac{dp_{y1}(u)/du}{dp_{x1}(u)/du} \right|_0 \right), 0 \right\rangle.$$

Here the *arctan* function produces the initial orientation from derivatives of the first polynomial evaluated at the starting point. The *Off* state is a transitory state and is immediately followed by the transition:

$$\delta_{int}(\text{Off}, \langle i, l, t_h, u, v \rangle) = (\text{Ground}, \langle 1, 0, 0, 0, 0 \rangle).$$

The *Ground* state is intended to perform ground actions just after the helicopter engine is started. This state does not change actual output, so the corresponding function can be defined as:

$$\lambda\left(\text{Ground}, \langle i, l, t_h, u, v\rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle\right) = \left\langle \mathbf{x}_{fsm}, \mathbf{v}_{fsm}, \psi, 1\right\rangle.$$

After *Ground* initialization, the following transition will take place,

$$\delta_{int}\left(\text{Ground}, \langle i, l, t_h, u, v\rangle\right) = \left(\text{Runup}, \langle i, l, t_h, u, v\rangle\right),$$

unless the external pilot obtains control producing the state change,

$$\delta_{ext}\left(\text{Ground}, \langle i, l, t_h, u, v\rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle\right) = \begin{cases} \left(\text{Pilot\_assist}, \langle i, l, t_h, u, v\rangle\right) & m_a = 0 \\ \left(\text{Ground}, \langle i, l, t_h, u, v\rangle\right) & m_a \neq 0 \end{cases}.$$

The *Runup* state main objective is to provide the necessary setup to allow the engine to achieve takeoff RPMs. Its output function is defined as:

$$\lambda\left(\text{Runup}, \langle i, l, t_h, u, v\rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle\right) = \left\langle \mathbf{x}_{fsm}, \mathbf{v}_{fsm}, \psi, 2\right\rangle,$$

and it is immediately followed by the *Takeoff* state:

$$\delta_{int}\left(\text{Runup}, \langle i, l, t_h, u, v\rangle\right) = \left(\text{Takeoff}, \langle i, l, t_h, u, v\rangle\right),$$

unless

$$\delta_{ext}\left(\text{Runup}, \langle i, l, t_h, u, v\rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle\right) = \begin{cases} \left(\text{Pilot\_assist}, \langle i, l, t_h, u, v\rangle\right) & m_a = 0 \\ \left(\text{Runup}, \langle i, l, t_h, u, v\rangle\right) & m_a \neq 0 \end{cases}.$$

6.2 Takeoff State

The *Takeoff* state main purpose is to lift the helicopter to a predefined flight altitude whilst slowly transitioning from zero to takeoff speed and then back to zero. *Takeoff* is the first state to actually produce a trajectory and to determine the control system input at every sample time. Trajectory generation on *Takeoff* produces values on the z axis and smoothly changes speed as specified by the smoothing algorithm. The position described by $p_{z1}(u)$ and the speed described by $p_{v1}(u)$ are then rendered. From traditional definition of position and speed functions, it is possible to expect that time is the independent variable on p_{z1} and p_{v1}, and that u is directly related to time progression. This however will imply that speed is defined by p'_{z1}, ignoring p_{v1}. The unifying term is instead arc length l. Arc length is nicely related to speed with the following relation:

$$l(u) = \int_0^u p_{vi}(\omega)\, d\omega.$$

So, u can be easily mapped from $u = l/l_i$, with $l \in [0, l_i)$ for segment i and segment length l_i. As distance progresses in the curve, the position and orientation can be obtained from $\langle p_{xi}(l/l_i), p_{yi}(l/l_i)\rangle$. Moreover, finding the current speed $p_{vi}(u)$ requires u, that at the same time requires l, and l requires $p_{vi}(u)$ to be found. This can be solved with the following iterative scheme in the discrete case:

$$l(u_{k+1}) = l(u_k) + T_s p_v(u_k),$$

with

$$u_{k+1} = \frac{l\left(u_{k+1}\right)}{l_i}$$

and knowing that always $p_v > 0$.

The output function for the *Takeoff* state can then be defined as:

$$\lambda\left(\text{Takeoff}, \langle i, l, t_h, u, v\rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle\right) = \left\langle \mathbf{x}_{takeoff}\left(l, u, i\right), \mathbf{v}_{takeoff}\left(u, i\right), \psi, 3\right\rangle.$$

With functions $\mathbf{x}_{takeoff}$, $\mathbf{v}_{takeoff}$ defined as:

$$\mathbf{x}_{takeoff}\left(l, u, i\right) = \mathbf{p}_{\mathbf{x}i}\left(\frac{l + T_s p_{vi}\left(u\right)}{l_i}\right),$$

$$\mathbf{v}_{takeoff}\left(u, i\right) = \left\langle 0, 0, p_{vi}\left(u\right)\right\rangle.$$

Note that every time the state is reentered, λ is recalled and the trajectory keeps being produced until the condition $u \geq 1$ is met. After this occurs the δ_{int} function exits the state. For takeoff, $\mathbf{p}_{xi}$ is defined by the smoothing algorithm only as a z trajectory, with x, y held constant. Variable ψ also remains constant during the takeoff.

Internal states on *Takeoff* are updated with the function,

$$\delta_{int}\left(\text{Takeoff}, \langle i, l, t_h, u, v\rangle\right) = \begin{cases} \left(\text{Takeoff}, \left\langle \begin{array}{c} i, l + T_s p_v\left(u\right), t_h, \\ \dfrac{l + T_s p_v\left(u\right)}{l_i}, p_v\left(u\right) \end{array} \right\rangle\right) & u < 1 \\ \left(\text{Hover}, \langle i + 1, l, t_h, 0, v\rangle\right) & u \geq 1 \end{cases}$$

Unless the external event occurs and δ_{ext} is executed:

$$\delta_{ext}\left(\begin{array}{c} \text{Takeoff}, \langle i, l, t_h, u, v\rangle, \\ \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle \end{array}\right) = \begin{cases} \left(\text{Pilot\_assist}, \langle i, l, t_h, u, v\rangle\right) & m_a = 0 \\ \left(\text{Takeoff}, \langle i, l, t_h, u, v\rangle\right) & m_a \neq 0 \end{cases}.$$

6.3 Hover State

Every time the *Hover* state is entered or re-entered the output produced is:

$$\lambda\left(\text{Hover}, \langle i, l, t_h, u, v\rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle\right) = \left\langle \mathbf{p}_{\mathbf{x}i}\left(0\right), \mathbf{0}, \psi, 4\right\rangle.$$

Meanwhile, internal states are updated and hovering time is counted,

$$\delta_{int}\left(\text{Hover}, \langle i, l, t_h, u, v\rangle\right) = \begin{cases} \left(\text{Hover}, \langle i, l, t_h + T_s, u, 0\rangle\right) & t_h < t_{hi} \\ \left(\begin{array}{c} \text{WP\_follow}, \\ \langle i + 1, l, 0, u, 0\rangle \end{array}\right) & t_h \geq t_{hi} \end{cases},$$

Unless it is interrupted,

$$\delta_{ext}\left(\text{Hover}, \langle i, l, t_h, u, v\rangle, \left\langle m_a, \mathbf{p}_{\mathbf{x}i}, n, l_i, t_{hi}\right\rangle\right) = \begin{cases} \left(\text{Pilot\_assist}, \langle i, l, t_h, u, v\rangle\right) & m_a = 0 \\ \left(\text{Hover}, \langle i, l, t_h, u, v\rangle\right) & m_a \neq 0 \end{cases}.$$

6.4 WP_follow State

The *WP_follow* (waypoint follow) state progression is similar to the *Takeoff* state. The main difference is that $\mathbf{x}_{fsm}$ remains constant in z and changes in x, y. Each time the state is entered or t_a expires, output of *WP_follow* is determined by

$$\lambda\left(\begin{array}{c}\text{WP\_follow},\langle i,l,t_h,u,v\rangle,\\ \langle m_a,\mathbf{p}_{\mathbf{x}i},n,l_i,t_{hi}\rangle\end{array}\right)=\left\langle \mathbf{x}_{flw}\left(i,l,u\right),\mathbf{v}_{flw}\left(i,u,\psi_{flw}(i,u)\right),\psi_{flw}\left(i,u\right),5\right\rangle$$

With functions $\mathbf{x}_{flw}$, $\mathbf{v}_{flw}$, ψ_{flw} defined as

$$\begin{aligned}\mathbf{x}_{flw}\left(i,l,u\right)&=\mathbf{p}_{\mathbf{x}i}\left(u_{new}\right)\\ \mathbf{v}_{flw}\left(i,u,\psi_f\right)&=\left\langle p_{vi}\left(u\right)\cos\left(\psi_f\right),\,p_{vi}\left(u\right)\sin\left(\psi_f\right),0\right\rangle\\ \psi_{flw}\left(i,u\right)&=\arctan\left(\frac{p_{yi}\left(u_{new}\right)-p_{yi}\left(u\right)}{p_{xi}\left(u_{new}\right)-p_{xi}\left(u\right)}\right)\\ u_{new}&=\frac{l+T_s p_{vi}\left(u\right)}{l_i}\end{aligned}$$

After every wakeup, internal states are then updated with the following conditions,

$$\delta_{int}\big(\text{WP\_follow},\langle i,l,t_h,u,v\rangle\big)=\begin{cases}\big(\text{Hover},\langle i+1,l,t_h,0,v\rangle\big)\\ \text{If } u\geq 1-\dfrac{T_s v}{l_i}\wedge v\leq v_\varepsilon\wedge i\neq n\\ \big(\text{End},\langle i+1,l,t_h,0,v\rangle\big)\\ \text{If } u\geq 1-\dfrac{T_s v}{l_i}\wedge v> v_\varepsilon\wedge i= n\\ \big(\text{WP\_follow},\langle i+1,l,t_h,0,v\rangle\big)\\ \text{If } u\geq 1-\dfrac{T_s v}{l_i}\wedge v> v_\varepsilon\wedge i\neq n\\ \left(\text{WP\_follow},\left\langle\begin{array}{c}i,l+T_s p_{vi}\left(u\right),\\ t_h,u_{new},p_{vi}\left(u\right)\end{array}\right\rangle\right)\\ \text{If } u< 1-\dfrac{T_s v}{l_i}\end{cases}$$

Note the term $\left(1-\frac{T_s v}{l_i}\right)$ is extremely important to a avoid discontinuities at the end of the polynomial preventing u from leaving the interval $[0,1)$. The *WP_follow* state can be interrupted if

$$\delta_{ext}\left(\text{Hover},\langle i,l,t_h,u,v\rangle,\langle m_a,\mathbf{p}_{\mathbf{x}i},n,l_i,t_{hi}\rangle\right)=\begin{cases}\big(\text{Pilot\_assist},\langle i,l,t_h,u,v\rangle\big) & m_a=0\\ \big(\text{Hover},\langle i,l,t_h,u,v\rangle\big) & m_a\neq 0\end{cases}$$

6.5 Pilot_assist, Waypoint_land, Descent and End States

The main function of the *Pilot_assist* mode is to include additional logic to the assisted flight mode transition. Most requirements for this transition are implemented in the control system PIDs as bumpless transfer. The helicopter autonomous landing

process will not be described but is similar in principle to the *Takeoff* state. The *Descent* state is required to perform special actions before landing, such as guidance by an independent sensor (e.g. a sonar). The *End* state is a transitory state that indicates the end of the trajectory.

7 Simulation Results

The methods proposed were implemented and tested in simulated flight. The main goal of the simulation tests was to check if the trajectory generator worked as expected and the effect of transients was reduced. Various trajectories were generated and multidimensional error comparisons across length, shape and speed were made. Non smoothed versions of the trajectory position and speed were compared with smoothed positions and semi-smoothed and smoothed speed schedules. All trajectories included mode transitions from takeoff to hover, hover to forward flight and in some cases forward flight to hover. In all tests, a wind perturbation of 1 m/s in a randomly changing direction was used. Overall, experimentation showed a major benefit from using the proposed smoothing method and the FSM. Control error, and especially attitude control error, was reduced on the smoothed trajectories. An important conclusion from simulation tests was that simpler speed transitions (not smooth but just linearly progressive) resulted in less error in most cases.

Formally, the error was defined as follows. Let the helicopter state vector sampled in the k-th instant be $\mathbf{x}(k)$ and let $\mathbf{x}_{fsm}(k)$ be the expected state determined by the smooth trajectory and FSM. The state error at moment k is the difference

$$\mathbf{e}_{\mathbf{x}}(k) = \left|\mathbf{x}(k) - \mathbf{x}_{fsm}(k)\right|. \tag{6}$$

The entire trajectory average error is defined as

$$\mathbf{E}_{\mathbf{x}} = \sum_{k=1}^{m} \frac{\mathbf{e}_{\mathbf{x}}(k)}{m}. \tag{7}$$

With m is the total number of samples in the entire flight.

7.1 Linear Trajectory and Baseline Error

A basic initial test involving takeoff, a short hover and a simple constant speed linear trajectory was executed to obtain the baseline average control error. A constant speed, straight trajectory was conceived as a good measure of base error because it does not include any trajectory smoothing and just one speed change (from zero to a constant speed value). The same simulation was executed for three different speeds (1, 3 and 5 m/s) and five smoothing configurations. The five configurations were: (1) a non smooth trajectory and a non smooth speed (*nn*) schedule; (2) a non smooth trajectory and a ramp-like speed (*nr*) schedule; (3) a smooth trajectory and a non smooth speed (*sn*) schedule; (4) a smooth trajectory and a ramp-like speed (*sr*) schedule; and (5) a smooth trajectory and a smooth speed (*ss*) schedule. A "non smooth trajectory" is simply a trajectory formed by lines connecting the waypoints with no smoothing applied. In a non smooth speed schedule, once the vehicle reaches a waypoint the control system speed reference is changed immediately. A ramp-like

speed establishes a speed change that progresses from one speed value to another linearly.

The control system commands (δ_{col}, δ_{lon}, δ_{lat}, δ_{ped}) were recorded and their standard deviation and absolute maximum were calculated as an indicator of control effort. The Kalman filter was disabled for this and subsequent tests to limit extrinsic effects. Estimation was not necessary as the full state is known in simulation.

Figure 6 presents the average error obtained in the test for different speeds. For further comparison and as a baseline error for the rest of plots, the average error for the best (*sr* at 1 m/s) and worst (*nn* at 5 m/s) configuration is be kept as a shadowed area. A value on the first row of graphics of Fig. 6 indicates the average positional error and can be interpreted as: "during the entire trajectory, on average, the helicopter was $\mathbf{E_x}$ meters away from its intended position". The same applies for the second and third row of graphics.

Figure 6 shows how, increasing speed affects control error. This effect was expected as faster dynamics make it harder for to the control system to maintain

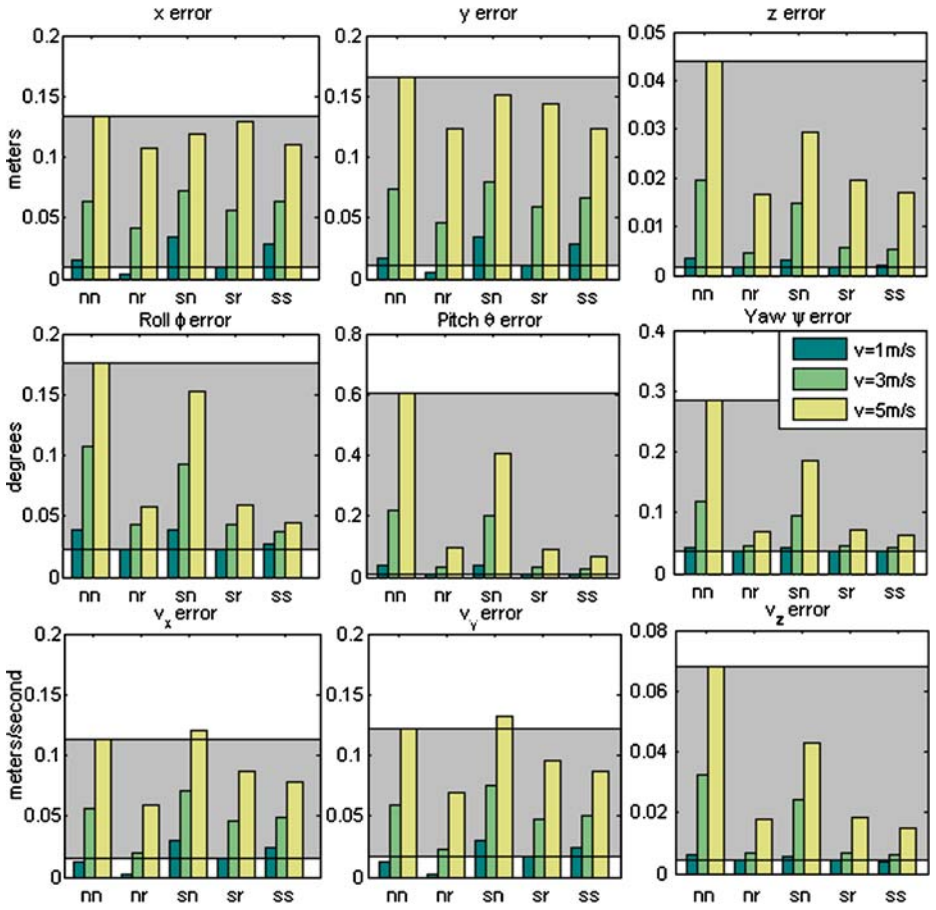

Fig. 6 Linear trajectory average error E_x on different smoothness configurations (*nn*, *nr*, *sn*, *sr*, *ss*) and speeds (1, 3, 5 m/s) for position (x, y, x), attitude (φ, θ, ψ) and speed (v_x, v_y, v_z)

stable flight. Lower errors are obtained in ramp-like speed changes at slow speed (recall, the only change performed in this case is from zero to 1, 3 or 5 m/s at the beginning of flight). Results from the non-smooth trajectory/ramp-like speed (*nr*) and smooth trajectory/ramp-like speed (*sr*) could be expected to be equal, as trajectory smoothing is not necessary. That is, however, not the case and non smooth trajectories exhibit lower errors (i.e. x, y, v_x, v_y). This effect was found to be caused by the behavior of the smoothing algorithm on the first segment of the trajectory. Note that although the trajectory $\langle \mathbf{p}_{xi}(u), \mathbf{p}_{yi}(u) \rangle$ could parametrically produce a line, it does not imply that alone $\mathbf{p}_{xi}$ and $\mathbf{p}_{yi}$ are lines. Producing the initial section on the Catmull–Rom algorithm requires the vector $[\mathbf{x}_1\ \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$, so a proper tangent is not used but an approximation, producing a small curvature on the first segment.

Figure 7 displays the control system output standard deviation and absolute maximums. Larger deviations at the control system output represent more effort maintaining the reference value. Again, higher speeds represent larger effort and more extreme control values on the actuators. Similarly to the plot of average error, a gray shaded area is used here to show the lowest (*sr*, 1 m/s) and highest results (*nn*, 5 m/s).

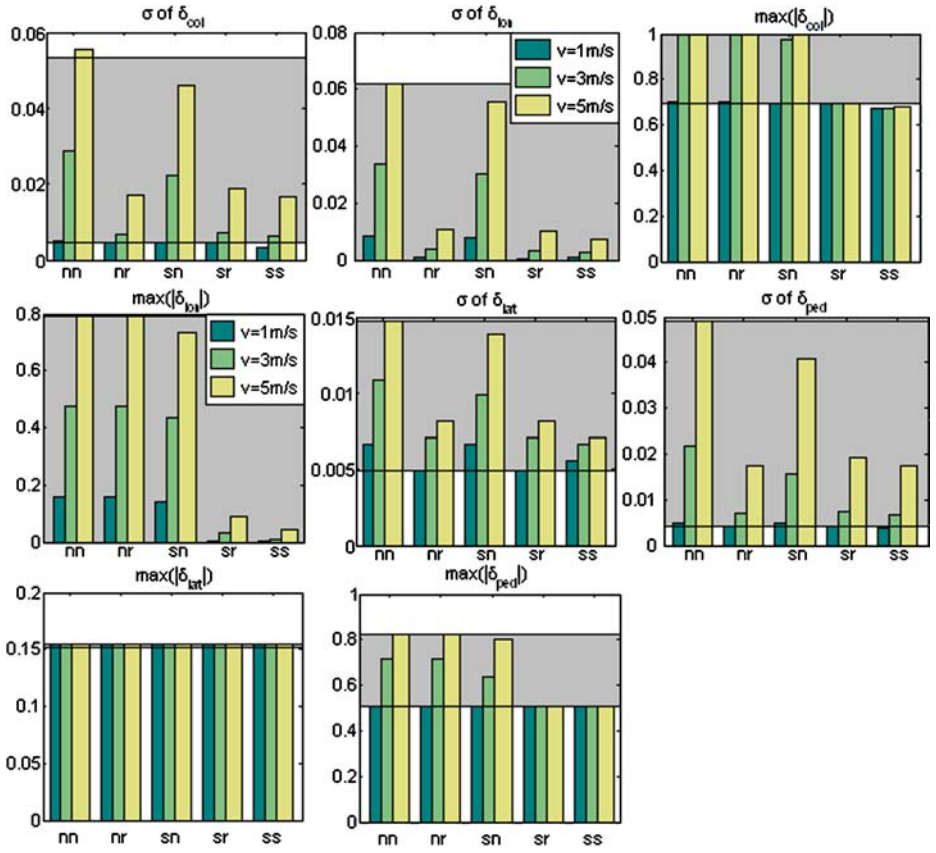

Fig. 7 Linear trajectory control output standard deviation (σ) and absolute maximum values for each vehicle input (δ_{col}, δ_{lon}, δ_{lat}, δ_{ped})

7.2 Rectangular and Circular Trajectories

Regular trajectories were used to measure benefits of trajectory smoothing in soft to sharp direction changes with speed increments. For the rectangular trajectories, squares of side 10, 30 and 50 m with speeds 0, 1, 2, 1 m/s at the vertices were used. On the circular trajectories, 12 points chosen on a circle with speeds ranging from 0 to 3.5 m/s were assigned at the joints. Three of these trajectories were generated for circles with radius 10, 30 and 50 m and the results compared.

Figure 8 presents the average control error for each state variable at the rectangular trajectories. For all the cases, larger trajectories imply lower error. Larger trajectories allow more space for speed transitions and, in consequence, the control system preserves easier the reference value. Ramp-like speed changes again show lower positional and speed errors. Smooth trajectories however reduce attitude error in all instances. Non smooth trajectories cause large yaw errors of 10–20° even on progressive speed changes, which is due to abrupt direction changes.

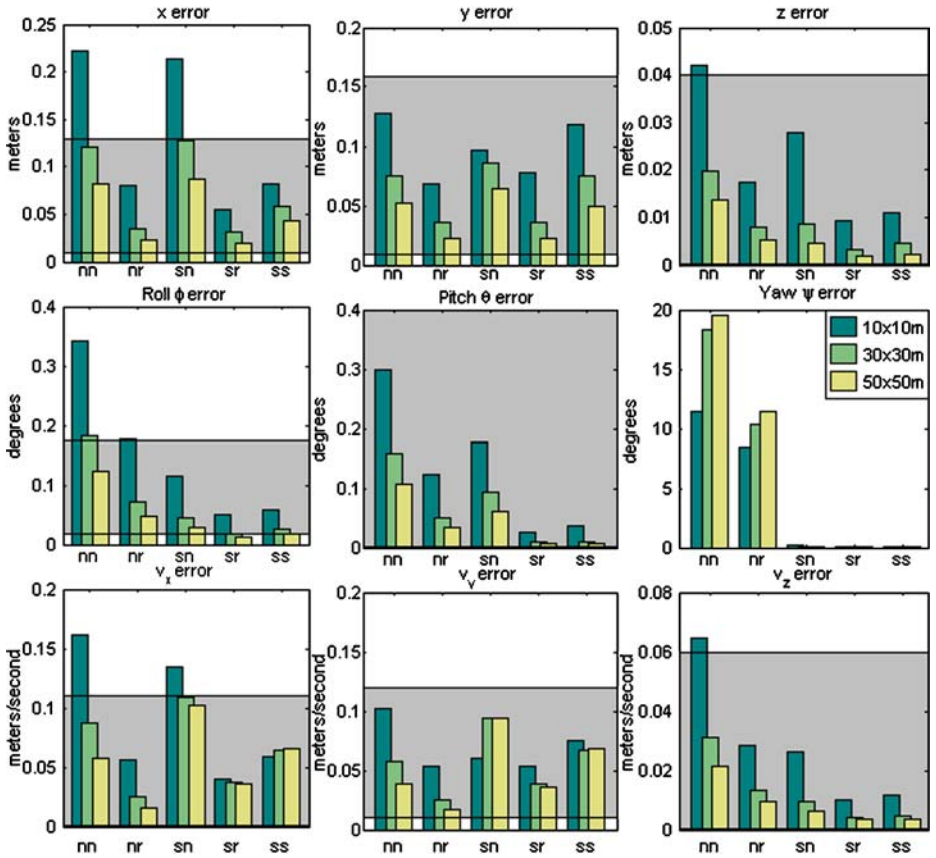

Fig. 8 Rectangular trajectory average error E_x on different smoothness configurations (*nn*, *nr*, *sn*, *sr*, *ss*) and lengths (10×10, 30×30, 50×50 m) for position (x, y, x), attitude (φ, θ, ψ) and speed (v_x, v_y, v_z)

The control effort plots presented on Fig. 9 show again larger variations and impulses on the non smooth trajectories and speeds (*nn*, *nr*, *sn*). Figures 10, 11 and 12 compare the rendered and simulation trajectories for circles of radius 10, 30 and 50 m. Figure 13 shows the corresponding schedules for the smooth configurations (*sr*, *ss*). It can be observed how the control system tries to preserve a smooth trajectory on the polygon edges for the non smoothed trajectory (*nn*), but still fails to meet many of the trajectory waypoints compared with the smooth case on 30 and 50 m. For the 10 m circle it is harder, even on the smooth trajectories, to reach each waypoint. These effects are reflected on the error measure of Fig. 14.

An important effect observed in Fig. 14 is the importance of not just positional smoothing but also its combination with progressive speed changes. This is observed in the case where a smooth trajectory (*sn*) is not enough to alleviate the 0.5 m/s speed changes (Refer also to Fig. 13). Control system deviation and maximum impulse show reduced control effort on all the smooth trajectories (*sn*, *sr*, *ss* of Fig. 15) with the exception of *sn* in δ_{lon}, which is related with the pitch setting and forward speed.

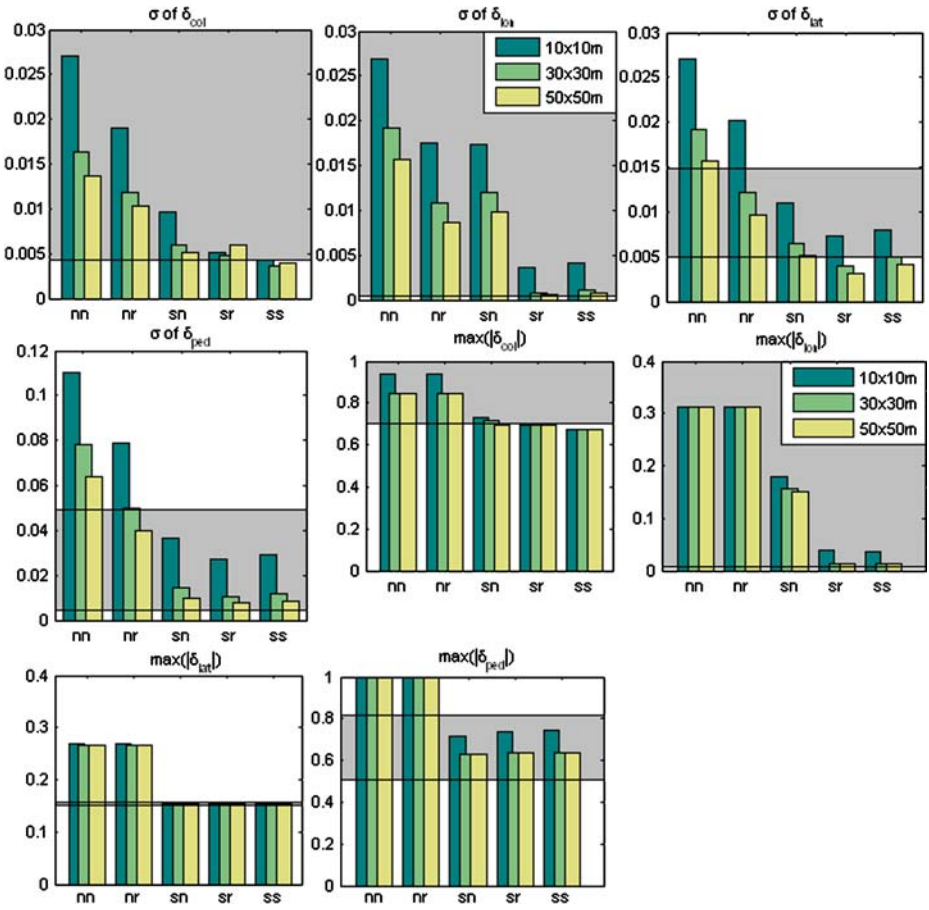

Fig. 9 Rectangular trajectory control standard deviation (σ) and absolute maximum values for each vehicle input ($\delta_{col}, \delta_{lon}, \delta_{lat}, \delta_{ped}$)

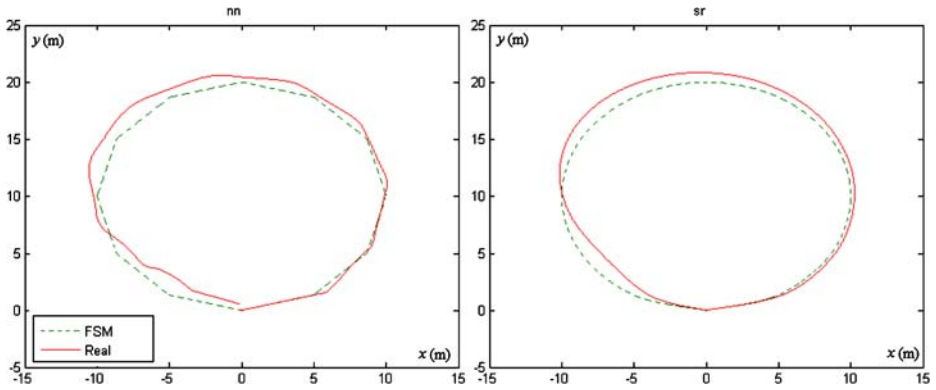

Fig. 10 Ten meters circular trajectory on the non smooth and smooth cases

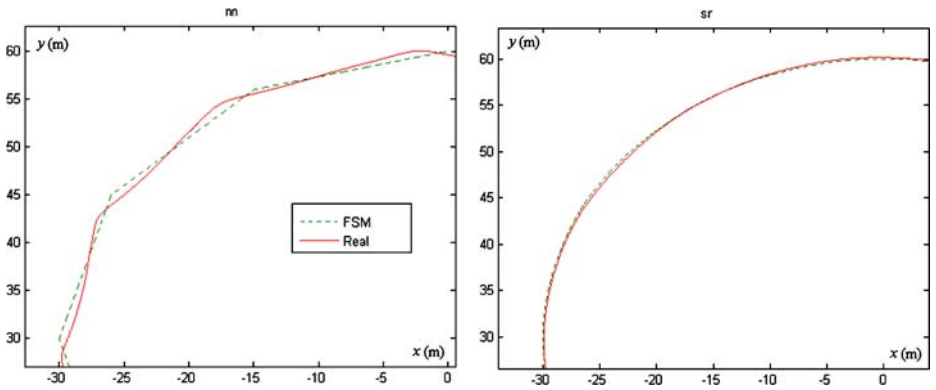

Fig. 11 Thirty meters circular trajectory section on the non smooth and smooth cases

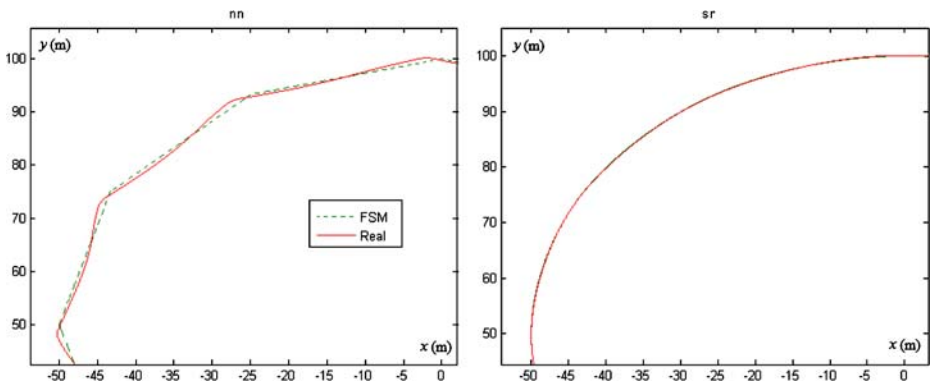

Fig. 12 Fifty meters circular trajectory on the non smooth and smooth cases

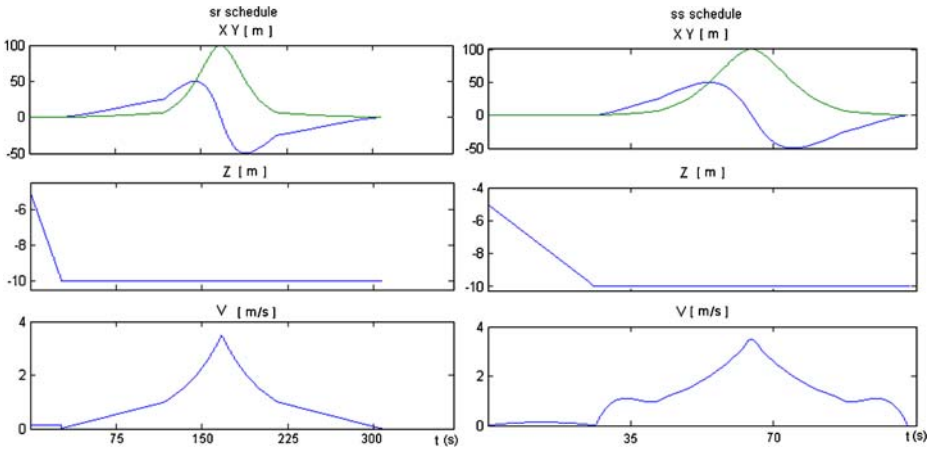

Fig. 13 Fifty meters circular trajectory schedule for the ramp-like (*sr*) and smooth speed (*ss*) cases

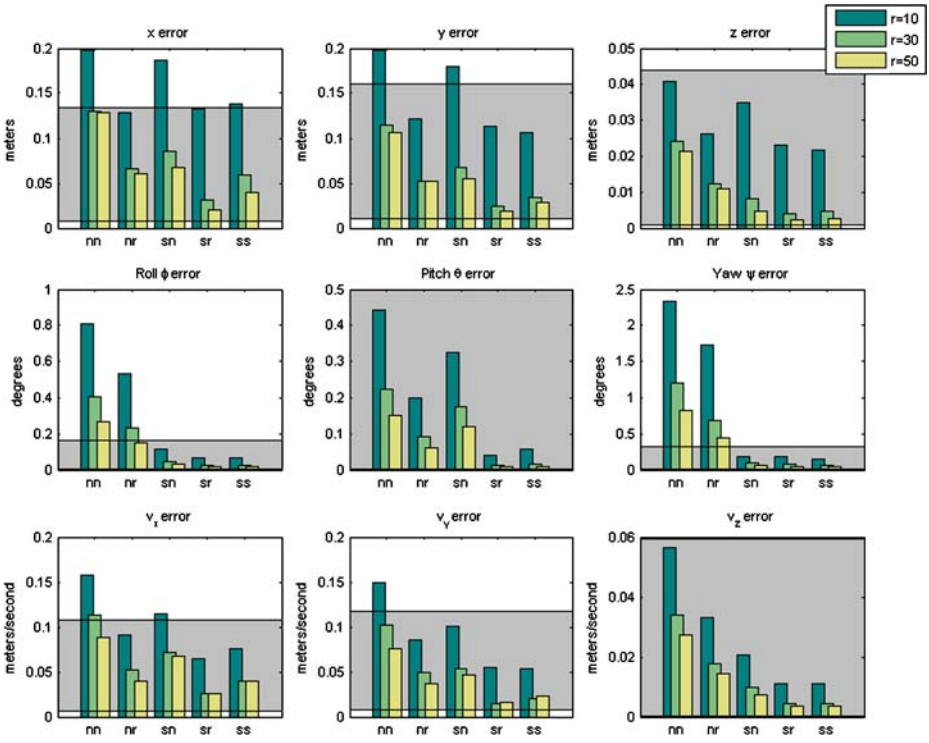

Fig. 14 Circular trajectory average error E_x on different smoothness configurations (*nn*, *nr*, *sn*, *sr*, *ss*) and radii (10, 30, 50 m) for position (x, y, x), attitude (φ, θ, ψ) and speed (v_x, v_y, v_z)

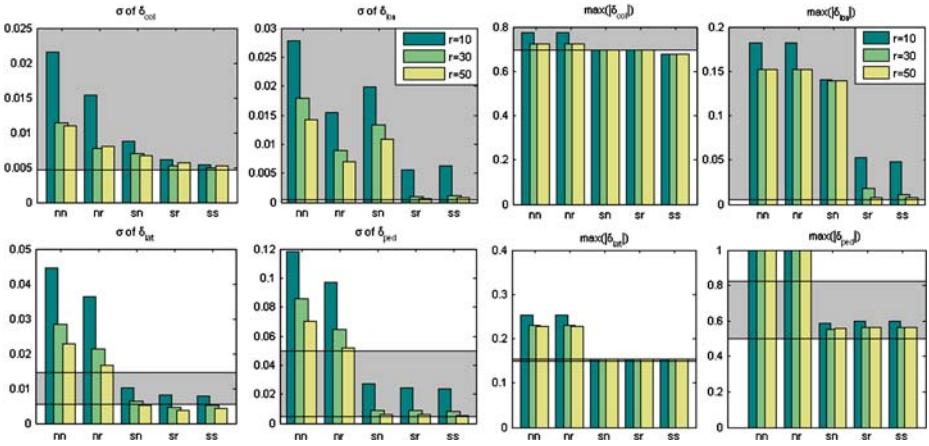

Fig. 15 Circular trajectory control standard deviation (σ) and absolute maximum values for each vehicle input ($\delta_{col}, \delta_{lon}, \delta_{lat}, \delta_{ped}$)

7.3 Complex Trajectory

A complex trajectory with several hover waypoints and large speed transitions (from 0 to 4 m/s and 4 to 0 m/s) was tested in simulation (Fig. 16). On this trajectory positional and speed error tests show better the influence of the ramp-like speed changes on decreasing error (Fig. 17). In this trajectory the *nr* case showed lower positional error. Note that this does not imply that the non smooth trajectories are better (compare with the *nn* case) but that the ramp velocity improves the behavior (order one speeds result in zero order accelerations). This still does not explain the outperformance of the positional *xy* smoothing trajectories and further research is required. Altitude and attitude errors and control effort (Fig. 18) are smaller for the smooth trajectories and speeds.

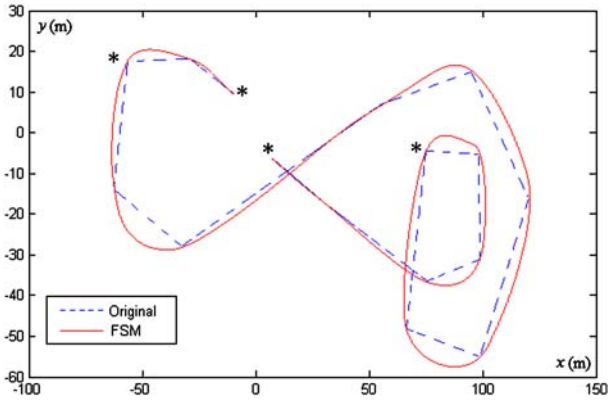

Fig. 16 A complex trajectory with hover points (*). Axis are in meters

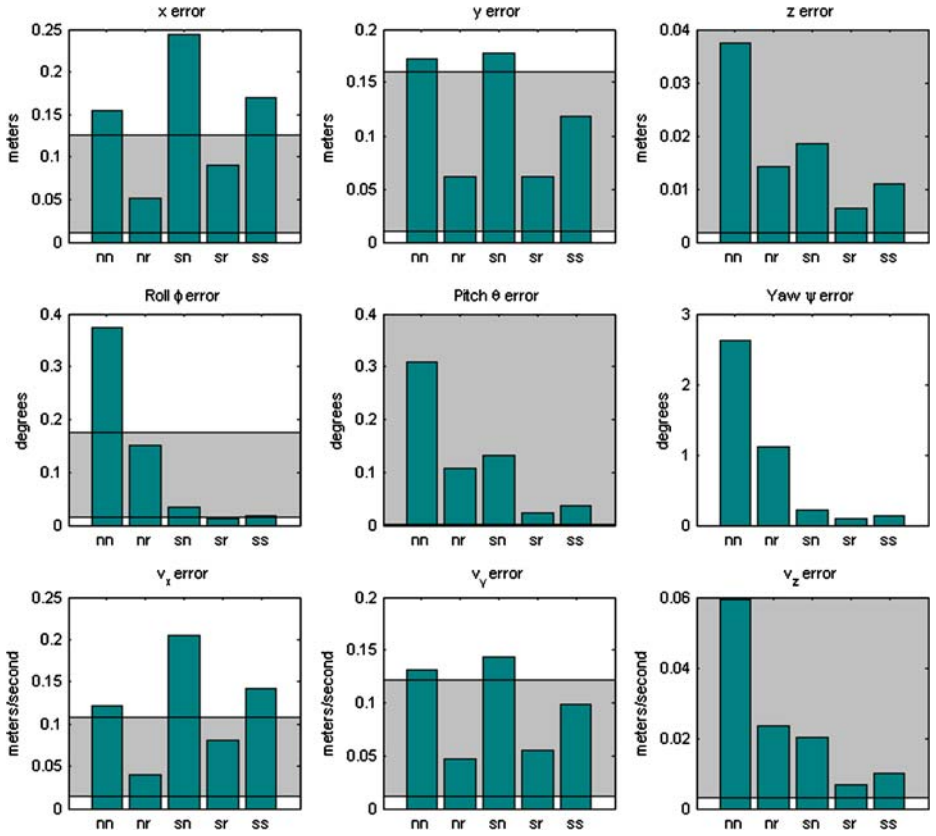

Fig. 17 A complex trajectory average error E_x on different smoothness configurations (*nn*, *nr*, *sn*, *sr*, *ss*) for position (x, y, x), attitude (φ, θ, ψ) and speed (v_x, v_y, v_z)

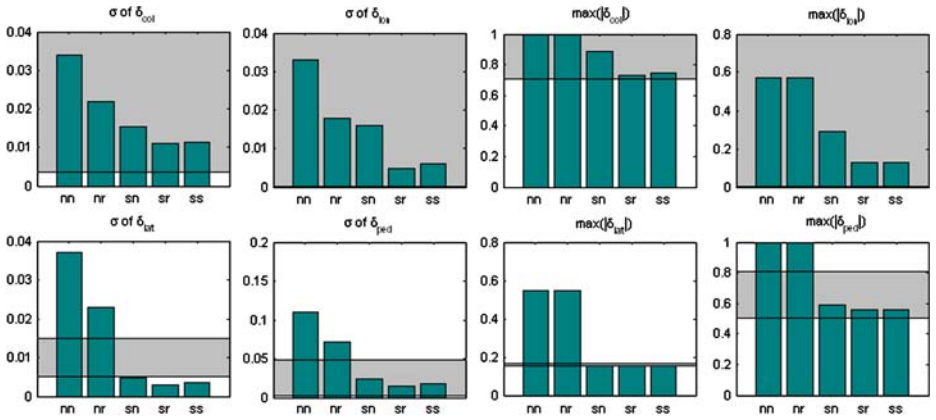

Fig. 18 Complex trajectory control standard deviation (σ) and absolute maximum for each vehicle input in various smoothness configurations

7.4 Computer Test

To verify feasibility of the proposed methods in real-time, the control system was translated to C code with an automated code generation tool [16] and run on a 300 MHz flight computer (PC-104) using a real time operating system [19]. Individual tests showed that the CPU usage of the FSM and control system never surpassed 1% usage of CPU at sample rates of $T_s = 0.02$ s. The rest of the system time was used by the Kalman filter (17%) and communication threads (50%), leaving more than 30% of the processor free.

8 Conclusions and Future Work

A method to smooth the planned trajectory and manage mode transitions for a small autonomous helicopter was presented. The smoothing methods were described and combined with a FSM able to render the trajectory in real time. Simulation results show that the smoothing proposed methods reduce transients, decreasing control error and effort.

Numerical methods intended to extract length and time properties of the trajectory were proposed. The Gauss–Legendre method was successfully applied to obtain length of two-dimensional Catmull–Rom curves. A numerical method able to combine a parametric smooth trajectory with a speed function was proposed and formally defined for the FSM.

Simulations showed that a smooth trajectory combined with progressive linear speed changes is able to reduce the effects of transients. Reduced error in first order speed schedules compared to smooth third order speeds could be attributed to resulting zero order accelerations. This influence needs to be clarified in the future.

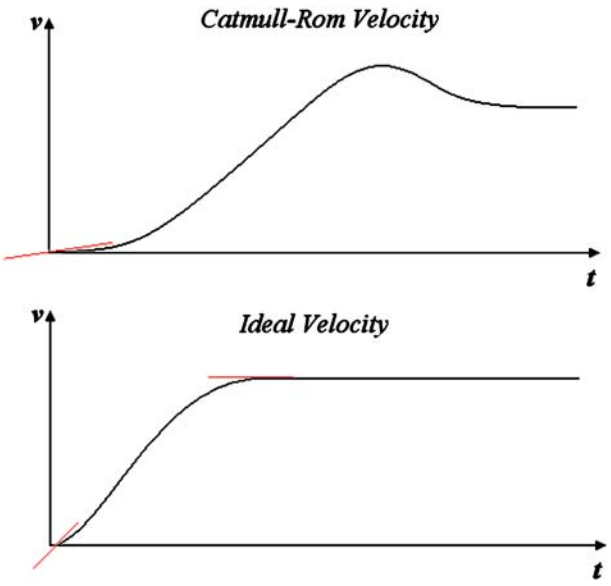

Fig. 19 Actual smooth speed schedule compared to the ideal speed schedule shape

To further reduce errors imposed at the joints of ramp speeds and reduce traversal time, the speed transition curve of Fig. 19 is proposed. Compared to Catmull–Rom smoothed speed, which has a slow start and overshot at the end, the ideal speed shape is almost lineal, grows faster (reducing traversal time) and has no overshoot (Catmull–Rom) or sharp bends (ramp). Such trajectory might be described by Hermite curves with chosen tangents [2] and will be tested on the future.

Simulation tests also showed that smooth trajectories tend to improve the vehicle attitude and control, more than smooth speeds. At the same time, progressive speeds decrease positional error.

Better forms to produce the first segment of the Catmull–Rom algorithm are suggested. The current algorithm repeats the first element on the vector $[\mathbf{x}_1\ \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3]$ creating an approximated tangent in the first point. Extrapolation methods could be used to generate an auxiliar data point $\mathbf{x}_{-1}$. The implementation of an algorithm to choose the tangents of the ideal speed shape (Fig. 19) could further reduce the errors obtained in the smooth trajectory and ramp-like speed configuration.

Acknowledgements The authors thank EAFIT University for support of the Colibrí project 45-000034 and Colciencias for the Young Researcher Award p-2006-0716.

References

1. Agudelo-Toro, A.: Trajectory smoothing and transition management for a small autonomous helicopter. Master's thesis. EAFIT University, Medellin, Colombia (2008)
2. Bartels, R.H., Beatty, J.C., Barsky, B.A.: An Introduction to Splines for Use in Computer Graphics & Geometric Modeling. Morgan Kaufmann, Los Altos (1987)
3. Branicky, M., Borkar, V., Mitter, S.: A unified framework for hybrid control: model and optimal control theory. IEEE Trans. Automat. Contr. **32**, 31–45 (1998). doi:10.1109/9.654885
4. Brockett, R.W.: Hybrid models for motion control systems. In: Trentleman, H.L., Willems, J.C. (eds.) Essays on Control: Perspectives in the Theory and its Applications, pp. 29–54. Birkhauser, Boston (1993)
5. Catmull, E.E., Rom, R.J.: A class of local interpolating splines. In: Barnhill, R.E., Riesenfeld, R.F. (eds.) Computer Aided Geometric Design, pp. 317–326. Academic, Orlando (1974)
6. Coleman, D., Creel, D., Drinka, D.: Implementation of an autonomous aerial reconnaissance system entry for the international aerial robotics competition. Technical Report. University of Texas, Austin (2002)
7. Doherty, P., Haslum, P., Heintz, F., Merz, T., Nyblom, P., Persson, T., Wingman, B.: A distributed architecture for autonomous unmanned aerial vehicle experimentation. In: 7th International Symposium on Distributed Autonomous Robotic Systems. Toulouse, France (2004)
8. Gavrilets, V.: Autonomous aerobatic maneuvering of miniature helicopters. Ph.D. thesis. Massachusetts Institute of Technology (2003)
9. Geyer, M.S., Johnson, E.N.: 3D obstacle avoidance in adversarial environments for unmanned aerial vehicles. In: Proceedings of the AIAA Guidance, Navigation, and Control Conference. AIAA, Keystone, Colorado (2006)
10. Guenter, B., Parent, R.: Motion control: computing the arc length of parametric curves. IEEE Comput. Graph. Appl. **10**(3), 72–78 (1990). doi:10.1109/38.55155
11. Guler, M., Clements, S., Wills, L., Heck, B., Vachtsevanos, G.G.: Generic transition management for reconfigurable hybrid control systems. In: Proceedings of the 20th American Control Conference. Arlington, VA (2001)
12. Harbick, K., Montgomery, J.F., Sukhatme, G.S.: Planar spline trajectory following for an autonomous helicopter. JACIII **8**(3), 237–242 (2004)
13. Koo, T.J., Hoffmann, F., Shim, H., Sinopoli, B., Sastry, S.: Hybrid control of model helicopter. In: IFAC Workshop on Motion Control, pp. 285–290. Grenoble, France (1999)

14. Koo, T.J., Sinopoli, B., Sangiovanni-Vicentelli, A., Sastry, S.: A formal approach to reactive system design: unmanned aerial vehicle flight management system design example. In: Proceedings of the IEEE International Symposium on Computer-Aided Control System Design, Kohala Coast, Hawaii (1999)
15. Mathews, J.H., Fink, K.D.: Numerical Methods using Matlab, 4th edn. Prentice Hall, New Jersey (2004)
16. Mathworks: Real-Time Workshop User's Guide (2006)
17. Menon, P.K.A., Kim, E.: Optimal helicopter trajectory planning for terrain following flight. Technical Report. NASA, Contractor Report (1990)
18. Mettler, B.: Identification Modeling and Characteristics of Miniature Rotorcraft. Kluwer, Boston (2003)
19. QNX Software Systems: QNX Momentics Development Suite (2008)
20. Richards, A.G.: Trajectory optimization using mixed-integer linear programming. Master's thesis. MIT (2002)
21. Rogers, R.M.: Applied Mathematics in Integrated Navigation Systems. AIAA Education Series, Reston, VA (2000)
22. Sanders, C.P., DeBitetto, P.A., Feron, E., Vuong, H.F., Leveson, N.: Hierarchical control of small autonomous helicopters. In: Proceedings of the 37th IEEE Conference on Decision and Control, vol. 4, pp. 3629–3634. Tampa, FL (1998)
23. Saripalli, S., Sukhatme, G.S., Montgomery, J.F.: An experimental study of the autonomous helicopter landing problem. In: Proceedings, International Symposium on Experimental Robotics. Sant'Angelo d'Ischia, Italy (2002)
24. Schrage, D., Vachtsevanos, G.: Software-enabled control for intelligent UAV's. In: Proceedings of 1999 Int. Conference on Control Applications, Hawaii (1999)
25. Velez, C.M., Agudelo, A.: Multirate control of an unmanned aerial vehicle. WSEAS Trans. Circuits Syst. **4**, 1628–1634 (2005)
26. Velez, C.M., Agudelo. A.: Rapid software prototyping for real-time simulation and control of a mini-helicopter robot. In: 10th WSEAS International Conference on Systems, pp. 289–295. Athens, Greece (2006)
27. Velez, C.M., Agudelo, A., Alvarez, J.: Modeling, simulation and rapid prototyping of an unmanned mini- helicopter. In: AIAA Modeling and Simulation Technologies Conference and Exhibit Conference Proceedings, Keystone, Colorado, USA (2006)
28. Zeigler, B.P., Praehofer, K., Kim, T.G.: Theory of Modeling and Simulation, 2nd edn. Academic, London (2000)