Aggressive maneuvers using differential flatness for a quadrotor

Pratik Chaudhari

Abstract— **Optimal control strategies for large nonlinear systems can be constructed easily using sampling based algorithms. The applications are usually limited due to the computational complexity of obtaining optimal connections for general nonlinear systems. This work uses differential flatness to propose an optimal / feedback control algorithm for any flat system using an asymptotically optimal algorithm based on Rapidly Expanding Random Trees (RRT*). An integrated solution approach for planning and control problems of differentially flat systems is described. A quadrotor is used to demonstrate the approach to obtain time-optimal trajectories with bounded controls.**

I. Introduction

Quadrotors are exciting testbeds for problems in feedback control, stochastic control and learning. Indeed numerous works, [1]–[3] being the most notable among them have shown incredible results related to motion planning and learning trajectories in the precence of disturbances. The motivation for this project comes from the above references. In particular, I would like to plan fast trajectories for quadrotors. Usual optimal-control techniques cannot handle the large state-space and the generality of the problem that we are dealing with. Also, optimal control is usually a one-shot approach in the sense that it generates one optimal trajectory and cannot modify it in the event of tracking errors.

Sampling-based algorithms are popular in motion-planning because (i) they are fast and able to randomly sample huge state-spaces (ii) anytime solution i.e. a sound solution is returned very quickly and the algorithm improves upon it to make it better if given more computational time (iii) incremental in nature, i.e. the previously obtained solution is used in calculating the new solution. The basic idea of these algorithms is to randomly sample a set of states and draw connections between them to come up with trajectories from an initial root state to the goal state. In particular, the RRT* algorithm provides asymptotical optimality guarantees for trajectories while giving probabilistic completeness i.e. the probability that the algorithm returns an optimal solution converges to one as the number of samples approaches infinity [4]. Single integrators [4], double integrators [5], Dubins' vehicle dynamics [5], robotic manipulator, and the single-track vehicle dynamics [6] are some of the systems that have been investigated successfully using the RRT* algorithm.

Pratik C. is with the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA {pratikac}@mit.edu

One strong requirement for the RRT* is the *steering function* that can steer a state to another exactly. Not surprisingly, this requirement makes the applications of the RRT* on dynamical systems non-trivial tasks. Most of the examples dealt with in literature have analytical solutions, indeed a general solution involves solving a nonlinear optimization problem for every two states in the graph. However, we can leverage the property of *differential flatness* [7], to generate steering functions easily. We can search in a fairly general class of basis functions e.g. B-splines and get optimal connections within this class. This makes RRT* which is a very general algorithm for optimal control easily applicable to a large class of nonlinear systems. Apparent advantages over other optimal control methods like variational and collocation methods include (i) very high dimensional state-spaces (ii) computational efficiency, and (iii) ability to state and control constraints.

Note that, even with a feasible trajectory, the tracking controller might not be able to track it properly due to disturbances in input and uncertain model parameters. Differential flatness enables us to trivially extend tracking controllers by giving a nominal value of the control input to be applied [8], the error over this can be used to devise a PD trajectory controller. We can however do something much better because of sampling based algorithms. This is discussed in Section VI.

This paper applies the RRT* algorithm to differentially flat systems and demonstrates an integrated solution approach for planning and control of flat systems. It uses a quadrotor as an example system to show that a large state-space can be handled quite easily to obtain aggressive time-optimal maneuvers.

This paper is organized as follows. Section II starts with the problem statement. Section III introduces the RRT* algorithm, and Section IV describes differentially flat systems. In Section VI, a motion planning algorithm for flat systems and an integrated approach for planning and control of flat systems are discussed. Section VII shows the applications on a quadrotor for a variety of maneuvers including nominal aggressive trajectories and flying through obstacles.

II. Problem Statement

Let $X \subset \mathbb{R}^n$ and $U \subset \mathbb{R}^m$ be compact sets that account for state and input constraints. Let $z_0 \in X$ be an initial state, and let f define the time-invariant dynamical system

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(0) = z_0 \tag{1}$$

where $x(t) \in X$ and $u(t) \in U$. A trajectory $x : [0, T] \to X$ is said to be a *dynamically feasible* if there exists $u : [0, T] \to U$ such that x and u satisfy Equation (1) for all $t \in [0, T]$.

Let $X_{\text{goal}}, X_{\text{obs}} \subset X$ be open sets, and define $X_{\text{free}} = X \setminus X_{\text{obs}}$. The sets X_{goal}, X_{obs}, and X_{free} are called the *goal set*, *obstacle set*, the *obstacle-free set*, respectively. A trajectory $x : [0, T] \to X$ is called *obstacle free* if $x(t) \in X_{\text{free}}$ for all $t \in [0, T]$.

Then, the problem is to find a dynamically-feasible obstacle-free trajectory $x : [0, T] \to X$ that reaches the goal region, i.e., $x(T) \in X_{\text{goal}}$, while minimizing some cost function $J(x)$. We indend to find time-optimal solutions.

III. RRT* Algorithm

The RRT* algorithm was first introduced in [4], and then extended to more complex dynamical systems in [5], [6]. The RRT* algorithm as presented in [5], [6] is provided next with some refinement and modification. We start by noting some primitive procedures that RRT* relies on.

`Sample`: The sampling procedure returns independent and identically distributed samples from the obstacle-free space. For simplicity, we assume that the sampling distribution is uniform, even though our results hold for a large class of sampling strategies.

`NearVertices`: Given a graph $G = (V, E)$ on X_{free}, a state $z \in X$, this returns a set $\{z'\}$ such that $||z - z'||_2 < \gamma(\log(n)/n)^{1/d}$ where $n = |V|$ and d is the dimension of the sampling space. $\gamma > \gamma^*$ according to [9].

`Steer`: Given two states $z_1, z_2 \in X$, this returns the optimal trajectory starting at z_1 and ending at z_2. This `Steer` procedure will be specified later in Section VI after the introduction to the flatness in Section IV.

`Extend`: Calling this procedure for a state $z \in V$ creates a new graph after including edges from calls of the `Steer` procedure on all neighbors of z given by `NearVertices`.

`CheckTrajectory`: Given a trajectory $x : [0, t] \to X$, this procedure returns true iff x lies entirely in the obstacle-free set, i.e., $x(s) \in X_{\text{free}}$ holds for all $s \in [0, t]$ and $u(s)$ lies within some pre-defined bounded set. Thus, this procedure checks for dynamically feasible and obstacle free trajectories.

`UpdateCost`(z_1, z_2): If $J(z_2) > J(z_1) + J(z_1, z_2)$, then this sets $J(z_2)$ as $J(z_1) + J(z_1, z_2)$ where $J(.)$ is the optimal cost of reaching the vertex z from the root (initialized to ∞) and $J(.,.)$ is the cost of the trajectory connecting the two vertices. We also denote z_1 as the parent of z_2 in such cases. This first calls `CheckTrajectory` to check if the trajectory is valid, else it returns fail.

Algorithm 1: The RRT* Algorithm

```
1 $V \leftarrow \{z_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
2 while $i < N$ do
3     $G \leftarrow (V, E)$;
4     $z_{\text{rand}} \leftarrow$ Sample($i$); $i \leftarrow i + 1$;
5     $(V, E) \leftarrow$ Extend($G, z_{\text{rand}}$);
```

Based on these primitive procedures, the outer loop of the RRT* algorithm is given in Algorithms 1. Initialized with the tree that includes z_{init} as its only vertex and no edges, the algorithm iteratively builds a tree of collision-free trajectories by first sampling a state from the obstacle-free space (Line 4) and then extending the tree towards this sample (Line 5) at each iteration. The `Extend` procedure is not necessarily successful in its execution.

Let us make the `Extend` procedure a bit more precise. Given a random sample z_{rand}, the algorithm attempts to extend near-by vertices toward the sample $z_{\text{new}} = z_{\text{rand}}$, and find the minimum cost connection with no collision. In that process, the vertex z_{near} that can be steered to z_{new} with minimum cost is chosen as the parent of z_{new}. With the minimum cost connection found, we add the new edge to the tree. It then does a sub-procedure called "rewiring". It checks all vertices in $Z_{\text{near}} = \texttt{Near}(z_{\text{new}})$ to see if their existing parents can be changed to z_{new} instead by running `UpdateCost` on those pairs.

The RRT* algorithm guarantees asymptotic optimality, i.e., almost-sure convergence to optimal solutions [4]–[6].

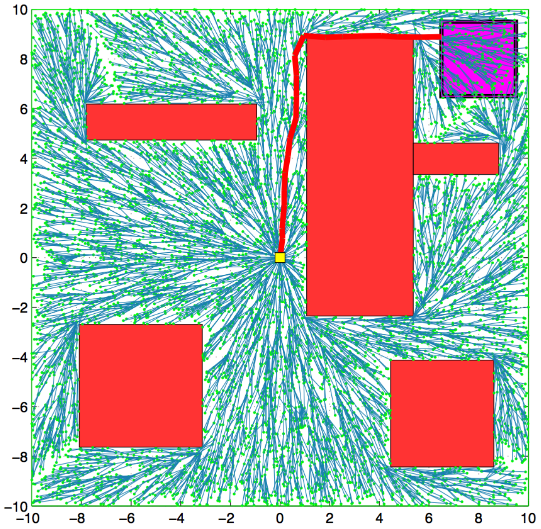


Fig. 1: An example RRT* graph for a 2D single integrator with obstacles. The system starts at the origin and plans a path to move into the goal region colored in purple [9].

IV. Differentially Flat Systems

The differential flatness concept was first introduced by Fliess et al. [7]. In this section, differential flatness is introduced, and a point-to-point steering method for differentially-flat systems is described.

A. *Differential Flatness*

A nonlinear system

$$\dot{x} = f(x,u), \quad x \in \mathbb{R}^n,\ u \in \mathbb{R}^m$$

is differentially flat if there exist *flat outputs*

$$z = \Psi_z(x, u, \dot{u}, \ldots, u^{(l)}), \quad z \in \mathbb{R}^m$$

such that

$$\begin{aligned} x &= \Psi_x(z, \dot{z}, \ldots, z^{(l)}) \\ u &= \Psi_u(z, \dot{z}, \ldots, z^{(l)}) \end{aligned}$$

where $z^{(k)}$ denotes the k-th derivative of z.

Essentially, any trajectories of flat outputs z can be realized since there are known functions Ψ_x and Ψ_u that map z into x and u that result in z. Flatness is also referred to as a Lie-Bäcklund equivalence between a nonlinear system and a trivial system [10]. It is also known that the conditions that result in feedback linearizability and differential flatness are the same.

There are numerous examples of flat systems including kinematic n-trailer, planar VTOL, Gantry crane, rolling penny, and simplified helicopter model. Consider the simple pendulum as an example.

$$\begin{aligned} & ml^2\ddot{\theta} + b\dot{\theta} + mgl\sin\theta = u \\ & z = \theta \\ & \theta = \Psi_\theta(z) = z \\ & u = \Psi_u(z, \dot{z}, \ddot{z}) = ml^2\ddot{z} + b\dot{z} + mgl\sin z \end{aligned}$$

are satisfied. This means that given a trajectory $z(t)$, we can evaluate the states, $\theta(t)$ and $\dot{\theta}(t)$ and the control $u(t)$ uniquely. Generating feedback control from the trajectory-plan is thus easy. It is this property that we leverage.

B. *Point-to-Point Steering*

Using the flatness, local steering from a state to another state becomes very easy. In particular, instead of searching for the globally optimal trajectories, we only search for trajectories which can be represented as polynomials. First, we parameterize each component of the flat outputs z_i, $i = 1, \ldots, m$ by

$$z_i(t) = \sum_j a_{ij}\phi_j(t), \quad \text{for } t \in [t_0, t_f]$$

where $\phi_j(t)$, $j = 1, \ldots, N$ are basis functions (polynomials in this case) and a_{ij} are coefficients. This parameterization relaxes an infinite dimensional problem to a finite dimensional problem. As $N \to \infty$, we can approximate any given trajectory by these basis functions and hence we can get any optimal trajectory as well.

The coefficients a_{ij} are calculated by solving the following system of equations:

$$\begin{array}{cc} z_i(t_0) = \sum_j a_{ij}\phi_j(t_0) & z_i(t_f) = \sum_j a_{ij}\phi_j(t_f) \\ \vdots & \vdots \\ z_i^{(l)}(t_0) = \sum_j a_{ij}\phi_j^{(l)}(t_0) & z_i^{(l)}(t_f) = \sum_j a_{ij}\phi_j^{(l)}(t_f). \end{array}$$

where $z_i^{(k)}(t_0)$ and $z_i^{(k)}(t_f)$ for $k = 1, \ldots, l$ are mapped from the desired states x and u at both ends t_0 and t_f, and the time t_f can be arbitrarily determined.

This system of equations is expressed as follows:

$$\begin{bmatrix} z_i(t_0) \\ \vdots \\ z_i^{(l)}(t_0) \\ z_i(t_f) \\ \vdots \\ z_i^{(l)}(t_f) \end{bmatrix} = \begin{bmatrix} \phi_1(t_0) & \cdots & \phi_N(t_0) \\ \vdots & \ddots & \vdots \\ \phi_1^{(l)}(t_0) & \cdots & \phi_N^{(l)}(t_0) \\ \phi_1(t_f) & \cdots & \phi_N(t_f) \\ \vdots & \ddots & \vdots \\ \phi_1^{(l)}(t_f) & \cdots & \phi_N^{(l)}(t_f) \end{bmatrix} \begin{bmatrix} a_{i1} \\ \vdots \\ a_{iN} \end{bmatrix}. \quad (2)$$

In particular, for $N = 2(l+1)$, the problem simply boils down to a matrix inversion. If $N > 2(l+1)$, the additional degree of freedom can be used to optimally choose constraints [11]. This procedure is decoupled for different flat outputs z_i.

V. Quadrotor is differentially flat

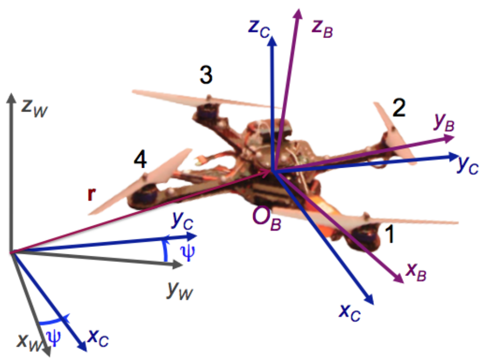


Fig. 2: Body axes and World axes [1]

The states of this system are $[x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]^T$ where ϕ, θ, ψ are the Euler angles, roll, pitch and yaw respectively, for Z–X–Y frame and p, q, r are the rates of rotation in the body frame. Let the velocities of the 4 rotors be ω_i producing forces F_i and moments M_i respectively. Define the control vector $\mathbf{u} = [u_1, u_2, u_3, u_4]^T$. Thus, u_1 is the net body force in the Z-direction in the body frame while the other three are body moments. We have a direct relation between these forces/moments and rotor velocities. Assuming that body dynamics is much slower than the motor dynamics, we can directly work with u_i instead of worrying about ω_is. The equations of motion are,

$$m\ddot{\mathbf{r}} = -mg\mathbf{z}_W + u_1\mathbf{z}_B \quad (3)$$

$$I\dot{\omega}_B + \omega_B \times I\omega_B = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} \quad (4)$$

where $\mathbf{z}_B$ and ω_B are measured in the body frame and $\mathbf{z}_W$ is measured in the world frame.

It turns out that this system is flat if we consider the vector $\sigma = [x, y, z, \psi]^T$ as the flat output. This section will briefly give the formulae for the states and control in terms of these flat outputs. Detailed derivation is given in [1].

The rotation matrix i.e. Euler angles R_B is obtained as follows. Define $\mathbf{t} = [\ddot{x},\ \ddot{y},\ \ddot{z}\ \ + g]^T$ and $\mathbf{x}_C = [\cos\psi,\ \sin\psi,\ 0]^T$. Then,

$$\mathbf{z}_B = \mathbf{t}/||\mathbf{t}||$$
$$\mathbf{y}_B = \frac{\mathbf{z}_B \times \mathbf{x}_C}{||\mathbf{z}_B \times \mathbf{x}_C||}$$
$$\mathbf{x}_B = \mathbf{y}_B \times \mathbf{z}_B$$

with $R_B = [\mathbf{x}_B\ \mathbf{y}_B\ \mathbf{z}_B]$. We can get the Euler angles from this matrix using algebraic manipulations.

Note that $\omega_B = p\ \mathbf{x}_B + q\ \mathbf{y}_B + r\ \mathbf{z}_B$. Expressions for p, q, r are obtained by differentiating Equation (3). Differentiate it once again to obtain expression for $\dot{\omega}_B$. Finally, the control is obtained as $u_1 = m\ ||\mathbf{t}||$ and $[u_2, u_3, u_4]^T$ from Equation (4).

VI. RRT* for Flat Systems

The `Sample` procedure uniformly samples the state space along with some heuristic sampling of the goal region. Note that the state space consists of the flat outpus z, not the original states x. The trajectories connecting two samples however consist of the original state-space.

The `Steer` procedure does point to point steering of the system and uses differential flatness to connect between two states. We would like optimal connectins between any two states. This problem even with point to point steering is hard. I used B-splines as the basis functions. The order of the polynomial is given by $2(l+1)$ and is fixed in the implmentation. The only remaining variable is the duration of the trajectory. For various cost functions, we can optimize over the trajectories but since we are only looking for time-optimal trajectories, I chose to search linearly over the time in small steps to find the the trajectory such that $u(t) < U \quad \forall t \in [0,\ T]$ where U is the upper bound on the actuation power available. For a quadrotor, this translates to the maximum and minimum speeds of rotation of the rotors.

For a classical trajectory planning problem, we can define a PD tracking controller on top of the $u(t)$

obtained from differential flatness trivially. However, in this case when we are using sampling based motion-planning algorithms to generate trajectories, we realise that we are already generating control policies i.e., if due to disturbances, the system veers away from the planned trajectory, the tree in RRT* as shown in Figure 1 already contains a different trajectory that it can follow to reach the goal.

VII. Experiments

A. Implementation details

RRT* works with the flat outputs as given in Section V. The states for sampling are $[\sigma,\ \dot{\sigma},\ \ddot{\sigma}]$ to create 12-dimensional sampling. Ideally, we can have only till the first derivative to get smooth paths. However, the Euler angles are functions of the second derivative of flat outputs. Even though the trajectory of Euler angles between any two samples is smooth, when it transitions from one sample to the other, there is discountinuity in the second derivative. Adding the second derivative as another state solves this problem. The numerical values of the quadrotor were taken from [12].

The equations in Sections IV and V were implemented using the symbolic toolbox in MATLAB. This was then exported to C expressions using the inbuilt `ccode` function. I implemented the RRT* algorithm in C++. This enables a very fast implementation for a large system with 12 states wherein the first solution is obtained within 5 secs for almost all cases. Asymptotically optimal guarantees of the algorithm ensure that we will get the optimal solution if we wait long enough. To hasten the convergence, we can use a number of easy to implement heuristics such as biased sampling and branch and bound in the search algorithm which I also implemented.

B. Nominal paths

An example trajectory obtained within 2 secs for an initial state with an upward velocity of 10 m/s and the final state with a velocity of 10 m/s in the forward direction. This trajectory is made by joining two inter-sample trajectories.

C. Obstacles

Let us evaluate the performance of the planner in the precense of obstacles. Figure 4 is obtained quickly but is clearly sub-optimal with a cost of 11.8 secs. As the planner samples more states, the trajectory changes to the one shown in Figure 5 taking only 5.6 secs. The state trajectories for a similar situation look as shown in Figure 6 and the control trajectories are Figure 7.

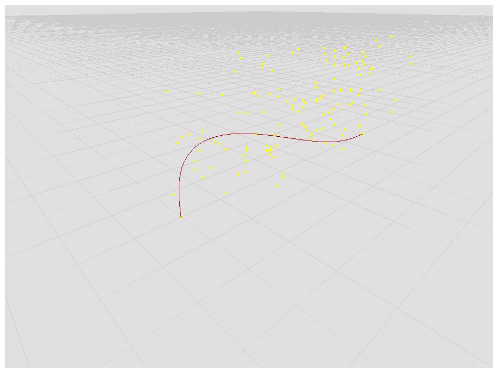


Fig. 3: Nominal trajectory

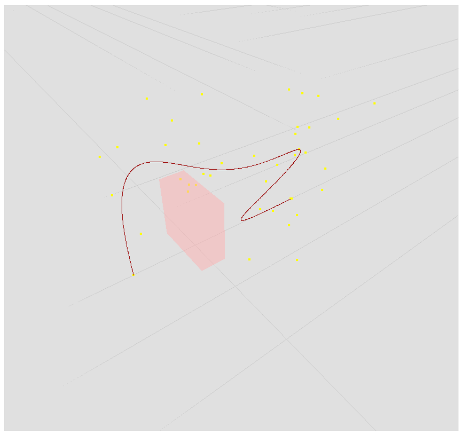


Fig. 4: One obstacle: around the wall

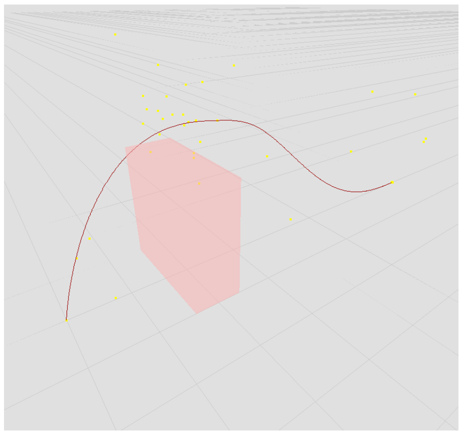
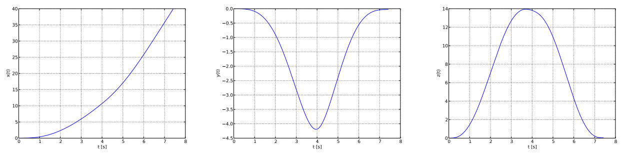
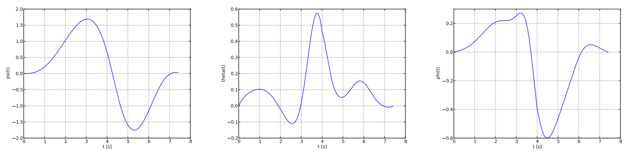


Fig. 5: One obstacle: over the wall

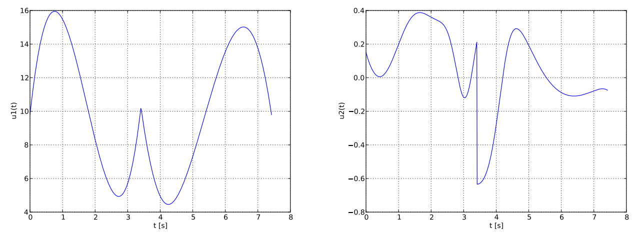


(a) x(t) (b) y(t) (c) z(t)

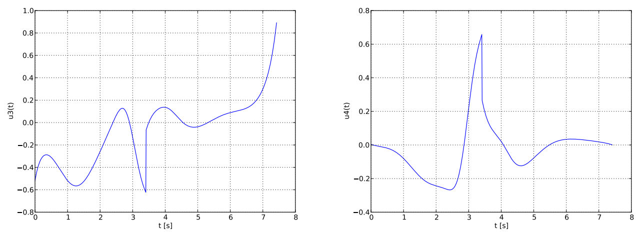


(d) ψ (t) (e) θ (t) (f) ϕ (t)

Fig. 6: State trajectories for one obstacle



(a) u_1(t) (b) u_2(t)



(c) u_3(t) (d) u_4(t)

Fig. 7: Control trajectories for one obstacle

D. Window

Let us introduce multiple obstacles to create a window. The attitude of the quadrotor needs to be checked inorder to see if the state is in collision with the obstacles or not. We thus intend to create a trajectory wherein, the quadrotor needs to roll on it's side inorder to pass through a narrow window. Figure 8 shows such a trajectory given by the planner. Note that, a state-space with many obstacles makes the problem harder and the sampling based algorithm is able to solve it under 10 secs. If we used non-linear optimization, the algorithm would spend all it's time trying to plan trajectories and then drop them when they collide with the obstacles. Differential flatness makes the connections very fast and is thus able to get a lot of samples in short time.

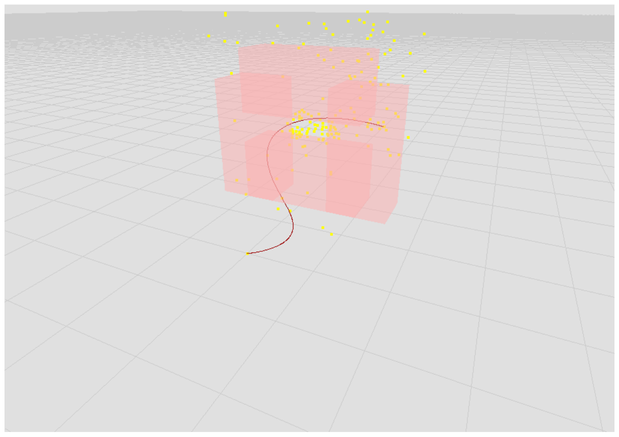


Fig. 8: Flying through a window

E. Flying around a wall

I tried to get a maneuver which forced the quadrotor to roll on it's side inorder to get the time-optimal trajectory. It involves lifting off and cornering around a wall. The velocity and the final state is such that it is optimal to roll on the side and turn. Sub-optimal solutions will include going slightly farther away from the wall and then coming back again with a roll. The actual trajectory is shown in Figure 9 whereas the state trajectories look as shown in Figure 10.

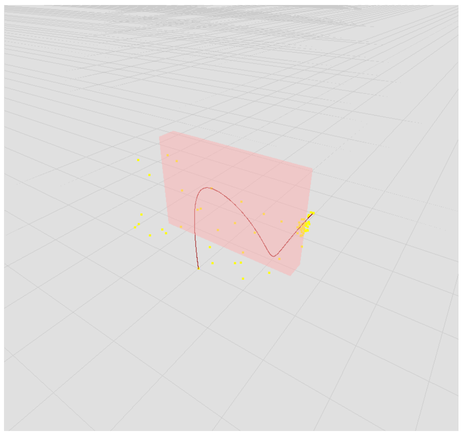
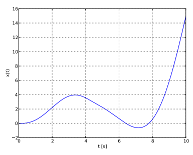
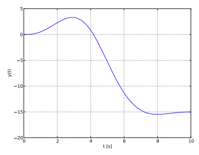


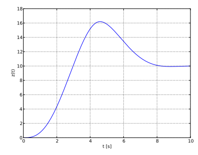
Fig. 9: Cornering around a wall

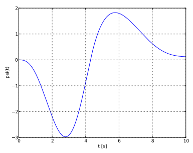
F. Hammerhead turn

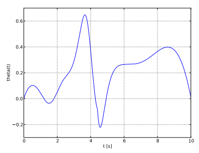
I also did experiments for a few other maneuvers such as the hammer-head turn, where a fixed wing aircraft


(a) x(t)


(b) y(t)


(c) z(t)


(d) ψ (t)


(e) θ (t)

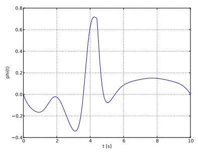

(f) ϕ (t)

Fig. 10: State trajectories for cornering around a wall. Note that there is a large roll when the quadrotor corners around the wall. There is slight unwanted movement to the left in the beginning but that is because there is a sample there actually connects. As more samples are taken, this should go away. I stopped the execution at 75 samples.

pitches up, stalls while doing a yaw and then comes down to result in the same velocity but in the opposite direction. In the case of a quadrotor however, it does not need to do this yaw. It pitches up, and comes down normally, thereby giving a trajectory as shown in Figure 11.

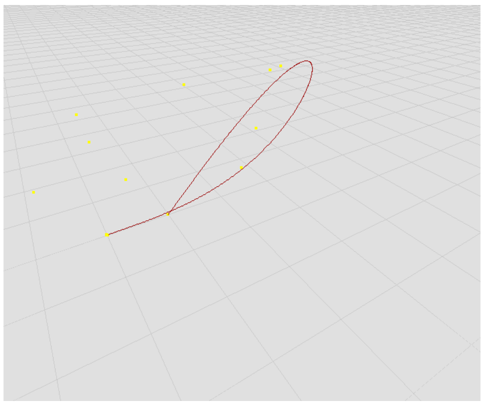


Fig. 11: A hammerhead turn

VIII. Conclusion

The property of differential flatness was described for a general nonlinear system. This enables an easy solution to the two point boundary value problem that is needed to be solved in optimal control. This work solved the optimal control by optimizing over the class of polynomials. An asymptotically optimal sampling based motion planning algorithm was then used to guarantee optimal trajectories using the point-to-point steering function. This demonstrated that differential flatness can be leveraged effectively to get optimal trajectories for traditionally hard nonlinear systems. Using sampling based algorithms also results in a more efficient solution to the trajectory tracking problem since we generate an optimal controller for every sampled state. A number of examples for aggressive time-topmal maneuvers using a quadrotor were shown using the proposed approach.

References

[1] Daniel Mellinger and Vijay Kumar. Minimum Snap Trajectory Generation and Control for Quadrotors. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2520–2525. IEEE, 2011.

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, 2007.

[3] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A Simple Learning Strategy for High-Speed Quadrocopter Multi-Flips. In *2010 IEEE International Conference on Robotics and Automation (ICRA 2010)*, pages 1642–1648. IEEE, 2010.

[4] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based optimal motion planning. In *Robotics: Science and Systems*, 2010.

[5] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Conference on Decision and Control*, 2010.

[6] Jeong hwan Jeon, Sertac Karaman, and Emilio Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. In *IEEE Conference on Decision and Control*, 2011. Submitted.

[7] Michel Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. Flatness and defect of non-linear systems: introductory theory and examples. *International Journal of Control*, 61(6):1327–1361, 1995.

[8] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Int. Symposium on Experimental Robotics*, 2010.

[9] S Karaman and E Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 2011.

[10] Michel Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. A Lie-Bäcklund approach to equivalence and flatness of nonlinear systems. *IEEE Transactions on Automatic Control*, 44(5):922–937, 1999.

[11] Mark B. Milam, Kudah Mushambi, and Richard M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000.

[12] T. Bresciani. *Modelling, identification and control of a quadrotor helicopter*. Department of Automatic Control, Lund University, 2008.