Waypoint based online trajectory generation and improved following control for the ACT/FHS

Philippe Petit*, Johannes Wartmann†, Benjamin Fragnière‡, and Steffen Greiser§,
DLR (German Aerospace Center), Institute of Flight Systems, 38108 Braunschweig, Germany

New research activities on DLR's EC135 ACT/FHS research helicopter has led to new challenges for the automatic flight control software of this testbed. Noise abatement flights, innovative automatic approach trajectories and other topics all require the helicopter to follow a set of waypoints as precisely as possible.
This paper presents a new algorithm which is capable of generating curvature-continuous trajectories. A modification of the natural cubic spline algorithm is presented, which allows to incrementally fit cubic splines with defined boundary conditions through subsets of the waypoint list. By joining several of these splines with matching boundary condition, a continuous trajectory is obtained. This approach allows to distribute the computational load of the cubic spline calculation over several execution cycles of the real-time system. Based on these results, a state feedback and a carrot chasing controller are developed and compared in terms of their ability to reduce and stabilize the distance between the trajectory and the helicopter. The approach is verified in DLR's Air Vehicle Simulator (AVES).

I. Nomenclature

$\vec{A}_k, \vec{B}_k, \vec{C}_k, \vec{D}_k$	Parameter vectors for k-th cubic spline	-
C_{init}, C_{end}	Initial & end condition for 2nd order derivative	-
$C(\lambda)$	Parametrized circle equation with angle λ	-
E_k^x, E_k^y	First order derivative of k-th cubic spline. x- and y-axis	-
K_p, K_d, K_i	PID gains	-
$L_{b_{1s}}, L_{\dot{b}_{1s}}$	Lateral flapping dynamics constants	$\frac{1}{s^2}$, $\frac{1}{s}$
L_p	Derivative of roll-moment w.r.t. angular velocity p	$\frac{rad}{s^2}$
$\vec{P}_k$	Waypoint vector of k-th waypoint	-
R	Circle radius	m
V_I	Groundspeed	$\frac{m}{s}$
$X_k(s), Y_k(s)$	Trajectory coordinates of k-th cubic spline	-
$\vec{\gamma}_k(s)$	Cartesian vector of k-th cubic spline	-
$a_k^x, b_k^x, c_k^x, d_k^x$	Parameters of k-th cubic spline in x-coordinates	-
$a_k^y, b_k^y, c_k^y, d_k^y$	Parameters of k-th cubic spline in y-coordinates	-
$b_{1_s}, \dot{b}_{1_s}$	Lateral flapping mode states	rad, $\frac{rad}{s}$
g	Gravitational acceleration of Earth	$\frac{m}{s^2}$
k_ϕ, k_{dy}	Controller parameters of MBC roll dynamics	$\frac{1}{s^2}$, -
s	Parameter of cubic spline equation	-
t_{pred}	Prediction time of VTP ahead of helicopter	s
x_k, y_k	x- and y-component of k-th waypoint	m
χ	Course angle	rad
κ	Curvature of trajectory	-
ϕ	Roll angle	rad
λ	Parameter of circle	-
Γ	Trajectory in Cartesian coordinates	-
Γ_n	Trajectory segment n	-
$\square_{FS}$	Frenet-Serret frame	-
$\square_{VTP}$	Virtual target point	-

*Research Associate, Rotorcraft Department, Philippe.Petit@DLR.de.
†Research Associate, Rotorcraft Department, Johannes.Wartmann@DLR.de.
‡Research Associate, Rotorcraft Department, Benjamin.Fragnier@DLR.de
§Research Associate, Rotorcraft Department, Steffen.Greiser@DLR.de.

Fig. 1 DLR's ACT/FHS Helicopter.

II. Introduction

DLR's EC135 **A**ctive **C**ontrol **T**echnology **/** **F**lying **H**elicopter **S**imulator (ACT/FHS, see Fig. 1) is used for a variety of research topics. These include validation of novel control and identification techniques, pilot assistance systems, noise abatement and innovative autonomous approach and landing techniques amongst others.

A lot of this research requires the precise following of trajectories which are typically defined and supplied as a list of waypoints. In this context, waypoints are defined as physical locations in space which the vehicle shall pass through. Trajectories on the other hand are mathematical curves defining a continuous flight path which the vehicle shall follow. One example for such a research topic is the autonomous approach and landing research in **D**egraded **V**isual **E**nvironment (**DVE**) scenarios. Hereby, the helicopter shall follow a trajectory communicated via a list of waypoints, which have to be followed exactly to avoid collision with ground obstacles. Additionally, due to dynamic obstacle detection and recalculation of the waypoint list, the ACT/FHS has to be able to cope with dynamic updates of the supplied waypoints. Many other research applications also supply trajectories defined by waypoint lists for their respective target. This illustrates the requirement for the **A**utomatic **F**light **C**ontrol **S**ystem (AFCS) of the ACT/FHS:

1) The ability of the AFCS to quickly plan and replan a trajectory through a given set of waypoints
2) The ability of the AFCS to follow this trajectory with little error

Trajectory generation and path following have been extensively studied for various fixed and rotary wing aircraft both manned [1] and unmanned [2], ground-based vehicles [3], and robotic arm manipulators [4]. Current autopilots often use simple line-and-circle methods based on the Dubins Car Model [5] or extensions of this model [6]. While these approaches are simple, robust, and computationally inexpensive [7] the transition between a straight line and a segment of a circle, introduces a discontinuity in the curvature of the trajectory. This C^2 discontinuity then leads to a step in the demanded roll angle. Different techniques exist to address this drawback, for example by introducing clothoid segments in the transition between straight line and arc segments [7].

Another approach to generate trajectories between waypoints are spline segments such as Bezier curves [8]. However, the curvature at the interface of two Bezier segments is again discontinuous. This problem can be alleviated by employing a special case of Bezier curve with zero-curvature at the initial and end point which is then inserted between two adjacent straight-line segments [4] or by using higher order Bezier curves and an optimization approach to generate curvature-smooth trajectories [3, 9].

A different method to generate curvature-smooth trajectories from waypoint lists is to use piece-wise defined cubic splines similar to Bezier curves, and matching first and second order derivatives by solving a linear equation system [10]. While offering the advantage of a trajectory which is C^0, C^1, and C^2 continuous, the problem scales with the number of waypoints. Therefore large number of waypoints imply a high computational burden.

Once the trajectory is generated, the vehicle has to be controlled in such a way that it stays on the calculated trajectory. Typically, the problem is broken down in an inner loop controlling the body rates and angles of the aircraft, and an outer loop, generating commands for the inner loop. The outer loop generates commands based on path deviations such that the vehicle stays on the desired trajectory. In the following, we will assume that an adequate inner loop has been designed, which simplifies the problem of designing a suitable trajectory controller.

Trajectory control design has been the subject of extensive studies. Examples are PID control [11], LQR control [12], nonlinear Model Predictive Control with and without learning components [13, 14], nonlinear guidance laws adapted to

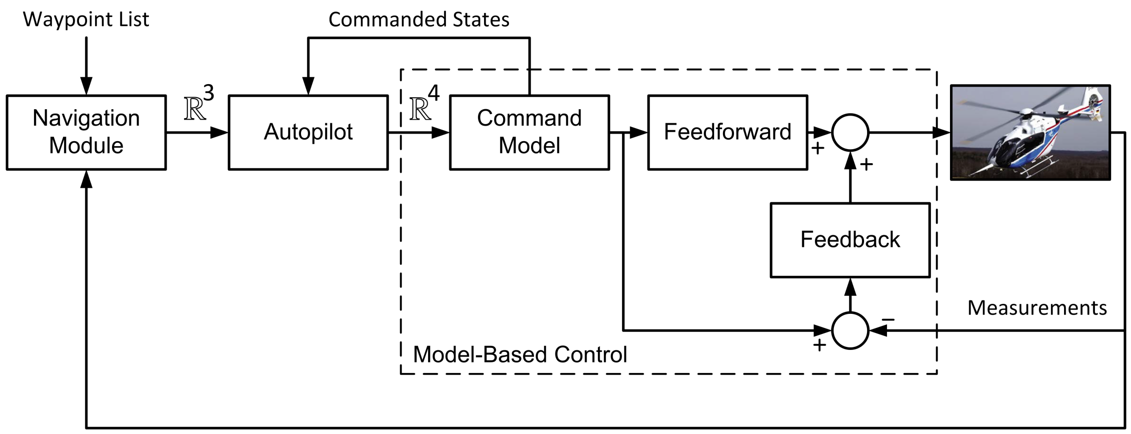


Fig. 2 Schematic of the control system structure of the ACT/FHS [19].

this problem [15], vector-field based control approaches, [16], and algorithms known as "carrot chasing" or "rabbit chasing" [17] amongst others. A good overview over a selection of algorithms is provided by [17] which explores the different properties of these algorithms.
All these algorithms were developed for unmanned aircrafts either in simulation or in real flight-tests. Additionally, most of these algorithms are designed for a constant velocity of the vehicle and have to be retuned for different velocities [6, 14, 15]. For full size helicopters, these algorithms have to be adjusted as seen in the results of the flight-tests with a full-sized Blackhawk helicopter using a modified "carrot chasing" algorithm which was implemented to steer the helicopter along a given trajectory [18].
This study aims to develop an incremental real-time interpolation method based on cubic splines, along with a suitable outer loop guidance and control algorithm. Both together can be used to generate and fly along trajectories with smooth curvature suited for manned helicopter applications in a real-time environment. The 2D case in the x-y plane is considered in the following.

III. The Control System of DLR's ACT/FHS

DLR's EC135 ACT/FHS is a light, twin-engine helicopter with bearlingless main rotor and fenestron (Fig. 1). It features a full-authority fly-by-wire/fly-by-light primary control system, which allows an experimental control system to manipulate the controls of the helicopter.
The core control strategy of the ACT/FHS uses a **M**odel-**B**ased **C**ontrol (MBC) approach in which the control system imposes a specified model behavior to the closed-loop system [19]. This can be a model which offers easy-to-control and favorable flight characteristics or emulates a different helicopter. A schematic of the MBC structure is displayed in Fig. 2. Hereby, the Command Model generates reference states which are used to control the ACT/FHS via a Feedforward and a Feedback Controller. An Autopilot is acting on the states of the Command Model to enable low-level stabilization and control. The Navigational Module provides high-level commands to the Autopilot based on the spatial position of the helicopter.
A Model-Based Control approach can be of great use to reduce the workload of the pilot by specifying an easy-to-fly control model or to test a new helicopter designs before production, which also motivates the name "Flying Helicopter Simulator". For normal flight, the control system is configured to emulate an easy-to-fly, decoupled helicopter with ADS-33 level-1 handling qualities. This model artificially couples the body rates p, q, and r in order to achieve coordinated turns in fast forward flight. During hover, this coordinated turn coupling is disabled.
A consequence of this coupling is that during coordinated turns, the roll angle ϕ is directly related to the course angle change $\dot{\chi}$

$$\dot{\chi} = \frac{g}{V_I} \cdot \tan(\phi). \tag{1}$$

It is also worth noting that the turn rate depends linearly on the velocity magnitude in ground frame V_I and the curvature of a given trajectory κ

$$\dot{\chi} = V_I \cdot \kappa, \tag{2}$$

which will be useful for the design of the Navigational Module in Chapter V. The parameter g denotes earth's gravitational acceleration.

IV. Cubic Spline interpolation of a waypoint list

A list of $N + 1$ waypoints $\vec{P}_0, \ldots, \vec{P}_N$ in Cartesian space (x,y) serves as input for the following algorithms and is considered given. Each waypoint $\vec{P}_k$ is a two-dimensional vector consisting of the x- and y- component x_k and y_k. A trajectory is denoted by Γ and consist of piece-wise defined cubic splines $\vec{\gamma}_k$ connecting the waypoint $\vec{P}_k$ with the next waypoint $\vec{P}_{k+1}$.

A. Cubic Spline Algorithm

A piece-wise defined trajectory Γ can be constructed by using N parametric vectors consisting of a two-dimensional vector polynomial function

$$\vec{\gamma}_k(s) := \begin{pmatrix} X_k(s) \\ Y_k(s) \end{pmatrix} = \vec{A}_k + \vec{B}_k \cdot s + \vec{C}_k \cdot s^2 + \vec{D}_k \cdot s^3, \qquad s \in [0, 1]. \tag{3}$$

The parameters $\vec{\gamma}_k$, $\vec{A}_k$, $\vec{B}_k$, $\vec{C}_k$, and $\vec{D}_k$ being two-dimensional vectors. The subscript k denotes the k-th polynomial of the trajectory Γ. Parameter vectors are defined as

$$\vec{A}_k = \begin{pmatrix} a_k^x \\ a_k^y \end{pmatrix} \quad , \quad \vec{B}_k = \begin{pmatrix} b_k^x \\ b_k^y \end{pmatrix} \quad , \quad \vec{C}_k = \begin{pmatrix} c_k^x \\ c_k^y \end{pmatrix} \quad , \quad \vec{D}_k = \begin{pmatrix} d_k^x \\ d_k^y \end{pmatrix}. \tag{4}$$

The symbol a_k^x indicates the x-component of the k-th parameter of $\vec{A}_k$. All other parameter vectors are constructed similarly. Each parameter is only valid on the interval from waypoint k to waypoint $k + 1$ with $k < N$. The parameter s is defined on the interval ranging from 0 to 1.
Derivatives of this function can be determined:

$$\vec{\gamma}_k'(s) := \begin{pmatrix} X_k'(s) \\ Y_k'(s) \end{pmatrix} = \vec{B}_k + 2 \cdot \vec{C}_k \cdot s + 3 \cdot \vec{D}_k \cdot s^2 \tag{5}$$

$$\vec{\gamma}_k''(s) := \begin{pmatrix} X_k''(s) \\ Y_k''(s) \end{pmatrix} = 2 \cdot \vec{C}_k + 6 \cdot \vec{D}_k \cdot s \tag{6}$$

The flight-path azimuth angle of a path segment is defined by:

$$\tan(\chi) = \frac{Y_k'(s)}{X_k'(s)} \tag{7}$$

The curvature of this path is defined by:

$$\kappa(s) = \frac{X_k'(s) \cdot Y_k''(s) - Y_k'(s) \cdot X_k''(s)}{(X_k'(s)^2 + Y_k'(s)^2)^{3/2}} \tag{8}$$

B. Computing Coefficients

The aforementioned parameters $\vec{A}_k$, $\vec{B}_k$, $\vec{C}_k$, and $\vec{D}_k$ can be determined with the natural spline interpolation method as described in [20, p. 6]. This algorithm fits a piece-wise defined cubic spline through a list of sampled points with C^0, C^1, and C^2 continuity and a zero second-order derivative at the start and end points. This process has to be performed twice, once for each dimension. The k-th spline coefficients a_k^x, b_k^x, c_k^x, and d_k^x can be calculated from the x-value of the k-th waypoint x_k via

$$a_k^x = x_k \tag{9}$$

$$b_k^x = D_k^x \tag{10}$$

$$c_k^x = 3(x_{k+1} - x_k) - 2 \cdot D_k^x - D_{k+1}^x \tag{11}$$

$$d_k^x = 2(x_k - x_{k+1}) + D_k^x + D_{k+1}^x. \tag{12}$$

The parameters D_k^x are the first order derivatives of the splines of the x-axis at the start of each cubic spline segment γ_k. For the y-axis, this parameter is analogously labeled as D_k^y. By requiring a continuous second order derivative, and with the help of Eq. (9) - Eq. (12), the following linear equation system can be derived [20, p. 9] (shown here for the x-axis)

$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & 0 & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & & \ddots & & \\ & 0 & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} D_0^x \\ D_1^x \\ D_2^x \\ D_3^x \\ \vdots \\ D_{N-1}^x \\ D_N^x \end{bmatrix} = \begin{bmatrix} 3(x_1 - x_0) \\ 3(x_2 - x_0) \\ 3(x_3 - x_1) \\ 3(x_4 - x_2) \\ \vdots \\ 3(x_N - x_{N-2}) \\ 3(x_N - x_{N-1}) \end{bmatrix}. \tag{13}$$

The solution for the problem is therefore obtained by solving Eq. (13) and substituting the result into Eq. (9) - Eq. (11). This yields the parameters for the x-axis parameters a_k^x, b_k^x, c_k^x, and d_k^x. The parameters a_k^y, b_k^y, c_k^y, and d_k^y can be calculated analogously, which is not shown here. Both sets of parameters define the trajectory Γ.
As mentioned before this algorithm generates a natural cubic spline with C^0, C^1, and C^2 continuity as well as a second order derivative equal to zero at the boundaries of the path. An example is shown in Fig. 4a. A natural spline was fitted through waypoints lying on a semicircle with a radius of 550 meters and a constant curvature of $\kappa = \frac{1}{550} = 1.82 \cdot 10^{-3}$. The fitted spline begins with a curvature of 0, then shows an increased curvature of up to $2.3 \cdot 10^{-3}$ before oscillating around the mean value of the circle's constant curvature. Important to notice is the aforementioned C^0, C^1, and C^2 continuity.
The fact that a natural spline has a zero initial curvature as well as its C^0, C^1, and C^2 continuity properties are desirable for trajectory generation based on waypoint lists. However, the disadvantage of this approach is that the entire path has to be calculated in one step. For many path planning algorithms, this is not an issue as the linear equation Eq. (13) is not particular expensive to solve especially when path planning is performed offline. For planning systems with receding horizon, dynamic re-planning of waypoints [21] or very limited computational resources this is not true. In the particular case of the ACT/FHS, waypoints may be recomputed as soon as new obstacles are detected or other limits are introduced. Also large sets of waypoints (>200) may violate real-time requirements of the concerning module. In order to address this shortcoming, the presented cubic spline algorithm is extended.

C. Real-time Adaptation

The idea behind the following algorithm is to stitch several cubic spline trajectories together with defined curvature κ at the interfaces. This approach allows the computation of each segment separately, therefore enabling a distribution of the computational load over several execution cycles. In order to emphasize the fact that the trajectory now consists of several sub-trajectories, the indices i is attached to Γ, such that several trajectory segments Γ_i can be distinguished. In order to match two trajectory segments Γ_i and Γ_{i+1}, at their boundaries, it is intuitively clear that the curvature κ and the azimuth angle χ of the two trajectories have to be the same at the interface waypoint. Therefore by requiring the curvature and azimuth angle at the beginning of Γ_{i+1} to be the same as the curvature and azimuth angle at the end of Γ_i, both trajectories will blend seamlessly.
Such a constraint assumption is valid, as these boundary conditions give four new constraints for each trajectory segment Γ_i, two at each end, which replace the four zero-second-order-derivative constraints defined in IV.B. Therefore by inserting (5) - (6) into the curvature equation (8) and the azimuth angle (7), one obtains the curvature κ_k and azimuth angle χ_k for for the segment k:

$$\kappa_k(s) = \frac{(b_k^x + 2c_k^x s + 3d_k^x s^2)(2c_k^y + 6d_k^y s) - (b_k^y + 2c_k^y s + 3d_k^y s^2)(2c_k^x + 6d_k^x s)}{((b_k^x + 2c_k^x s + 3d_k^x s^2)^2 + (b_k^y + 2c_k^y s + 3d_k^y s^2))^{3/2}} \tag{14}$$

$$\chi_k(s) = \arctan\left(\frac{b_k^y + 2c_k^y s + 3d_k^y s^2}{b_k^x + 2c_k^x s + 3d_k^x s^2}\right) \tag{15}$$

Evaluating equations (14) and (15) at the very beginning of the trajectory, therefore at $k = 0$ and $s = 0$ and then inserting it into each other as well as inserting the coefficients defined in (9) - (12), yield the greatly simplified constraint for the curvature in terms of the required azimuth angle at the interface $\chi_0(0)$, and the derivatives of the x- and y- cubic splines E_k^x and E_k^y

$$\kappa_0(0) = \frac{3((y_1 - y_0) - \tan(\chi_0(0))(x_1 - x_0)) - E_1^y + E_1^x \tan(\chi_0(0))}{\frac{1}{2}(1 + \tan^2(\chi(0)_0)^{3/2} E_0^x}. \tag{16}$$

As can be seen, Equation (16) is non-affine and non-linear. Normally, the next step would be to incorporate Equation (16) into equation system (13). But this would transform the equation system to a non-linear equation system which also couples the x- and y-axis. This combined system is not suitable for the sought online solution to the problem. Therefore an approximation to this problem will be introduced in the following.

As an alternative, the second-order derivative can be approximated by the following method: As the curvature at the interface of each trajectory segment Γ_i can be chosen arbitrarily, it is determined to be the same as a circle fitted through waypoints $k-1$, k, and $k+1$. A sketch of this concept is shown in Fig. 3. By calculating an equivalent second order derivative for the x- and y-axis, this curvature can be approximated. With this idea in mind, some modifications to the original cubic spline algorithm are performed. At first, the boundary conditions of the natural cubic spline algorithm has to be altered to allow for a non-zero second order derivative at the interfaces:

$$X_0''(0) = C_{init} \tag{17}$$

$$X_{N-1}''(1) = C_{end} \tag{18}$$

Combining this boundary condition with Eq. (5), Eq. (11), and Eq. (12) yields the following modified version of the linear equation system given in Eq. (13)

$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & 0 \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & & \ddots & & \\ & 0 & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_{N-1} \\ D_N \end{bmatrix} = \begin{bmatrix} 3(x_1 - x_0) - \frac{1}{2} \cdot C_{init} \\ 3(x_2 - x_0) \\ 3(x_3 - x_1) \\ 3(x_4 - x_2) \\ \vdots \\ 3(x_N - x_{N-2}) \\ 3(x_N - x_{N-1}) + \frac{1}{2} \cdot C_{end} \end{bmatrix}. \tag{19}$$

Note that as before, this equation system has to be solved for x- and y-direction. The second order derivative which shall be achieved is that of a circle with radius R. A circle and its derivatives is parametrically described with its angle λ

$$C(\lambda) = R \cdot \begin{pmatrix} \cos(\lambda) \\ \sin(\lambda) \end{pmatrix}. \tag{20}$$

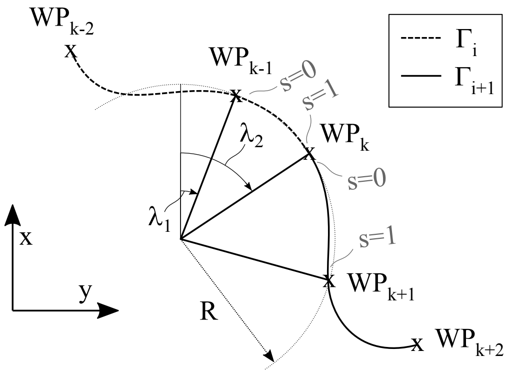
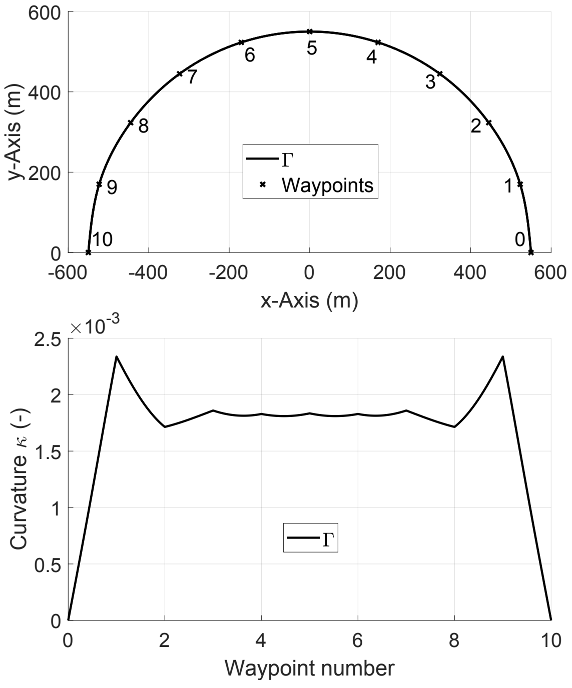


Fig. 3 Matching two splines at the interface waypoint *k*.


(a) Single cubic spline planned through 10 waypoints

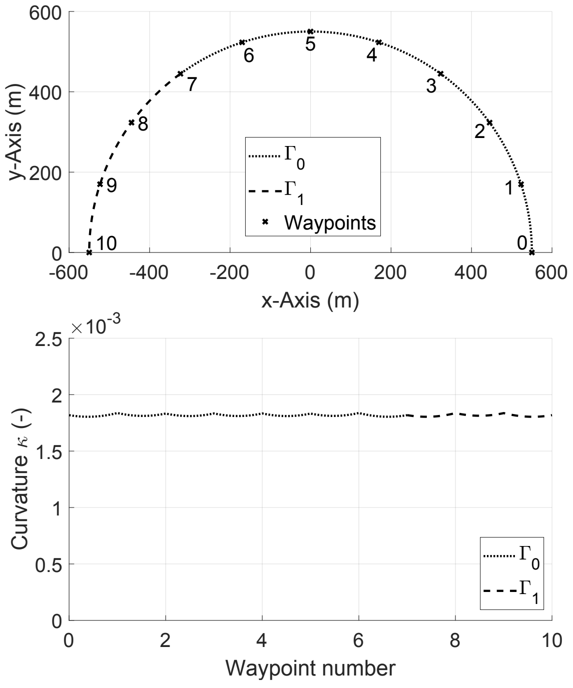

(b) Two cubic splines, stitched at waypoint number 7 with continuous non-zero curvature

Fig. 4 Cartesian space and curvature of sample trajectories planned trough 10 waypoints.

Because of the different variable ranges of the cubic spline and the circle parametrization of Eq. (20), a substitution of variables has to be performed before the second derivative of the circle can be compared to that of the cubic spline. The parameter s runs from 0 to 1, hence the following substitution is introduced:

$$\lambda = \lambda_1 + \Delta\lambda \cdot s, \tag{21}$$

with $\Delta\lambda = \lambda_2 - \lambda_1$. Inserting Eq. (21) into Eq. (20) and calculating the second derivative yields

$$C''(s) = -\Delta\lambda^2 \cdot R \cdot \begin{pmatrix} \cos(\lambda_1 + \Delta\lambda \cdot s) \\ \sin(\lambda_1 + \Delta\lambda \cdot s) \end{pmatrix}. \tag{22}$$

In order to determine the variables R, λ_1, and λ_2, a circle is fitted through the interface points of two cubic splines [22]. Considering the case sketched in Fig. 3, the end condition of path Γ_i and the initial condition of path Γ_{i+1} for the second order derivative can be constructed by evaluating Eq. (17) and Eq. (18)

$$\Gamma_i''(1) = \Gamma_{i+1}''(0) = C''(0) = -\Delta\lambda^2 \cdot R \cdot \begin{pmatrix} \cos(\lambda_1) \\ \sin(\lambda_1) \end{pmatrix}. \tag{23}$$

The proposed algorithm can be formulated as outlined in Algorithm 1:

Algorithm 1 The real-time adapted generation of cubic splines

```
1: Determine Waypoint indices $k_{init}$ and $k_{end}$ of begin and end of current trajectory segment $\Gamma_i$
2: Fit circle through Waypoints $k_{init-1}$, $k_{init}$, and $k_{init+1}$
3: Fit circle through Waypoints $k_{end-1}$, $k_{end}$, and $k_{end+1}$
4: Calculate second order derivatives with Eq. (22)
5: Calculate derivatives of cubic spline with Eq. (19)
6: Calculate cubic spline parameters with Eq. (9) to Eq. (12)
```

An application example of this algorithm is presented in Fig. 4b. In this example, waypoints were generated lying on a circle of radius $R = 550$ meters, which were then interpolated with two trajectory segments Γ_0 and Γ_1 based on the algorithm described above. The initial and end conditions for both trajectories were set to match the curvature of the circle $\kappa = \frac{1}{550m} \approx 1.8 \cdot 10^{-3}\frac{1}{m}$ while the interface between Γ_0 and Γ_1 was chosen arbitrarily to be the seventh waypoint. A constant non-zero curvature of a trajectory cannot be reproduced with the presented cubic spline approach because of the polynomial nature of the utilized cubic splines. This is also evident in Fig. 4b, in which the curvature cannot maintain a constant value, but still exhibits an "oscillation". The curvature of the circle is however approximately achieved at the interface points of the trajectory segments which illustrates the point of the proposed boundary conditions.

V. Trajectory Control

In order to evaluate the implemented trajectory generation algorithms, two simple path-following control algorithm were designed to enable simulated closed-loop flight.

Typically, the objective of a trajectory controller is to steer the helicopter in such a way that the lateral and vertical position errors are minimized [17]. For manned helicopter applications, the problem additionally implies a broad range of velocities ranging from hover at zero velocity to fast forward flight in the vicinity of 120 knots. In the following vertical and forward speed control are not considered. Even coupling can be neglected for the design of the lateral controller as the MBC provides decoupled response characteristic [19].

The Frenet-Serret frame [17, 23] is used in the following to describe different quantities relative to the path. In this coordinate frame, the x- axis x_{FS} is chosen tangentially to the track in direction of path-advancement while the y-axis y_{FS} is chosen normally to the first axis. The whole situation is sketched in Fig. 5. Now, the lateral distance to the path Δy_{FS} as well as the lateral velocity relativ to the path $\Delta \dot{y}_{FS}$ can be described. The subscript $\square_{FS}$ is used to describe quantities in the Frenet-Serret frame.

Please note the point P at the origin of the Frenet-Serret frame in Fig. 5. This point lies at the intersection between the track and a line connecting the track with the vehicle whereby this line forms a 90° angle with the track.

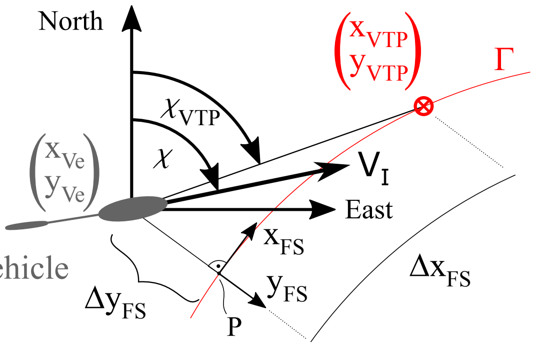

North
$\begin{pmatrix} x_{VTP} \\ y_{VTP} \end{pmatrix}$
Γ
χ_{VTP}
χ
$\begin{pmatrix} x_{Ve} \\ y_{Ve} \end{pmatrix}$
V_I
East
x_{FS}
Vehicle
Δx_{FS}
y_{FS}
Δy_{FS}
P

Fig. 5 The horizontal situation of the path with crosstrack distance Δy_{FS}, flight-path azimuth angle χ and flight-path azimuth angle to VTP χ_{VTP}. The Frenet-Serret frame in the point P is also sketched.

A. Control Model

The dynamics of the path deviation can be split into two parts: the rotational dynamics of the MBC, consisting of the azimuth angle and azimuth angle rate (χ and $\dot{\chi}$) of the MBC, as well as the cross-track dynamics compromised of the cross-track error and cross-track error velocity (Δy, $\Delta \dot{y}$). The control substitute model can therefore be obtained by coupling these two blocks and closing the loop with the controller. Fig. 6 shows an overview of the obtained control model.

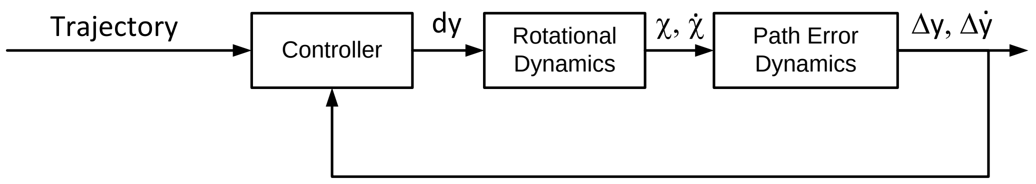


Fig. 6 The complete control model, consisting of the controller, the rotational heading dynamics and the cross-track dynamics.

1. Rotational Dynamics

The rotational dynamics of the azimuth angle χ and azimuth rate $\dot{\chi}$ are the output of a dynamic system with the lateral steering input dy as input compromised of the rotational dynamics of the MBC, the turn coordination coupling, and the autopilot transfer function, which can be summarized as a linear fourth order state space function with output non-linearity.

$$\frac{d}{dt}\begin{pmatrix}\phi\\ p\\ b_{1_s}\\ \dot{b}_{1_s}\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ 0 & -L_p & L_{b_{1s}} & 0\\ 0 & 0 & 0 & 1\\ k_\phi & -L_{\dot{b}_{1s}} & L_{b_{1s}} & L_{\dot{b}_{1s}}\end{pmatrix}\cdot\begin{pmatrix}\phi\\ p\\ b_{1_s}\\ \dot{b}_{1_s}\end{pmatrix} + \begin{pmatrix}0\\ 0\\ 0\\ L_{dy}k_{dy}\end{pmatrix}\cdot dy, \tag{24}$$

$$\dot{\chi} = \frac{g}{V_I}\cdot\tan(\phi). \tag{25}$$

The parameters $L_{b_{1s}}$, and $L_{\dot{b}_{1s}}$ are parameters of the lateral flapping modes b_{1_s} and $\dot{b}_{1_s}$ [24]. The desired roll dynamics of the MBC are tuned via the parameter L_p while control parameters k_ϕ and k_{dy} are chosen such that steady-state accuracy and desired system dynamics are achieved.
The dynamic system described in Eq. (24) is a stable, and fast system with a unity steady-state gain and poles as presented in table 1.

Table 1 Pole location of MBC roll dynamics

Pole index	Pole location	Description
$s_{1/2}$	$-3.7 \pm 2.8i$	MBC Roll Dynamics
$s_{3/4}$	$-30.3 \pm 2.8i$	Rotor States

2. Translational Dynamics

The Lateral path dynamics are described via the Frenet-Serret frame as sketched in Fig. 5. Therefore, the cross-track error Δy_{FS} is describing the distance between the point P and the position of the vehicle. By assuming a constant velocity, the dynamics of the lateral position error Δy_{FS}, and the lateral position error velocity $\Delta\dot{y}_{FS}$ can be formulated as follows:

$$\frac{d}{dt}\begin{pmatrix}\Delta\dot{y}_{FS}\\ \Delta y_{FS}\end{pmatrix} = \begin{pmatrix}V_I\cdot\cos(\chi-\chi_T)\cdot(\dot{\chi}-\dot{\chi}_T)\\ \Delta\dot{y}_{FS}\end{pmatrix} \tag{26}$$

whereby the azimuth angle of the track and its derivative are described by the symbol χ_T and $\dot{\chi}_T$.

B. Trajectory Controller

In order to control the cross-track error, two controllers will be presented in the following. The first is a rather conventional carrot chasing controller with feedforward, while the second one is based on cross-track error and cross-track error velocity feedback.

1. Carrot Chasing with Feedforward Command

The widely used carrot chasing algorithm [17] was selected as a simple and robust control algorithm [25]. The considered horizontal situation is sketched in Fig. 5, together with the **V**irtual **T**arget **P**oint (VTP) displayed on the track Γ. Additionally, the point perpendicular to the trajectory connected to the helicopter is marked with P. Current course angle χ and the course angle connecting the helicopter with the VTP χ_{VTP} are also sketched. The VTP is obtained by extrapolating the position of the point P over a user-defined time t_{pred} with the current velocity of the helicopter V_I projected onto the trajectory Γ. Therefore, the distance the VTP is placed ahead of the point P on the trajectory Γ is given by

$$\Delta x = V_I \cdot t_{pred}. \tag{27}$$

The distance Δx is therefore the arc length of the distance between the Point P and the VTP along the trajectory Γ.

After the position of the VTP (or carrot) has been determined, the heading to that point has to be calculated via

$$\chi_{VTP} = \arctan\left(\frac{y_{VTP} - y_{V_e}}{x_{VTP} - x_{V_e}}\right). \tag{28}$$

A P-controller is used to align the vehicle with the desired heading χ_{VTP}. In order to improve performance and to exploit the continuity of the generated trajectories, the controller is augmented with a feedforward designed to use the available curvature information κ to precondition the commanded roll angle ϕ. The curvature for this feedforward term is obtained by evaluating the curvature at point P. By assuming that the rotational dynamics are sufficiently fast, stable and have a unity gain, the feeforward term can be obtained by inserting Eq. (2) into Eq. (1)

$$dy = \underbrace{K_p \cdot (\chi_{VTP} - \chi)}_{\text{feedback}} + \underbrace{\arctan\left(\frac{V_I^2 \cdot \kappa}{g}\right)}_{\text{feedforward}}. \tag{29}$$

2. Cross-track Error Feedback Control with Feedforward Command

The rotational dynamics of Eq. (24) can be ignored, if it is assumed that they are sufficiently fast and stable. As stated in Table 1, this is a valid assumption. Based on this reasoning a cross-track error feedback controller which controls the cross-track error and velocity (Δy_{FS}, $\Delta \dot{y}_{FS}$) is designed. Just as in the case of the carrot chasing controller, a feedforward term is introduced, which computes an output based on the curvature of the track at point P, in order to pre-steer curves of the trajectory.
The complete cross-track error feedback controller therefore takes the following form:

$$dy = \underbrace{K_d \cdot \Delta y_{FS} + K_v \cdot \Delta \dot{y}_{FS}}_{\text{feedback}} + \underbrace{\arctan\left(\frac{V_I^2 \cdot \kappa}{g}\right)}_{\text{feedforward}} \tag{30}$$

C. Parameter design and analysis

In order to gain some insights into the parameter design for both the cross-track error feedback and carrot chasing controllers, both closed-loop systems were linearized. For this Eq. (24)-(26) were combined with the controllers given in Eq. (29) and Eq. (30), in order to form the complete non-linear control loop. The resulting two systems were linearized around a condition, equivalent to following a straight line ($\chi_T = 0$, $\dot{\chi}_T = 0$) which results in Eq. (31) for the carrot chasing controller and Eq. (32) for the cross-track-feedback controller.

$$\frac{d}{dt}\begin{pmatrix}\Delta y_{FS}\\ \Delta\dot{y}_{FS}\\ \phi\\ p\\ b_{1_s}\\ \dot{b}_{1_s}\end{pmatrix}=\begin{pmatrix}0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & g & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & -L_p & L_{b_{1s}} & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ L_{dy}k_{dy}\frac{K_p}{t_{pred}V_I} & L_{dy}k_{dy}\frac{K_p}{V_I} & k_\phi & -L_{\dot{b}_{1s}} & L_{b_{1s}} & L_{\dot{b}_{1s}}\end{pmatrix}\cdot\begin{pmatrix}\Delta y_{FS}\\ \Delta\dot{y}_{FS}\\ \phi\\ p\\ b_{1_s}\\ \dot{b}_{1_s}\end{pmatrix} \tag{31}$$

$$\frac{d}{dt}\begin{pmatrix}\Delta y_{FS}\\ \Delta\dot{y}_{FS}\\ \phi\\ p\\ b_{1_s}\\ \dot{b}_{1_s}\end{pmatrix}=\begin{pmatrix}0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & g & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & -L_p & L_{b_{1s}} & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ L_{dy}k_{dy}K_d & L_{dy}k_{dy}K_v & k_\phi & -L_{\dot{b}_{1s}} & L_{b_{1s}} & L_{\dot{b}_{1s}}\end{pmatrix}\cdot\begin{pmatrix}\Delta y_{FS}\\ \Delta\dot{y}_{FS}\\ \phi\\ p\\ b_{1_s}\\ \dot{b}_{1_s}\end{pmatrix} \tag{32}$$

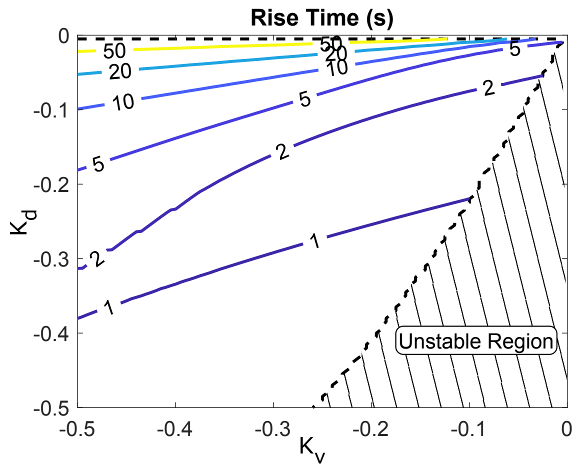
For the linearization of the carrot chasing controller as given in Eq. (29), the flight-path azimuth χ has to be calculated. In the case of straight line following, this angle can be calculated via simple geometry

$$\chi = \arcsin\left(\frac{\Delta y_{FS}}{V_I}\right). \tag{33}$$

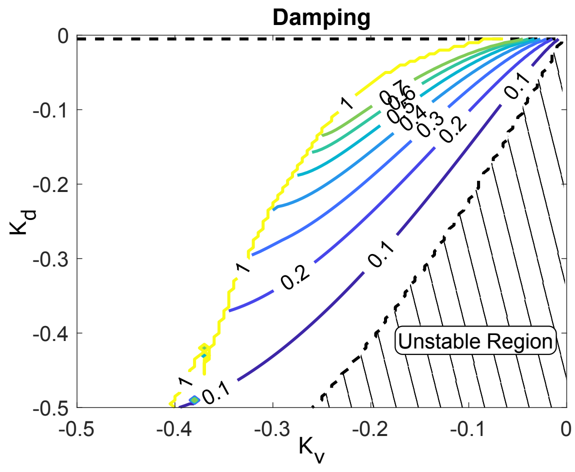
A couple of interesting remarks can be deducted from these linearizations. Firstly it can be concluded that the two resulting linear systems look very much the same in both cases. Both system utilize the lateral position error Δy_{FS} and the lateral position error velocity $\Delta\dot{y}_{FS}$ as a feedback. Additionally, the parameters of the carrot chasing controller can be matched to those of the cross-track error feedback controller via

$$K_d = \frac{K_p}{t_{pred}V_I} \tag{34}$$

$$K_v = \frac{K_p}{V_I}. \tag{35}$$



(a) Rise time of Δy_{FS}



(b) Damping of Δy_{FS}

Fig. 7 Rise time, damping and stable/unstable region in reference to the cross-track error feedback gains K_d and K_v.

Differences appear however in the speed dependency of the two algorithms: On the one hand, note that the linearization for the cross-track error feedback controller in Eq. (32) does not depend on the speed V_I. This is due to the fact that the flight-path angle azimuth rate $\dot{\chi}$ carries a V_I in its denominator in Eq. (25), which cancels the velocity dependence of the linearized Eq. (26). In the linearization, the only term which is left is the factor g in matrix position (2,3). As this is the only place in which the speed dependence explicitly appears, this cancellation means that the speed dependency for the cross-track error feedback controller is no longer present.
On the other hand, note that the linearization for the carrot chasing controller Eq. (31) reintroduces such a speed dependence, at matrix position (6,1) and (6,2). This happens due to the fact, that the VTP as well as χ are depending on V_I, which consequently emerges in the linearized Eq. (31). This shows the fact that the linear dynamics of the cross-track error feedback structure is inherently less dependent on the velocity then the carrot chasing control structure.

Based on the linearization for the cross-track error feedback controller, some key metrics can be plotted over the range of different K_D and K_V values. Because the complete system consists of three different pole pairs as described in (25) - (26), which are additionally very well separated, the damping of the pole pair corresponding to the path error dynamics can be formulated. The damping and the rise time for some parameter combinations are shown in Fig. 7. Additionally, the region for which the linearized system is unstable, is marked by the dotted line.

VI. Simulation and Results

The designed algorithms were implemented in MATLAB/Simulink® and subsequently auto-coded and integrated into the software framework of the ACT/FHS. The complete program was tested in the **A**ir **Ve**hicle **S**imulator (AVES) - DLR's simulator center, which features a replica of the ACT/FHS cockpit connected to a one-to-one copy of the flight control hardware, a simulation of the helicopter and a sensor simulation [26].
Fig. 8 shows a typical landing approach trajectory consisting of 120 waypoints which was simulated in the AVES. The waypoints and associated velocities are chosen, such that they take the helicopter from 60 knots forward speed to a near-hover condition. Waypoints are spaced approximately 50 meters apart. In order to test the developed algorithms, a fully automatic flight with these waypoints as inputs was simulated with the developed trajectory generation algorithm and automatic controllers.

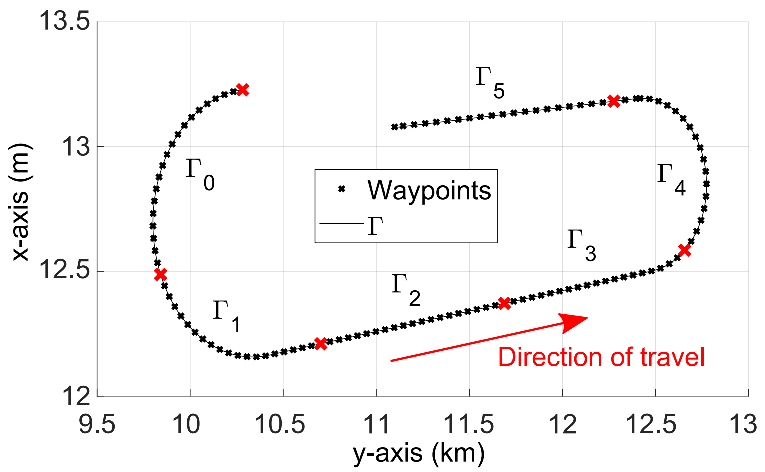
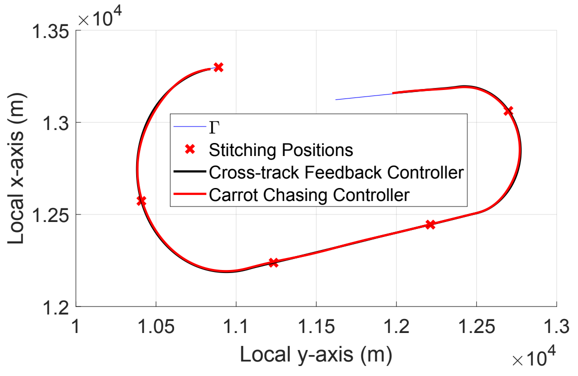


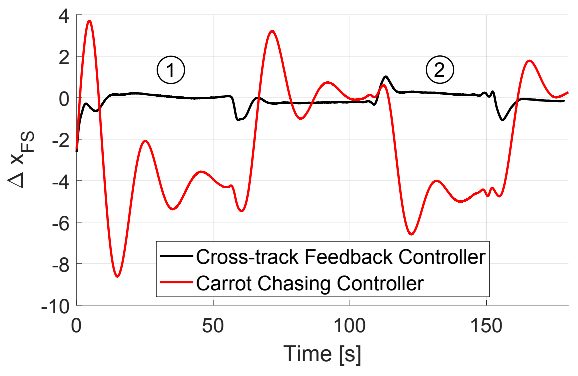
Fig. 8 Hover approach trajectory used for testing of the trajectory generation algorithm and path following control. Stitching locations of the different trajectories Γ_0 - Γ_5 are marked with red crosses.

A. Real-time Adaptation

The presented real-time adaptation for planning cubic splines was used to construct a trajectory through the waypoints of the problem as presented in Fig. 8. The configuration of the algorithm was chosen such that each path-segment includes 20 waypoints. The next path segment is planned when the second-to-last waypoint of the current path segment is reached. The navigational module in which the path planning is executed runs with a period of 32 ms. In order to assess the performance of the real-time adaptation of the cubic spline algorithm, the runtime of the navigational algorithm was logged and analyzed.



(a) Cubic Spline path with simulated trajectory of both controllers



(b) Lateral distance error Δy_{FS} in Frenet-Serret frame for both controller simulations

Fig. 9 Path following performance for carrot chasing and cross-track error feedback controller during a simulated approach trajectory.

During the first switch-on of the algorithm, and the subsequent calculation of the trajectory through the first 20 waypoint, the runtime of the whole navigational algorithm was determined to be approximately 9 ms, which however includes other operations which normally take around 3 ms. Additionally, the first call of the trajectory planning algorithm is expensive due to the allocation of memory and the initialization of variables. A summary of statistics of the runtime during the subsequent timesteps at which the algorithm was run, is compiled in Table 2. With a mean of approximately 7 ms and maximum peaks to approximately 10 ms, the algorithm performs satisfactory, as it stays well below the maximum permissible 32 ms available to the module. It is hypothesized that the spikes in computational time can be observed due to other calculations being executed during the runtime of the navigational module, as these spike occur regularly and seem to arbitrarily fall into the expected frame of the trajectory generation algorithm.
These results show that the goal of real-time path planning for large waypoint lists or dynamically arriving new waypoint lists, can be split up via the proposed trajectory generation algorithm. This ensures a consistent and well-behaved runtime during normal operation and in the case of the unexpected arrival of new waypoint lists and is essential for a reliable path-planning module for the ACT/FHS.

Table 2 Statistics of the execution time for the proposed path smoothing algorithm

Mean Value	Standard Deviation	Max Value	Min Value
7.00 ms	2.75 ms	9.58 ms	3.91

B. Controller

The carrot chasing controller as well as the cross-track error feedback controller presented in section V were tested on the trajectory shown in Fig. 8 during tests in the AVES. The gains of the cross-track error feedback controller were tuned via the Robust Control Toolbox of MATLAB© with the design goal of achieving a minimal damping of $\zeta = \frac{\sqrt{2}}{2}$ and a time constant of $\tau \approx 5s$. Similar gains can been found via the plots presented in 7, validating the analysis of section V.C. The gains for the carrot chasing controller were manually tuned with a prediction time of the VTP set to $t_{pred} = 3$ s.
Results of these tests are shown in Fig. 9. The trajectory of both tested controllers can be seen in Fig. 9a and the error plots in Fig. 9b show the achieved tracking performance.

The cross-track error feedback controller exhibits a better tracking performance during this task, mostly due to the better tuning process for this controller. During transients from straight to curved flight and vice-versa, some dynamics are introduced to both controllers. However due to the feedforward term of of the two, the lateral path deviation is low for both controllers. The carrot chasing controller exhibits a negative lateral path error during curved flight. This

behavior stems from the fact that the VTP of the carrot chasing controller hurries ahead of the helicopter, thereby "dragging" the helicopter in a smaller circle behind it. This can be be intuitively visualized by imagining a dog running in a circle, while the owner is dragged behind. However, because of the fact that the leash forms a chord of the circle the dog is running, the owner will describe a smaller circle. Because of this behavior, the carrot chasing algorithm will exhibit a negative lateral path error during left-hand turns, and a positive error during right-hand turns even with the feedforward term as described in section V. This characteristic of the carrot chasing algorithm is dependent on the prediction time t_{pred}. The higher this time is, the more lateral path deviations in turns is reached. Therefore in the case of the carrot chasing algorithm, a design trade-off must be reached. The prediction time could be matched via Eq. (34)-(35) to match the dynamics of the cross-track error controller. However if the values for K_d and K_d are used the prediction time would turn out to be roughly $t_{pred} = 10$ seconds, which in turn would result in an even higher lateral deviation during turning flight. This indicates, that the carrot chasing controller is less suitable for this application then the cross-track error controller.
One interesting observation is that the carrot chasing algorithm seems to have different transient dynamics in the second turn, beginning at approximately 130 s, marked with a ② in Fig. 8, when compared to its first turn, beginning at 0 s marked with ①. This behavior can be attributed to the higher velocity of first curve when compared to the second curve, and the fact that the carrot chasing controller is velocity dependent as analyzed in section V.C. In contrary, the cross-track error feedback controller exhibits the same dynamics for every flight speed. This confirms the analysis performed in Chapter V.

VII. Conclusion and Outlook

Developing methods and tools for helicopter to generate and follow smooth trajectories based on waypoints are essential for fast and accurate waypoint following. The proposed adaptation of the existing natural cubic spline method for interpolating waypoint based flight paths has shown its potential to be suitable for:

1) application with limited computational power
2) tight real-time constraints, in the tested scenarios
3) scenarios in which en-route replanning of waypoints occur

The generated trajectory still exhibits a smooth characteristic with C^0, C^1, and C^2 continuity. It has been shown that the implemented version of this algorithm works well in the real-time environment of the ACT/FHS flight-hardware for the examined test case.
A carrot chasing control strategy as well as a cross-track error feedback controller for guiding the helicopter along the generated path have been analyzed, implemented and tested in simulation. It was shown that for the linearized case, the two control strategies both utilize the lateral path error and lateral path error velocity as feedback. However the carrot chasing controller is velocity dependent and also shows inferior performance during curved flight due to the characteristics of the calculation of the Virtual Target Point. The cross-track error feedback controller with feedforward action however exhibits good performance throughout the whole speed range, because of its independence of the velocity as shown in Chapter V.
In the future, it is planned to extend the proposed algorithm in three dimensions together with a velocity reference trajectory. Additionally, in middle-term future, flight tests on the ACT/FHS are planned to validate the performance of the developed algorithms.

References

[1] Whalley, M. S., Takahashi, M. D., Fletcher, J. W., Moralez, E., Ott, L. C. R., Olmstead, L. M. G., Savage, J. C., Goerzen, C. L., Schulein, G. J., Burns, H. N., and Conrad, B., "Autonomous Black Hawk in Flight: Obstacle Field Navigation and Landing-site Selection on the RASCAL JUH-60A," *Journal of Field Robotics*, Vol. 31, No. 4, 2014, pp. 591–616. doi:10.1002/rob.21511.

[2] Hota, S., and Ghose, D., "Optimal geometrical path in 3D with curvature constraint," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2010, pp. 113–118. doi:10.1109/IROS.2010.5653663.

[3] Choi, J.-w., Curry, R., and Elkaim, G., "Continuous Curvature Path Generation Based on Bezier Curves for Autonomous Vehicles," *IAENG International Journal of Applied Mathematics*, 2010.

[4] Sencer, B., Ishizaki, K., and Shamoto, E., "A curvature optimal sharp corner smoothing algorithm for high-speed feed motion generation of NC systems along linear tool paths," *The International Journal of Advanced Manufacturing Technology*, Vol. 76, No. 9-12, 2015, pp. 1977–1992. doi:10.1007/s00170-014-6386-2.

[5] Dubins, L. E., “On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents,” *American Journal of Mathematics*, Vol. 79, No. 3, 1957, pp. 497–516.

[6] Chitsaz, H., and LaValle, S. M., “Time-optimal paths for a Dubins airplane,” *2007 46th IEEE Conference on Decision and Control*, IEEE, 2007, pp. 2379–2384. doi:10.1109/CDC.2007.4434966.

[7] Schneider, V., Piprek, P., Schatz, S. P., Baier, T., Dörhöfer, C., Hochstrasser, M., Gabrys, A., Karlsson, E., Krause, C., Lauffs, P. J., et al., “Online trajectory generation using clothoid segments,” *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on*, IEEE, 2016, pp. 1–6.

[8] Choi, J.-w., Curry, R., and Elkaim, G., “Path Planning Based on Bézier Curve for Autonomous Ground Vehicles,” *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, IEEE, 2008, pp. 158–166. doi:10.1109/WCECS.2008.27.

[9] Yang, K., and Sukkarieh, S., “An Analytical Continuous-Curvature Path-Smoothing Algorithm,” *IEEE Transactions on Robotics*, Vol. 26, No. 3, 2010, pp. 561–568. doi:10.1109/TRO.2010.2042990.

[10] Conte, G., Duranti, S., and Merz, T., “Dynamic 3D path following for an autonomous helicopter,” *IFAC Proceedings Volumes*, Vol. 37, No. 8, 2004, pp. 472–477. doi:10.1016/S1474-6670(17)32021-9.

[11] I. Rhee, S. P., and Ryoo, C.-K., “A Tight Path Following Algorithm of an UAS Based on PID Control,” *SICE Annual Connference*, 2010, pp. 1270–1273.

[12] Ratnoo, A., Sujit, P. B., and Kothari, M., “Adaptive Optimal Path Following for High Wind Flights,” *IFAC Proceedings Volumes*, Vol. 44, No. 1, 2011, pp. 12985–12990. doi:10.3182/20110828-6-IT-1002.03720.

[13] Dauer, J. C., Faulwasser, T., Lorenz, S., and Findeisen, R., “Optimization-based feedforward path following for model reference adaptive control of an unmanned helicopter,” *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013, p. 5002.

[14] Yang, K., Sukkarieh, S., and Kang, Y., “Adaptive nonlinear model predictive path tracking control for a fixed-wing unmanned aerial vehicle,” *AIAA Guidance, Navigation, and Control Conference*, 2009, p. 5622.

[15] Park, S., Deyst, J., and How, J., “A New Nonlinear Guidance Logic for Trajectory Tracking,” *AIAA Guidance, Navigation, and Control Conference and Exhibit*, American Institute of Aeronautics and Astronautics, Reston, Virigina, 2004. doi:10.2514/6.2004-4900.

[16] Nelson, D. R., Barber, D. B., McLain, T. W., and Beard, R. W., “Vector Field Path Following for Miniature Air Vehicles,” *IEEE Transactions on Robotics*, Vol. 23, No. 3, 2007, pp. 519–529. doi:10.1109/TRO.2007.898976.

[17] Sujit, P. B., Saripalli, S., and Sousa, J. B., “Unmanned Aerial Vehicle Path Following: A Survey and Analysis of Algorithms for Fixed-Wing Unmanned Aerial Vehicless,” *IEEE Control Systems*, Vol. 34, No. 1, 2014, pp. 42–59. doi:10.1109/MCS.2013.2287568.

[18] Takahashi, M. D., Whalley M.D., Matthew S., Mansur, H., Ott, L. C., Minor, M. J., and Morford, M. Z., “Autonomous Rotorcraft Flight Control with Multilevel Pilot Interaction in Hover and Forward Flight,” *Journal of the American Helicopter Society*, Vol. 62, 2017, pp. 1–13.

[19] Lantzsch, R., Greiser, S., Wolfram, J., Wartmann, J., Müllhäuser, M., Lüken, T., Döhler, H.-U., and Peinecke, N., “Results of the pilot assistance system “Assisted Low-Level Flight and Landing on Unprepared Landing Sites” obtained with the ACT/FHS research rotorcraft,” *Aerospace Science and Technology*, Vol. 45, 2015, pp. 215–227.

[20] Bartels, R. H., Beatty, J. C., and Barsky, B. A., *An Introduction of the Use of Splines in Compute Graphics*, University of Waterloo Computer Science Department, Waterloo, 1983.

[21] Zimmermann, M., “Sensor Based Online Path Planning for Rotorcraft Landing,” *Deutscher Luft- und Raumfahrtkongress*, 2016.

[22] Wikipedia Contributors, “Wikipedia, The Free Encyclopedia – Circle: Construct a circle through 3 noncollinear points,” , 2004. URL `https://en.wikipedia.org/wiki/Circle#Construct_a_circle_through_3_noncollinear_points`, [Online; accessed 10. April 2018].

[23] Chicella, V., Xargay, E., Dobrokhodov, V., Kaminer, I., Pascoal, A., and Hovakimyan, N., “Geometric 3D Path-Following Control for a Fixed-Wing UAV on SO(3),” *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2011, p. 6415.

[24] Heffley, R. K., Bourne, S. M., Curtiss, H., Hindson, W. S., and Hess, R. A., “Study of Helicopter Roll Control Effectiveness Criteria,” *NASA CR177404*, 1986.

[25] Tabatabaei, S. A. H., Yousefi-koma, A., Ayati, M., and Mohtasebi, S. S., “Three dimensional fuzzy carrot-chasing path following algorithm for fixed-wing vehicles,” *Robotics and Mechatronics (ICROM), 2015 3rd RSI International Conference on*, IEEE, 2015, pp. 784–788.

[26] Duda, H., Advani, S. K., and Potter, M., “Design of the DLR AVES research flight simulator,” *AIAA Modeling and Simulation Technologies (MST) Conference*, 2013, p. 4737.