RRT*-based Trajectory Planning for Fixed Wing UAVs using Bézier Curves

Martin Seemann, TU Dresden, Germany, martin.seemann@tu-dresden.de
Klaus Janschek, TU Dresden, Germany, klaus.janschek@tu-dresden.de

Abstract

We present a trajectory planning method for fixed-wing aircraft based on the RRT* randomized planning framework. Our main contribution is a trajectory segment design technique that relies on the differential flatness property of the system model. This enables fast analytical evaluation of state and input trajectories. To simplify the integration with an underlying trajectory tracking controller, adjacent trajectory segments are designed to be $\mathcal{C}^2$ continuous, which translates into at least $\mathcal{C}^0$ continuity for all relevant system variables. In simulation experiments, real-time performance is achieved for typical application scenarios on hardware equivalent to decent mobile computing units.

1 Introduction

In recent years, research in aerial robotics, i.e. Unmanned Aerial Vehicles (UAVs) has focused mainly on small multirotor aircraft. While their unique capabilities like Vertical-Takeoff-and-Landing (VTOL) and hovering may be beneficial for some applications, fixed-wing configurations usually allow for higher speeds, longer ranges and higher payloads, which make them the better choice for tasks like mapping, area surveillance or the deployment of pesticides and similar substances in agriculture. They are, however, more complex to guide through obstacle-rich environments due to non-holonomic constraints and a cruising speed that is bounded below. Relying on the absence of obstacles, as in traditional high altitude operation, does not allow for truly autonomous operation in very low altitude conditions or even in indoor environments, which are of practical interest.

For ground robots it is often reasonable to ignore vehicle dynamics in the planning process. However, this can be fatal for fixed-wing aircraft in areas of high obstacle density. Ensuring dynamic feasibility at the planning level is therefore a mandatory requirement. Due to the rather high dimension of the state space, a randomized search strategy like the *Rapidly Exploring Random Tree* (RRT) algorithm [1] seems advantageous, which has already been applied successfully to various mobile robotics and manipulator planning problems. Unfortunately, it has been proven in [2] that the RRT algorithm does not converge to an optimal solution (in a stochastical sense, i. e. probability of convergence is zero). To overcome this issue, in the same publication the authors presented RRT* as an extension to the original algorithm. In this paper we apply the RRT* algorithm to a fixed-wing aircraft. We focus on the trajectory planning problem. A sufficiently detailed map of the (static) environment is assumed to be available either from a priori knowledge, from on-board sensor processing or a combination of both and will not be described here. Our main contribution is a trajectory design technique relying on Bézier curves and differential flatness of the model equations for utilization within the RRT* global optimization framework.

The paper is organized as follows. In section 2 we give an outline of the RRT* algorithm and review several different implementations form literature. Section 3 presents the motion model that is used by our planner. As an essential property, differential flatness of the model is established. The design of feasible trajectory segments is described in section 4 and results from simulation experiments, including the discussion of application specific performance improvements are given in section 5. Section 6 summarizes the results and gives directions for further enhancements.

2 The RRT* algorithm and related work

Algorithm 1: Basic RRT* Algorithm (Please refer to [2] for detailed symbol explanations)

```
$\mathcal{V} \leftarrow \{x_{\text{init}}\}; \mathcal{E} \leftarrow \emptyset$;
while $\mathcal{V} \cap X_{\text{goal}} = \emptyset$ and TimeAvailable() do
    $x_{\text{rand}} \leftarrow$ SampleFrom($\mathcal{X}$);
    $x_{\text{nearest}} \leftarrow$ Nearest(($\mathcal{V}, \mathcal{E}$), $x_{\text{rand}}$);
    $e_{\text{new}} \leftarrow$ Steer($x_{\text{nearest}}, x_{\text{rand}}$);
    if ObstacleFreeAndFeasible($e_{\text{new}}$) then
        $x_{\text{new}} \leftarrow e_{\text{new}}.to$;
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{x_{\text{new}}\}; \mathcal{E} \leftarrow \mathcal{E} \cup \{e_{\text{new}}\}$;
        $X_{\text{near}} \leftarrow$ Near(($\mathcal{V}, \mathcal{E}$), $.x_{\text{new}}$);
        for all $x_{\text{near}} \in X_{\text{near}}$ do
            $e_{\text{old}} \leftarrow x_{\text{near}}.parentEdge$;
            $e_{\text{rewire}} \leftarrow$ Connect($x_{\text{new}}, x_{\text{near}}$);
            if ObstacleFreeAndFeasible($e_{\text{rewire}}$) and
            Cost($x_{\text{near}}$) > Cost($x_{\text{new}}$) + Cost($e_{\text{rewire}}$) then
                $x_{\text{near}}.parentEdge \leftarrow e_{\text{rewire}}$;
                $\mathcal{E} \leftarrow$ ReplaceEdge($\mathcal{E}, e_{\text{old}}, e_{\text{rewire}}$);
```

RRT* is a randomized global optimization method specially suited for path and trajectory planning. Its pseudo code is listed in **Algorithm 1**. The algorithm incrementally grows a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ of nodes $\mathcal{V}$, connected by feasible trajectory segments $\mathcal{E}$ in the obstacle-free subset of a state space $\mathcal{X}$ from an initial state $x_{\text{init}} \in \mathcal{V}$ to a goal region $X_{\text{goal}} \subset \mathcal{X}$. The algorithm iterates until an initial trajectory has been found. If more time is available, it may be used to improve this solution. The basic RRT algorithm stops at line 8, i. e. after *extending* the tree with a single new node x_{new}. RRT* attempts to *rewire* nearby nodes to the newly added one in order to reduce accumulated edge costs for existing nodes. This modification guarantees convergence to a minimal-cost solution with probability one. Both variants rely on an unspecified *Local Planner* for the methods `Steer` and `Connect` to connect pairs of states with trajectory segments. These segments have to be feasible, i. e. they obey the equations of motion and any additional constraints, and should ideally be minimum cost trajectories. In other words, each local planning step is a nontrivial optimization problem by itself that would be typically solved numerically via shooting [3], direct collocation [4] or similar approaches. However, practical RRT* applications demand for a much faster solution, since local planning is usually invoked thousands of times per query.

The original RRT* [2] therefore disregarded vehicle dynamics completely and instead used straight line connections between states. Subsequent authors approximated their systems by dynamics with a known closed-form solution to the optimal control problem, e. g. [5]. In this publication, an aircraft model was constructed from a double integrator and a Dubins car for vertical and planar motion, respectively. While this may be appropriate for conventional (passenger) planes, we do not consider it to be appropriate for a more agile UAV. In [6] a LQR feedback design is used that is applicable to all linear systems with quadratic cost formulations. An alternative, closed-form solution for linear system with a nilpotent system matrix is presented in [7] and involves solving a polynomial, whose degree is quadratic in the number of states. To a moderate extent, both methods are applicable to nonlinear systems via linearization around the target state.

When dropping the demand for optimality in the local planner, observation suggests that RRT*'s own convergence property still holds, but the convergence rate decreases. Several authors, including ourselves, accept this compromise to allow a broader range of systems and local planning techniques. In [8] Bézier curves are utilized to design local path segments and forwarded as a reference input to a tracking controller. A numerical simulation of the vehicle dynamics, including the tracking controller is carried out to get the resulting state and input trajectories, which are then checked for constraint violations. While this is in principle a very powerful approach, it is also computationally demanding and therefore too slow for real-time implementation. The approach presented in this paper shares the Bézier formulation for the description of flight paths, but instead of simulating a tracking controller we get the corresponding input trajectories by exploiting differential flatness of the system model. Feasibility is then tested by checking if states and inputs stay in their allowed intervals over the whole trajectory. The associated segment cost incorporates trajectory length as well as steering effort. Recently it came to our attention that the combination of differential flatness and the RRT* has already been applied to a quadrotor in [9]. However this work does not handle state and input constraints explicitely and only the path length is considered as a cost factor.

3 3DOF Aircraft model

3.1 Model Description

In our opinion, long-term trajectory planning should not be based on a highly complex 6-DOF (degree of freedom) aircraft model, since there are too many influences (uncertain inertial and aerodynamic parameters, gusts, fuel slosh, etc.) that cannot be predicted accurately and might render the planned trajectory unsafe to execute even if it seemed feasible during planning. We therefore employ a simplified 3-DOF model, based on [10] (chapter 6.5) that describes the motion of the center of mass (COM) of an aircraft by

$$\begin{aligned} \dot{x} &= v\cos(\gamma)\cos(\chi) & \dot{\gamma} &= \frac{\mathrm{g}}{v}\left(n\cos(\mu) - cos(\gamma)\right) \\ \dot{y} &= v\cos(\gamma)\sin(\chi) & & \\ \dot{z} &= -v\sin(\gamma) & \dot{\chi} &= \frac{\mathrm{g}}{v}\cdot\frac{n\sin(\mu)}{\cos(\gamma)}\,. \end{aligned} \tag{1}$$

Here, the COM position $\boldsymbol{p} = (x, y, z)^{\mathrm{T}}$ is described w.r.t. a North-East-Down (NED) frame $\{\mathrm{e}\}$ attached to a flat, nonrotating earth. The angles χ and γ describe the instantaneous azimut and elevation of the flight path. They also define a path-aligned frame $\{\mathrm{k}\}$ with its axis $\hat{X}_{\mathrm{k}}$ always pointing into the direction of motion. These definitions are depicted in **Figure 1**. The airspeed v, a unit-less load factor n (lift-to-weight ratio) and the path bank angle μ constitute the model inputs. The latter can be thought of as an approximation of the vehicle's bank angle if both side slip and angle-of-attack are small. The model does well capture basic fixed-wing motion behaviour without the need for any inertial or aerodynamic parameters. Of course, it would be overly simplified for planning aerobatic maneuvers, which – according to the argument given above – is not our intention.

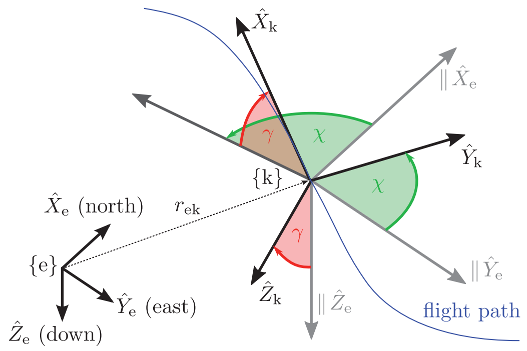


Figure 1: Definition of relevant coordinate frames

3.2 Differential Flatness Property of the 3-DOF-Model

According to [11] a system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ is said to be differentially flat if one could find a (flat) output $\boldsymbol{\sigma}$ and relations $\boldsymbol{x}(t) = \xi_{\boldsymbol{x}}(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, ...\boldsymbol{\sigma}^{(\alpha)})$ and $\boldsymbol{u} = \xi_{\boldsymbol{u}}(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, \ddot{\boldsymbol{\sigma}}, ..., \boldsymbol{\sigma}^{(\beta)})$ to express both state and input trajectories in terms of the flat output and a finite number of its derivatives. In case the flat output has any meaningful physical interpretation, trajectory design reduces to choosing an appropriate trajectory $\boldsymbol{\sigma}(t)$ (e. g. a polynomial) and subsequently computing state and input trajectories from $\xi_{\boldsymbol{x}}$ and $\xi_{\boldsymbol{u}}$, both without the need for (numerically) solving an ODE system.
In case of our model (1), a physically meaningful flat output is the position $\boldsymbol{p} = (x, y, z)^{\mathrm{T}} =: \boldsymbol{\sigma}$. The states expressed by $(\boldsymbol{\sigma}, \dot{\boldsymbol{\sigma}}, ...\boldsymbol{\sigma}^{(\alpha)})$ become

$$\begin{aligned} (x, y, z)^{\mathrm{T}} &= (\sigma_1, \sigma_2, \sigma_3)^{\mathrm{T}} \\ \chi &= \operatorname{atan2}(\dot{\sigma}_2, \dot{\sigma}_1) \\ \gamma &= -\operatorname{atan}\left(\frac{\dot{\sigma}_3}{\sqrt{\dot{\sigma}_1^2 + \dot{\sigma}_2^2}}\right) . \end{aligned} \tag{2}$$

The airspeed is

$$v = \sqrt{\dot{\sigma}_1^2 + \dot{\sigma}_2^2 + \dot{\sigma}_3^2} = \sqrt{\boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{\sigma}} \tag{3}$$

and the remaining inputs can now be written as

$$n = \sqrt{\zeta_1^2 + \zeta_2^2} \quad \mu = \operatorname{atan}\left(\frac{\zeta_1}{\zeta_2}\right) \tag{4}$$

in terms of the two auxiliary variables

$$\zeta_1 = \frac{v}{\mathrm{g}}\dot{\chi}\cos(\gamma) \quad \zeta_2 = \frac{v}{\mathrm{g}}\dot{\chi} + \cos(\gamma) \tag{5}$$

where $\dot{\chi}$ is

$$\dot{\chi} = \frac{\dot{\sigma}_1\ddot{\sigma}_2 - \dot{\sigma}_2\ddot{\sigma}_1}{\dot{\sigma}_1^2 + \dot{\sigma}_2^2} . \tag{6}$$

4 Flatness-based Trajectory design

In this section, we present a trajectory design method based on *Bézier curves* for local planning within the RRT* framework. The RRT* solution trajectory is then a concatenation of Bézier curves, i. e. a *Bézier spline*. Without loss of generality, we let all curves start at $t = 0$.

4.1 Time-Reparameterized Bézier Curves

A common approach for a flat output trajectory design is the use of parameterized curves. In this work, we use Bézier curves

$$\boldsymbol{p}(\tau) = \sum_{i=0}^{N} \binom{N}{i} (1-\tau)^{(N-i)} \tau^i \boldsymbol{c}_i \quad \tau \in [0, 1] , \tag{7}$$

which are uniquely determined by their control points $\boldsymbol{c}_0, ..., \boldsymbol{c}_N$. The polygon formed by these points allows for very intuitive control of the shape of the curve, which is the predominant reason for choosing this formulation. Section 4.3 will motivate the selection of an appropriate order N and the positioning of the control points.
An inconvenience that Bézier curves share with most parametric curves is their definition in terms of an artificial parameter τ that is not in a straightforward way related to the distance traveled along the curve. Equating the curve directly with our flat output $\boldsymbol{\sigma}$ would thus result in large variations of the airspeed $v = |\partial\boldsymbol{p}/\partial\tau|$, which is neither desirable nor realizable with adequate effort. To overcome this issue, we aim for a time reparameterization $\boldsymbol{p}(t) = \boldsymbol{p}(\tau(t))$. This way, we could realize any velocity profile. For simplicity, we chose a constant airspeed $v(t) \equiv V$, effectively leaving n and μ as the only inputs to our motion model. The reparameterization $\tau(t)$ is then defined by

$$\dot{\tau} = V \cdot \left| \frac{\partial\boldsymbol{p}}{\partial\tau}\bigg|_{\tau=\tau(t)} \right|^{-1} . \tag{8}$$

This is equivalent to finding an *arc length reparameterization* – a problem of continued interest in computer graphics and animation, which unfortunately has no closed form solution except for some very special cases. It would be straightforward to solve (8) numerically for every evaluation of the curve, but this would be too time-consuming for our intended application. Several methods for an approximate reparameterization have been proposed, e. g. [12], [13]. They all yield good agreement with the original curve. However, when evaluating the derivatives $\dot{\boldsymbol{p}}, \ddot{\boldsymbol{p}}$ to compute state and input trajectories from (2) – (6), the approximation errors usually become less desirable. A more suitable approximation quality measure is therefore the velocity error $|\dot{\boldsymbol{p}} - V|$. In **Figure 2** the errors for approximating $\tau(t)$ by a piecewise linear function, a piecewise cubic function and a piecewise quintic function are shown for a randomly chosen Bézier curve. While all approximations show a rather low error in $\tau(t)$ (observe the axis scale), the maximum velocity error for the piecewise linear function approaches 35 %. Although the piecewise quintics show only little advantage over the cubics, we selected the former, because quintics yield better agreement for $\ddot{\tau}$, which is the highest derivative used in our computation.

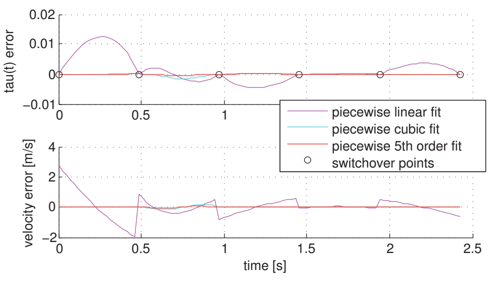


Figure 2: Reparameterization errors for a sample curve traversed at $V = 8\,\mathrm{m/s}$

The values τ_i, $\dot{\tau}_i$ and $\ddot{\tau}_i$ at the subdivision boundaries t_i ($i = 0...M$), required for fitting the quintics, are computed numerically from (8). This implies that despite differential flatness, state and input trajectories cannot be evaluated in a closed form anymore. However, the amount of numeric computations is quite low and does not depend on the number of trajectory evaluations. We found $M = 5$ subdivisions to be a good compromise between precision and computational efficiency for the short trajectory segments generated by RRT*. Putting it all together, the necessary steps to evalute the state and input trajectories at time t are:

1. Identify trajectory section i with $t_{i-1} \leq t < t_i$
2. Compute $\tau(t)$ using the corresponding quintic
3. Evaluate $\boldsymbol{\sigma}(t) = \boldsymbol{p}(\tau(t))$ as well as $\dot{\boldsymbol{\sigma}}$ and $\ddot{\boldsymbol{\sigma}}$
4. Use $\xi_{\boldsymbol{x}}$ and $\xi_{\boldsymbol{u}}$ to compute $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$

4.2 Constraints and Trajectory Costs

For a time-reparameterized Bézier curve to be a feasible flight path, there must be an underlying controller that either tracks our model inputs (n, μ), the curve itself or a combination of both with sufficient accuracy. In this section we identify the constraints that a curve must adhere to for trackability by mainstream control techniques. The first set of conditions is continuity in the state and input trajectories over the whole RRT* solution path from the initial node to the goal. This is obvious for each component of the state vector. As the μ input is related to the bank angle, discontinuities in its trajectory would be rather bad, too. For n, which can be roughly sought of as pitch rate, small jumps might not harm that much, but continuity would clearly lower the stress on the controller. Assuming a sufficiently high order N of the Bézier curves, these conditions are met within each trajectory segment. For ensuring continuity across segment boundaries, there apply restrictions to the placement of control points, which are implemented by the design process described in section 4.3.

The second set of constraints are upper and lower limits on γ (limited climb rate/descent), n (limited lift) and μ (banking limit), which have to be met throughout the whole trajectory:

$$\begin{aligned} \gamma_{\min} \leq \gamma(t) &\leq \gamma_{\max} \\ n_{\min} \leq n(t) &\leq n_{\max} \\ |\mu(t)| &\leq \mu_{\max} \end{aligned} \tag{9}$$

The cost g_k associated with a segment k is comprised of its length L_k, which due to constant V is equivalent to execution time, and the "steering effort":

$$g_k = w_L L + w_n \int (n(t) - 1)^2 \mathrm{d}t + w_\mu \int \mu^2(t) \mathrm{d}t \tag{10}$$

The weights w_* may be chosen to balance the objectives according to practical considerations. The integrals are solved numerically using an adaptive quadrature routine to medium precision. We observed a moderate amount of roughly 50 integrand evaluations per integral that seldomly rises to a few hundreds. In addition, the integral computation is used to check for violations of the constraints (9) in a piggyback fashion. During each integrand evaluation the constrained values are checked against their limits and an error is signaled as soon as a violation is detected. Adaptive quadrature will accumulate more evaluations at spiky parts of the integrands. This will help to detect more precisely if a spike only comes close to or actually crosses the limit. Nevertheless, due to limited integration accuracy, the method is imprecise for heavily spiked integrands. However, for reasonably selected limits, this occurs only on trajectories way beyond feasibility and for such "extremely infeasible" trajectories it is sufficient to detect any invalid value at the slopes of the near-singularity instead of precisely locating the extremal value. The only infeasibility that might still stay undetected is a true singularity, corresponding to a 180-degree-turn in zero time, but this shows upfront in the control points of the curve being all aligned but not in successive order.

4.3 Design of the Control Polygon

By recalling that $\dot{\boldsymbol{p}}^{\mathrm{T}} \ddot{\boldsymbol{p}} \equiv 0$ holds for constant-velocity trajectories, $\xi_{\boldsymbol{x}}$ and $\xi_{\boldsymbol{u}}$ are invertible and thereby every state and input combination can be uniquely converted back into the corresponding flat output and its derivatives:

$$\begin{aligned} \boldsymbol{p} &= (x, y, z)^{\mathrm{T}} \\ \dot{\boldsymbol{p}} &= V\left(\cos\gamma\cos\chi, \cos\gamma\sin\chi, -\sin\chi\right)^{\mathrm{T}} \\ \ddot{\boldsymbol{p}} &= \left(-\dot{y}\dot{\chi} - \frac{\dot{x}\dot{z}\ddot{z}}{\dot{x}^2+\dot{y}^2}, \dot{x}\dot{\chi} - \frac{\dot{y}\dot{z}\ddot{z}}{\dot{x}^2+\dot{y}^2}, -\dot{\gamma}\sqrt{V^2-\dot{z}^2}\right)^{\mathrm{T}} \end{aligned} \tag{11}$$

The state and input continuity constraint from section 4.2 transforms into continuity at transitions between adjacent trajectory segments up to the second derivative. In other words, for every curve, there are constraints $\boldsymbol{p}_0, \dot{\boldsymbol{p}}_0, \ddot{\boldsymbol{p}}_0$ at $t = 0$ and $\boldsymbol{p}_{\mathrm{f}}, \dot{\boldsymbol{p}}_{\mathrm{f}}, \ddot{\boldsymbol{p}}_{\mathrm{f}}$ at $t = t_{\mathrm{f}} = L/V$. We can expressing them in terms of the yet unkown control points $\boldsymbol{c}_i$ to solve for their positions.

Obviously, the first control point $\boldsymbol{c}_0$ equals $\boldsymbol{p}_0$. The second one has to be aligned with $\dot{\boldsymbol{p}}_0$ but we can freely choose its distance l_1 from the starting point:

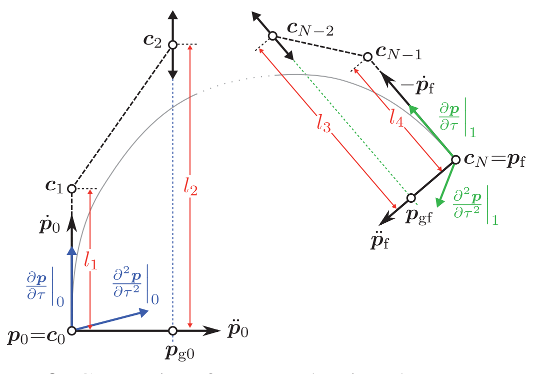
$$\boldsymbol{c}_1 = \boldsymbol{c}_0 + l_1 \frac{\dot{\boldsymbol{p}}_0}{V} \tag{12}$$

To pin down the next control point, we note that

$$\begin{aligned} \ddot{\boldsymbol{p}}_0 &= \left.\frac{\partial^2 \boldsymbol{p}}{\partial \tau^2}\right|_{\tau=0} \cdot \dot{\tau}^2(0) + \left.\frac{\partial \boldsymbol{p}}{\partial \tau}\right|_{\tau=0} \cdot \ddot{\tau}(0) \\ &= N(N-1)\dot{\tau}^2(\boldsymbol{c}_2 - 2\boldsymbol{c}_1 + \boldsymbol{c}_0) + N\ddot{\tau}(\boldsymbol{c}_1 - \boldsymbol{c}_0) \end{aligned} \tag{13}$$

is a linear combination of $\frac{\partial^2 \boldsymbol{p}}{\partial \tau^2}$ and $\frac{\partial \boldsymbol{p}}{\partial \tau}$, which unlike $\dot{\boldsymbol{p}}$ and $\ddot{\boldsymbol{p}}$ are in general not perpendicular. Since $\frac{\partial \boldsymbol{p}}{\partial \tau}|_0$ is aligned with $\dot{\boldsymbol{p}}_0$, only $\frac{\partial^2 \boldsymbol{p}}{\partial \tau^2}|_0$ can deliver the directional component required for $\ddot{\boldsymbol{p}}_0$ by placing $\boldsymbol{c}_2$ at an appropriate location. This choice is limited to a line parallel to $\dot{\boldsymbol{p}}_0$ at a fixed distance from $\boldsymbol{c}_0$:

$$\boldsymbol{c}_2 = \frac{|\ddot{\boldsymbol{p}}_0|}{N(N-1)\dot{\tau}^2} + \frac{l_2}{V}\dot{\boldsymbol{p}}_0 = \boldsymbol{p}_{\mathrm{g}0} + \frac{l_2}{V}\dot{\boldsymbol{p}}_0 \tag{14}$$



$\boldsymbol{c}_2$
$\boldsymbol{c}_{N-2}$
$\boldsymbol{c}_{N-1}$
$-\dot{\boldsymbol{p}}_{\mathrm{f}}$
$\frac{\partial \boldsymbol{p}}{\partial \tau}\Big|_1$
l_3
l_4
$\boldsymbol{c}_N{=}\boldsymbol{p}_{\mathrm{f}}$
$\boldsymbol{c}_1$
l_2
$\frac{\partial^2 \boldsymbol{p}}{\partial \tau^2}\Big|_1$
$\boldsymbol{p}_{\mathrm{gf}}$
$\ddot{\boldsymbol{p}}_{\mathrm{f}}$
$\dot{\boldsymbol{p}}_0$
l_1
$\frac{\partial \boldsymbol{p}}{\partial \tau}\Big|_0$
$\frac{\partial^2 \boldsymbol{p}}{\partial \tau^2}\Big|_0$
$\boldsymbol{p}_0{=}\boldsymbol{c}_0$
$\boldsymbol{p}_{\mathrm{g}0}$
$\ddot{\boldsymbol{p}}_0$

Figure 3: Constraints for control point placement

The situation is depicted in **Figure 3**. The control points $\boldsymbol{c}_N$, $\boldsymbol{c}_{N-1}$ and $\boldsymbol{c}_{N-2}$ at the other end of the curve can be found by a similar computation from the constraints $\boldsymbol{p}_\mathrm{f}$, $\dot{\boldsymbol{p}}_\mathrm{f}$ and $\ddot{\boldsymbol{p}}_\mathrm{f}$. Overall, when not adding further control points in between, there are four undetermined parameters $l_1 ... l_4$. In the next section, we will give geometrically motivated heuristics for their selection in the context of RRT* extension and rewire operations.

4.3.1 Parameter Selection for Rewire Planning

During a rewire attempt (Algorithm 1, line 12) the aforementioned constraints for control point placement arise from the nodes x_new and x_near. Hence, the minimum required curve order is $N = 5$. [1]

The lenghts l_1 and l_4 are selected equal for symmetry:

$$l_1 = l_4 = \min\left(\frac{|\boldsymbol{p}_\mathrm{f} - \boldsymbol{p}_0|}{N}, \frac{V^2}{\mathrm{g}} \cdot \frac{1}{\tan(\mu_\mathrm{max})}\right) \tag{15}$$

The first term is an optimal choice for straight line Bézier curves because it results in a constant (i. e. free of approximation errors) reparameterization $\tau(t)$. The choice becomes less good, the more the curve is bended. However rewire attempts requiring too heavily bended curves are not expected to succeed anyway. The upper limit corresponds to the radius of a maximally banked coordinated turn. The approach for selecting the remaining control points $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$ and thus the parameters l_2 and l_3 is illustrated in **Figure 4**. To get a fair curve, we position the point $\boldsymbol{c}_2$ at the intersection of its constraining line (blue) with a plane exactly centered between $\boldsymbol{c}_1$ and $\boldsymbol{c}_3$. Likewise, $\boldsymbol{c}_3$ should be located at the intersection of the green constraint line with a second plane centered between $\boldsymbol{c}_2$ and $\boldsymbol{c}_4$. By this construction, each control point is defined in terms of the other, so we have to solve for both of them simultaneously.

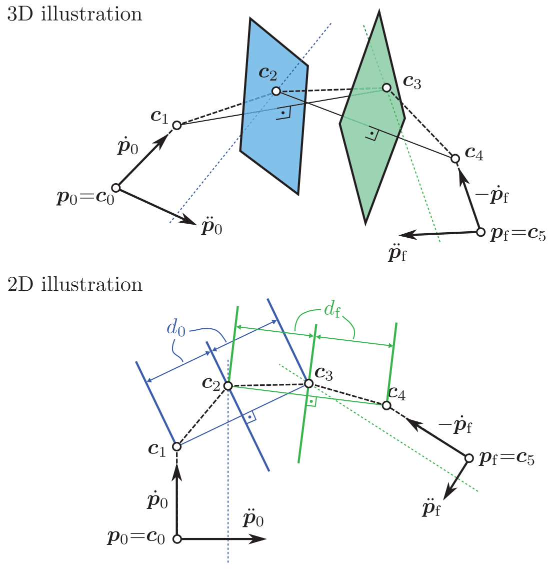

3D illustration
$\boldsymbol{c}_2$
$\boldsymbol{c}_3$
$\boldsymbol{c}_1$
$\dot{\boldsymbol{p}}_0$
$\boldsymbol{c}_4$
$\boldsymbol{p}_0{=}\boldsymbol{c}_0$
$-\dot{\boldsymbol{p}}_\mathrm{f}$
$\ddot{\boldsymbol{p}}_0$
$\boldsymbol{p}_\mathrm{f}{=}\boldsymbol{c}_5$
$\ddot{\boldsymbol{p}}_\mathrm{f}$
2D illustration
d_f
d_0
$\boldsymbol{c}_2$
$\boldsymbol{c}_3$
$\boldsymbol{c}_4$
$-\dot{\boldsymbol{p}}_\mathrm{f}$
$\boldsymbol{c}_1$
$\boldsymbol{p}_\mathrm{f}{=}\boldsymbol{c}_5$
$\dot{\boldsymbol{p}}_0$
$\ddot{\boldsymbol{p}}_0$
$\ddot{\boldsymbol{p}}_\mathrm{f}$
$\boldsymbol{p}_0{=}\boldsymbol{c}_0$

Figure 4: Geometric construction for selecting the innermost control points $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$.

Putting this idea into math, we get

$$\begin{aligned}\boldsymbol{c}_1^\mathrm{T}\boldsymbol{c}_1 &= \boldsymbol{c}_3^\mathrm{T}\boldsymbol{c}_3 + 2\boldsymbol{c}_2^\mathrm{T}(\boldsymbol{c}_1 - \boldsymbol{c}_3)\\ \boldsymbol{c}_4^\mathrm{T}\boldsymbol{c}_4 &= \boldsymbol{c}_2^\mathrm{T}\boldsymbol{c}_2 + 2\boldsymbol{c}_3^\mathrm{T}(\boldsymbol{c}_4 - \boldsymbol{c}_2)\end{aligned} \tag{16}$$

with $\boldsymbol{c}_2 = \boldsymbol{p}_{\mathrm{g}0} + t_0\dot{\boldsymbol{p}}_0$ and with $\boldsymbol{c}_3 = \boldsymbol{p}_\mathrm{gf} + t_\mathrm{f}\dot{\boldsymbol{p}}_\mathrm{f}$. Reordering for the unknowns yield a system of the form

$$A_1 = \quad B_1t_0 + Ct_0t_\mathrm{f} + D_1t_\mathrm{f} + Et_\mathrm{f}^2 \tag{17}$$

$$A_2 = Et_0^2 + B_2t_0 + Ct_0t_\mathrm{f} + D_2t_\mathrm{f}\,. \tag{18}$$

Its solution starts by subtracting (18) from (17). The result is quadratic in t_0. After inserting the roots into (17), we get a quartic in t_f that can be solved either exactly or via Newton iteration. From the set of up to four real-valued roots, we select the smallest positive solution. Any negative t_f would place $\boldsymbol{c}_3$ and $\boldsymbol{c}_4$ at opposite sides of $\ddot{\boldsymbol{p}}_\mathrm{f}$ and generate a heavily bended curve. Larger positive solutions – if they exist – would instead lead to unnecessarily bulgy connections between RRT* nodes. Although we have not noticed the situation in our experiments, a rewire attempt is rejected if there is no positive solution.

4.3.2 Parameter Selection for Extension Planning

For tree extension (Algorithm 1, line 5), there are only initial value constraints $\boldsymbol{p}_0$, $\dot{\boldsymbol{p}}_0$ and $\ddot{\boldsymbol{p}}_0$ from the node x_nearest for positioning control points $\boldsymbol{c}_0$ - $\boldsymbol{c}_2$, but no constraints at the far end. Instead, it is sufficient for the curve to just "bend" into the direction dictated by the random sample x_rand by placing one additional control point $\boldsymbol{c}_3$ at an appropriate location. Thus, a reasonable curve order for extension planning is $N = 3$. However, for the RRT* to quickly find an initial solution to its original trajectory planning problem, it is important that tree extension does not fail too often due to violations of the constraints (9). We therefore have to select $\boldsymbol{c}_3$ carefully to reduce the risk of constraint violation as much as possible.

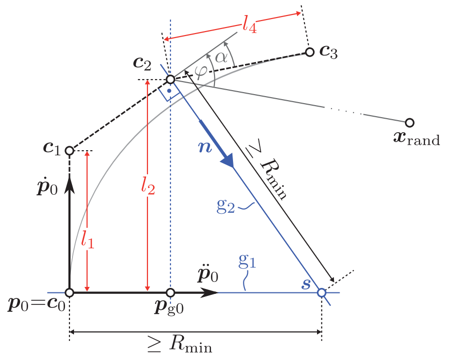


Figure 5: Definitions for control point placement for extension planning

The definitions for parameter selection are set up in **Figure 5**. Lengths l_1 and l_4 are again selected equal

$$l_1 = l_4 = \frac{\min(|\boldsymbol{p}_0 - \boldsymbol{x}_\mathrm{rand}|\,, d_\mathrm{max})}{N} \tag{19}$$

where d_max is an implementation defined constant (we

[1] Note that in theory it is possible to fulfill all constraints with an order-4 Bézier curve. However we found that this often leads to nonsatisfactorily bended curves and for $(\dot{\boldsymbol{p}}_0 \times \ddot{\boldsymbol{p}}_0) \parallel (\dot{\boldsymbol{p}}_\mathrm{f} \times \ddot{\boldsymbol{p}}_\mathrm{f}) \wedge (\boldsymbol{p}_0 - \boldsymbol{p}_\mathrm{f})^\mathrm{T}(\dot{\boldsymbol{p}}_0 \times \ddot{\boldsymbol{p}}_0) \neq 0$ it is not possible at all.

set $d_{\max} = V$ in our implementation). The location of

$$c_2 = p_{g0} + \tilde{t}\,\tilde{\ddot{p}}_0 = c_0 + \underbrace{\left(\frac{|\ddot{p}_0|}{N(N-1)\dot{\tau}_0^2}\right)}_{\Delta_g}\tilde{\ddot{p}}_0 + \tilde{t}\,\tilde{\ddot{p}}_0 \quad (20)$$

(with $\tilde{a} = a/|a|$ for all vectors a) is chosen to move the intersection s at least $R_{\min} = V^2 \cdot (\mathrm{g} \tan(\mu_{\max}))^{-1}$ away from both c_0 and c_2. When writing the two blue lines as

$$\mathrm{g}_1\colon p = c_0 + d_1\tilde{\ddot{p}}_0 \qquad \mathrm{g}_2\colon p = c_2 + d_2 n \quad (21)$$

this translates into $d_1 \geq R_{\min}$ and $d_2|n| \geq R_{\min}$. Solving for the critical points results in a quadratic in $\tilde{t}$

$$\tilde{t}^2 - l_1\tilde{t} + \Delta_g(\Delta_g - R_{\min}) = 0 \quad (22)$$

for the first condition and in a 4th order polynomial for the second condition:

$$\tilde{t}^4 - 2l_1\tilde{t}^3 + (l_1^2 + \Delta_g^2)\tilde{t}^2 - R_{\min}^2\Delta_g^2 = 0 \quad (23)$$

We select $\tilde{t}$ greater or equal to the first positive solution of both (22) and (23) by following the argument from section 4.3.1. If no such solution exists, the extension attempt fails. The remaining control point c_3 will ideally head directly towards the random sample x_{rand}. An upper limit is imposed on the angle deviation α to prevent too much bending and thus infeasible steering commands:

$$\alpha = \min(\varphi, \varphi_{\max}) \qquad (\text{here: } \varphi_{\max} = l_4/V) \quad (24)$$

Note that angles are measured in 3D space. The situation depicted in Figure 5 is not necessarily planar. As a final adjustment, the vector $(c_3 - c_2)$ is made consistent with the γ envelope from (9).

5 Simulation Experiments

5.1 Implementation Details

For efficiency reasons, the RRT* algorithm has been implemented in C++ and MATLAB is used as a visualization frontend. The implementation uses a polygonal obstacle model. We rely on CGAL AABB trees [14] for collision detection. The checks are carried out by discretizing each trajectory segment into straight line segments and testing each segment against the obstacle polygons. The discretization is done by an equidistant subdivision of the τ unit interval. The number of subdivisions is decided from a length approximation of the curve and a configurable precision parameter. This way, collision detection can happen at a very early stage before time reparameterization and associated numerical computations. From the description it is obvious that obstacles apply to the aircraft's center of mass. Thus, in the pictures shown in this section, they should be interpreted as obstacles to the trajectory and not to the aircraft's geometry, which is shown for illustrational purposes only. In practice, a suitable model could be created by inflating the real obstacle geometry with the extent of the vehicle. One could also add a safety margin, if desired.

Several common RRT* optimizations have been implemented. Since the trajectory design process in 4.3.2 does not steer to a specific orientation, the sample space is reduced to (x, y, z) coordinates (sometimes called "task space sampling"). Until an initial solution has been found, we skip the time-consuming rewiring process and also use a 35 % goal bias. Afterwards, sampling is biased in the vicinity of existing tree branches instead, to improve the convergence rate. This however might delay finding better solutions from different homotopy classes. Periodic pruning, i. e. removal of any branch of which accumulated costs exceed the current best solution, is triggered every 200 iterations after an initial solution emerged. This optimization again may have negative consequences, as it might remove branches that may contribute to a better solution in the future. Though in our experiments neither influence of prunning was hardly noticeable.

Due to differential constraints imposed by (1) and our cost definition (10) the euclidian distance is not a good metric in lines 4 and 10 of Algorithm 1. For a better approximation, we follow the idea of reachability presented in [15] and consider any pair of nodes infinitely far apart from each other if the position of one of them is not inside a funnel originating from the other. In **Figure 6** our funnel shape is compared to the numerically computed true reachable set.

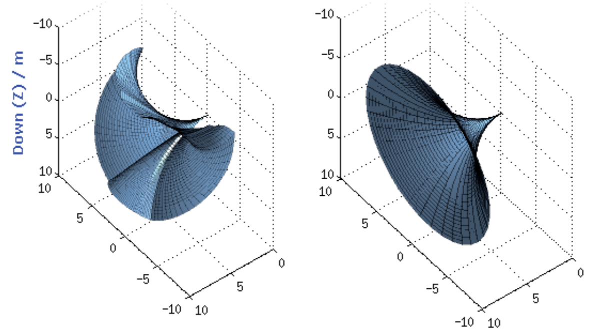
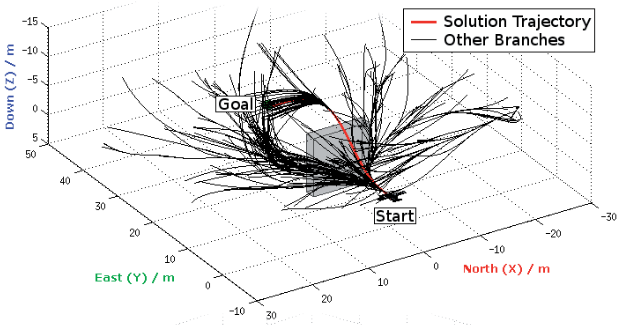


Figure 6: True reachable set (left) at level flight with $V = 8\,\mathrm{m/s}$ and funnel approximation (right)



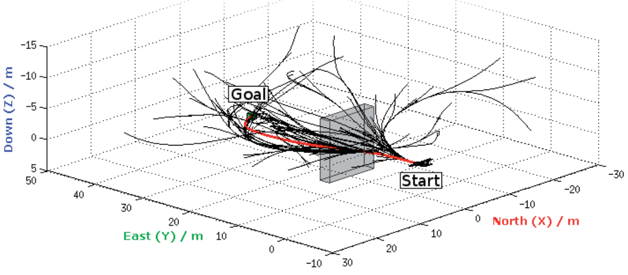


Figure 7: Planning from $(0, 0, 0)$ to $(0, 35, 0)$ at $V = 8\,\mathrm{m/s}$ with cost weights $[w_\mu, w_n, w_L] = [1, 0, 0]$ (top) and $[w_\mu, w_n, w_L] = [0, 1, 0]$ (bottom).

5.2 Testing the Local Planner

To validate basic operability of the local planner and our RRT* implementation, we used the simple test scenario from **Figure 7**. By varying the weights in (10), planning results can be biased in favor for more banking or more climb. Experiments for two extreme cases are shown in the figure. The resulting solution trajectories (red) are well in line with expectations, which indicates correct behavior.

5.3 RRT* Simulations

We now present planning results for the three scenarios portrayed in **Figure 8**. The first one is a fictional indoor environment of an atrium with some obstacles (stairs, skybridge), surrounded by hallways. The second scenario represents an outdoor environment. It is made from an elevation bitmap. The last scenario is a stress test to the planner. As RRTs usually struggle with narrow passages, we constructed an environment that requires passing two of them in order to reach the goal.

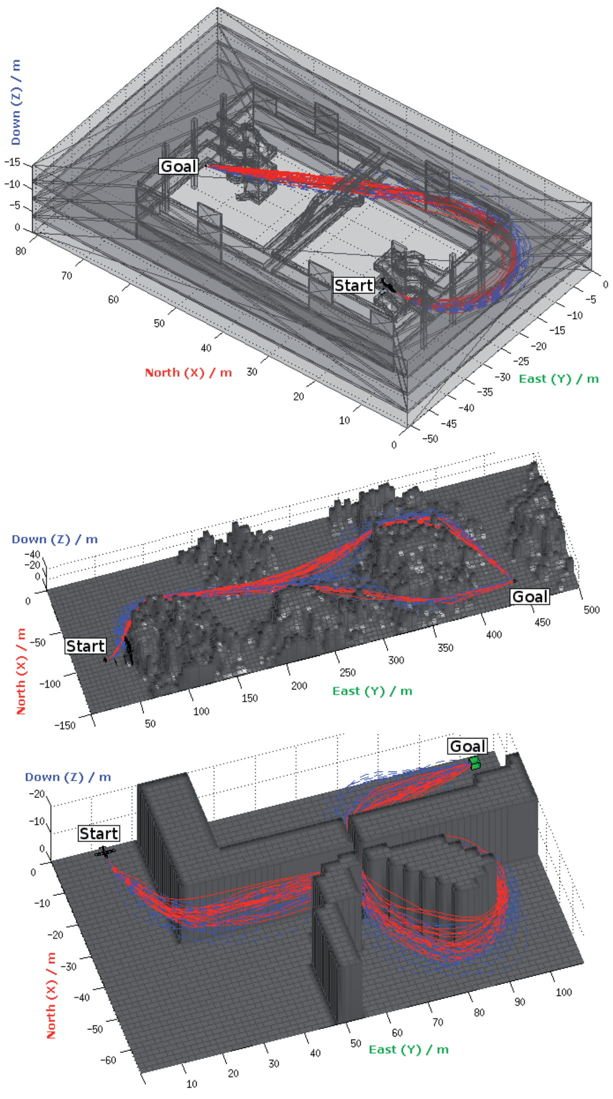


Figure 8: Test scenarios *Indoor*, *Outdoor* and *Narrow Passages* (top to bottom) with solution trajectories after 5000 iterations (red) and initial solutions (blue)

Table 1: Summary of experimental results

	Initial Solution		After 5000 It.	
Scenario	runtime mean std.	length mean std.	runtime mean std.	length mean std.
Indoor	0.11 s	93.2 m	127 s	88.1 m
	0.08 s	2.04 m	14.4 s	1.78 m
Outdoor	1.15 s	499 m	27.2 s	471 m
	0.53 s	20.9 m	3.71 s	17.2 m
Narrow	3.99 s	228 m	69.8 s	212 m
Passages	1.95 s	5.87 m	9.75 s	2.36 m

All experiments were carried out on a single core of a 3 GHz Core2 Processor, released in 2008. The processing power is comparable to todays high-end mobile embedded hardware that could be integrated into a small (but not tiny) model-grade aircraft, especially when performing multicore optimization. The planner was invoked 25 times per scenario with the parameters set to $V = 8\,{}^{\mathrm{m}}/_{\mathrm{s}}$, $\gamma_{\mathrm{min}} = -45\,°$, $\gamma_{\mathrm{max}} = 45\,°$, $n_{\mathrm{min}} = 0$, $n_{\mathrm{max}} = 2$ and $\mu_{\mathrm{max}} = 45\,°$. Each attempt was stopped after 5000 iterations. In contrast to section 5.2, we considered path length as our only cost factor, because it facilitates assessment of trajectory quality from looking at the 3D plots.

A summary of the results is given in **Table 1**. All attempts succeeded in finding a solution trajectory after 5000 iterations. These are overlayed in Figure 8 (red). For comparison, we added initial solutions as blue traces. They do not differ too much from the final results because the differential constraints in our model prevent too much zigzag, but the extra iterations always yield some reduction in length.

The evolution of path length vs. runtime is displayed in **Figure 9**. In general, we recognize converging behavior. However, the outliers in the first two scenarios indicate that most attempts do not converge to a minimum cost solution. We note that all outliers belong to homotopy classes different from the majority of solutions. Convergence to a better solution thus would require homotopy class switching, which is unlikely because our sampling bias during the rewire phase favors local optimization.

The runtimes required for finding an initial solution as well as the duration for 5000 iterations are marked with squares in the plots. Apparently, initial solutions are found very reliably and almost instantly for the indoor scenario. As expected, there is far more variance in the stress test scenario, where it usually took about 2000 out of the 5000 iterations. We also marked the points of break-even, i. e. the situations where computation time exactly matches the time for flying the trajectory. For all scenarios and attempts, there is a safe distance from finding the initial solution. This qualifies our planning approach for real-time application, since a solution trajectory is computed faster then it could be flown. The second plot lacks appropriate markers because 5000 iterations always completed before break-even happens for this rather large scenario.

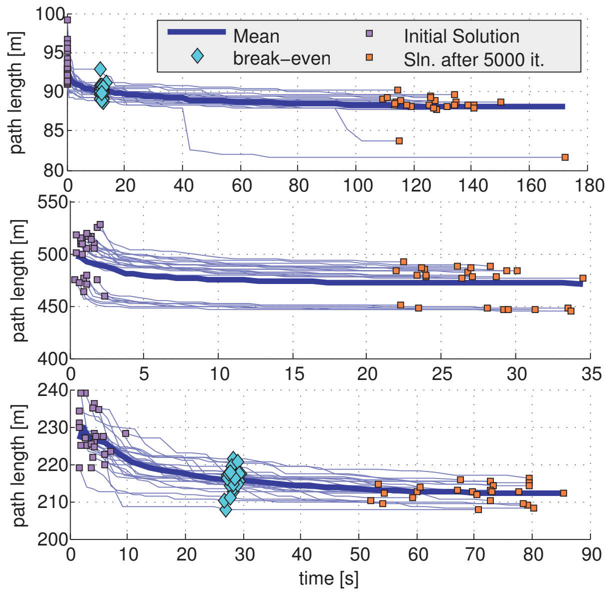


Figure 9: Path length vs. runtime recorded for 25 passes per scenario. From top to bottom: Indoor, Outdoor, Narrow Passages.

6 Conclusion and Further Work

We presented a trajectory planning approach for fixed wing aircraft by combining the RRT* global optimization framework and a local trajectory planning technique based on Bézier curves that relies on differential flatness of the model equations. A mass-point approximization for aircraft motion was utilized, which captures basic differential constraints without the need for vehicle-specific parameters. Several simulation experiments yield satisfactory planning results, both in terms of the identified trajectory and the required computation time. A solution was found in all of our experiments in less time than required to fly the corresponding trajectory, which justifies our claim of real-time performance. The computation times are also roughly in line with the figures reported by [9], which is in some aspects similar to our approach and implementation.

Unfortunately, we had to implement many algorithm details in an application-specific way and mostly driven by heuristics to get the performance right (e. g. Bézier control point placement, distance metric approximation). This makes it difficult to transfer results to other system models and – as we have noted for the convergence behavior of the indoor and outdoor simulation experiments – some implementation decisions we made seem to interfere with RRT*'s original convergence guarantee. From our understanding of the reasoning in [2] there is no argument that prevents convergence of our approach, yet there is no formal convergence proof either. (Indeed, the formal convergence proof does not apply to any local planning strategy except for straight line connections.)

In future research, we will investigate two major modifications of our approach. The first is a local optimizer to improve a solution trajectory within its homotopy class more efficiently than by RRT* rewiring. This way, we hope to drop the current sampling bias during the rewire phase that seems to cause convergence issues. The second modification will be an attempt to replace the indirect sampling of a node and an extension direction in lines 3 and 4 of Algorithm 1 with a direct sampling strategy that takes into account local node and obstacle density. From this modification we expect better performance in complicated scenarios like the "Narrow Passages" experiment and the currently rather strong goal bias could be replaced by something more aware of the local obstacle situation. Finally it would be nice to combine the approach with an actual tracking controller and test it in real flight experiments.

Acknowledgement

The authors would like to thank Deutsche Forschungsgemeinschaft (DFG) for funding this research under contract # JA1559/2-1.

References

[1] S. M. LaValle: *Rapidly-exploring random trees: A new tool for path planning*, TR 98-11, Computer Science Dept., Iowa State University, 1998.
[2] S. Karaman, E. Frazzolli: *Incremental Sampling-based Algorithms for Optimal Motion Planning*, in Robotics: Science and Systems (RSS), Zaragoza, Spain, 2010.
[3] J. H. Jeon, S. Karaman, E. Frazzolli *Anytime Computation of Time-Optimal Off-Road Vehicle Maneuvers using the RRT**, in IEEE Conference on Decision and Control (CDC), 2011.
[4] R. D. Russel, L. Shampine *A collocation method for boundary value problems*, Numerische Mathematik, vol. 19, no. 1, pp 1-28, 1972.
[5] S. Karaman and E. Frazzolli: *Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods*, in IEEE Conference on Decision and Control (CDC), Atlanta, GA, 2010.
[6] A. Perez, et. al.: *LQR-RRT*: Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics*, ICRA, 2012.
[7] D. J. Webb and J. v.d. Berg: *Kinodynamic RRT*: Asymptotically Optimal Motion Planning for Robots with Linear Dynamics*, `http://arxiv.org/abs/1205.5088`, 2012.
[8] T. Bolemann: *RRT*-basierte Bahnplanung mit Erweiterung auf Spline-Bahnen*, Student Research Thesis, University of Stuttgart, 2011.
[9] P. Chaudhari *Aggressive maneuvers using differential flatness for a quadrotor*, `http://www.mit.edu/~pratikac/pub/chaudhari.aggressive.maneuver.pdf`
[10] W. Fichter, W. Grimm: *Flugmechanik*, Shaker Verlag, ISBN 978-3-8322-8482-4, 2009.
[11] M. Fliess, J. Lévine, P. Martin, P. Rouchon *Flatness and Defect of Nonlinear Systems: Introductory Theory and Examples*, CAS internal report A-284, 1994.
[12] M. Walter, A. Fournier *Approximate Arc Length Parametrization*, in Proceedings IX Brazilian Symposium of Computer Graphics and Image Processing, 1996.
[13] H. Wang, J. Kearney, K. Atkinson *Arc-Length Parameterized Spline Curves for Real-Time Simulation*, in Curve and Surface Design, pp. 387–396, Saint-Malo, 2002.
[14] P. Alliez, S. Tayeb, C. Wormser *3D Fast Intersection and Distance Computation (AABB Tree)*, in CGAL v4.3 User and Reference Manual, 2013.
[15] A. Shkolnik, M. Walter, R. Tedrake *Reachability-Guided Sampling for Planning Under Differential Constraints*, in IEEE International Conference on Robotics and Automation (ICRA), Japan, 2009.