Laboratory Session 6 - trajectory generation

Trajectory generation and implementation using differential flatness

November, 2023

Contents

Throughout this Matlab session we will introduce some basic notions related to differential flatness which will be further used for the resolution of a trajectory planning problem. This is in principle an open-loop control problem arising in multi-agent motion planning. The a priory generated trajectory can be useful in a predictive control context which builds on a prediction capability over a finite horizon in the presence of state and/or input constraints. Therefore, the previous Matlab sessions can be easily combined with the present session.

1 Trajectory generation using flatness

Differential flatness was studied as a generalization of the structural properties of the linear systems, which exhibit a state representation obtained via derivatives of the input and output signals. The class of systems that exhibit the property of differential flatness were first studied by [3]. Other contributions to the topic were presented in [4], [1], [5].

Consider the general nonlinear system:

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_0, \; x(T) = x_f, \tag{1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector and $u(t) \in \mathbb{R}^m$ is the input vector.

Definition 1. The system (1) is called differentially flat if there exist variables $z(t) \in \mathbb{R}^m$ such that the states and inputs can be algebraically expressed in terms of $z(t)$ and a finite number of its higher-order derivatives:

$$\begin{aligned} x(t) &= \Phi_0(z(t), \dot{z}(t), \cdots, z^{(q)}(t)), \\ u(t) &= \Phi_1(z(t), \dot{z}(t), \cdots, z^{(q)}(t)), \end{aligned} \tag{2}$$

where $z(t) = \gamma(x(t), u(t), \dot{u}(t), \cdots, u^{(q)}(t))$ is called the flat output and q is the maximum order of $z(t)$ arising in the problem. □

Remark 1. One of the observation made in the differential flatness literature is that, for any linear and nonlinear flat system, the number of flat outputs equals the number of inputs [4]. □

Remark 2. Note also that, for linear systems, the flat differentiability (existence and constructive forms) is implied by the controllability property. □

The most important aspect of flatness is that it reduces the problem of trajectory generation for the original system modeled by an ordinary differential equation, to finding a trajectory of the flat outputs via the resolution of an algebraic system of equations. The trajectories are planned for $z(t)$ and then, transformed via $\Phi_0(\cdot)$ and $\Phi_1(\cdot)$ in (2) to obtain the trajectories of states and inputs. The most important aspect from the subsequent feedback control is that these trajectories are consistent with the system dynamic equation. Practically, any trajectory for $z(t)$ satisfying the boundary conditions:

$$\begin{aligned} x(0) = \Phi_0(z(0), \dot{z}(0), \cdots, z^{(q)}(0)) &= x_0, \\ x(T) = \Phi_0(z(T), \dot{z}(T), \cdots, z^{(q)}(T)) &= x_f, \\ u(0) = \Phi_1(z(0), \dot{z}(0), \cdots, z^{(q)}(0)) &= u_0, \\ u(T) = \Phi_1(z(T), \dot{z}(T), \cdots, z^{(q)}(T)) &= u_f, \end{aligned} \tag{3}$$

will be a feasible trajectory for the system that passes through the given initial and final conditions.

One objection that can be raised is that equation (3) still contains the derivatives of $z(t)$ and as such is not purely algebraic expression. However, the flat output in (3)

can be parametrized using a set of smooth basis functions $\Lambda^i(t)$ in order to allow the derivatives manipulation:

$$z(t) = \sum_{i=1}^{N_\alpha} \alpha_i \Lambda^i(t), \quad \alpha_i \in \mathbb{R}. \tag{4}$$

The set of coefficients α_i in (4), with $i = 1, \cdots, N_\alpha$ represents practically the unknown variables in the system described by the equations (3) which lead in this specific form (3)–(4) to a system of linear equations. In this framework the derivatives of the flat output can be computed in terms of the derivatives of the basis functions:

$$\dot{z}(t) = \sum_{i=1}^{N_\alpha} \alpha_i \dot{\Lambda}(t), \ldots, z^{(q)}(t) = \sum_{i=1}^{N_\alpha} \alpha_i \Lambda^{(q)}(t). \tag{5}$$

Subsequently, the conditions on the flat outputs and their derivatives are written as follows:

$$\begin{bmatrix} \Lambda_1(0) & \Lambda_2(0) & \cdots & \Lambda_{N_\alpha}(0) \\ \dot{\Lambda}_1(0) & \dot{\Lambda}_2(0) & \cdots & \dot{\Lambda}_{N_\alpha}(0) \\ \vdots & \vdots & & \vdots \\ \Lambda_1^{(q)}(0) & \Lambda_2^{(q)}(0) & \cdots & \Lambda_{N_\alpha}^{(q)}(0) \\ \Lambda_1(T) & \Lambda_2(T) & \cdots & \Lambda_{N_\alpha}(T) \\ \dot{\Lambda}_1(T) & \dot{\Lambda}_2(T) & \cdots & \dot{\Lambda}_{N_\alpha}(T) \\ \vdots & \vdots & & \vdots \\ \Lambda_1^{(q)}(T) & \Lambda_2^{(q)}(T) & \cdots & \Lambda_{N_\alpha}^{(q)}(T) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{N_\alpha} \end{bmatrix} = \begin{bmatrix} z(0) \\ \dot{z}(0) \\ \vdots \\ z^{(q)}(0) \\ z(T) \\ \dot{z}(T) \\ \vdots \\ z^{(q)}(T) \end{bmatrix}. \tag{6}$$

The formulation (6) takes the form of a system of *linear* equation, which can be rewritten in the form $M\alpha = \bar{z}$. Finally, assuming that M has a sufficient number of columns and it is a full column rank (classical well-possedness conditions), the trajectory generation problem is solved by finding the coefficient α (possibly non-unique).

Remark 3. Note that, the equations (3) represent typical conditions which appear in flat trajectory generation and can be solved as a classical system of equations as a function of parameters in the generation of the signal $z(t)$. This basic boundary problem can be modified and enhanced in several ways, if necessary. Firstly, additional constraints can be applied upon the derivatives of the state and the input, thus reducing the feasible set of solutions. Finally, we can consider conditions not only at the initial and the final times but also at any intermediate time instant, by increasing the number of equations (and adequately adjusting the number of parameters in order to ensure the well-posedness) [8]. □

Note that in (6) the conditions on the initial and final time instants can be extended for simplicity to the introduction of a set of way-points through which the flat trajectory must pass. This in equivalent to the introduction of additional rows in (6). Since a continuous trajectory (and smooth, usually) is needed, there are two ways of accomplishing it in the current framework. One is to apply (3)–(6) as they are, and then force the resulting "segments" to link (by matching conditions on the intermediary points) or,

otherwise, as mentioned in Remark 3, to solve everything as a single problem where we add conditions on intermediate time instants.

For the global trajectory planning it is important to point out that, there are different basis functions used for parametrization of the flat output ($\Lambda^i(t)$ in (4)), each with their advantages and disadvantages.

- A class of functions commonly used is represented by *polynomials* in the time variable – t^i, which are capable of representing curves in a flexible way. However, they have a poor numerical performance, their dimension depends on the number of conditions imposed on the inputs, states and their derivatives.
- A different class of polynomials for representing a curve is given by *Bésier basis functions*. The equivalence in terms of degrees of freedom is enhanced from the numerical condition point of view. This equivalence ensures that one representation can be translated into the other without loss of information [2], [9]. The limitation of Bésier curves is related again to the need of higher degrees to satisfy a large number of inequality and/or equality constraints (for more details, see for instance [7]).
- It is more convenient to represent the flat output $z(t)$ using *B-splines* [6], because of their ease of enforcing continuity across way-points and ease of computing their derivatives. Moreover, their degree depends only up to which derivative is needed to ensure continuity, this being in contrast with the polynomial basis functions previously mentioned.

2 Matlab implementation over an UAV system

We explore the case of a 2D 3-DOF model of an UAV in which the autopilot forces coordinated turns at a fixed altitude:

$$\begin{aligned} \dot{x}(t) &= V_{\mathrm{a}}(t)\cos\Psi(t), \\ \dot{y}(t) &= V_{\mathrm{a}}(t)\sin\Psi(t), \\ \dot{\Psi}(t) &= \frac{g\ \tan\Phi(t)}{V_{\mathrm{a}}(t)} \end{aligned} \tag{7}$$

The state variables are represented by the position $(x(t), y(t))$ and the heading (yaw) angle $\Psi(t) \in [0, 2\pi]$ rad. The input signals are the airspeed velocity $V_{\mathrm{a}}(t)$ and the roll angle $\Phi(t)$, respectively. The constraints are the following:

- air relative velocity $V_a \in [18, 25]$ m/s;
- bank angle $\Phi \in [-0.43, 0.43]$ rad;
- rate of change of V_a is limited to the maximum acceleration the aircraft can produce, i.e., $0.1 \sim 0.2$ m/s^2;
- variation of Φ is limited to $0.5 \sim 1.1$ rad/s.

We take as flat output the position components of the state, $z(t) = \begin{bmatrix} z_1(t) & z_2(t) \end{bmatrix}^T = \begin{bmatrix} x(t) & y(t) \end{bmatrix}^T$ which permits to compute the remaining variables:

$$\Psi(t) = \arctan\left(\frac{\dot{z}_2(t)}{\dot{z}_1(t)}\right), \tag{8a}$$

$$V_a(t) = \sqrt{\dot{z}_1^2(t) + \dot{z}_2^2(t)}, \tag{8b}$$

$$\Phi(t) = \arctan\left(\frac{1}{g}\frac{\ddot{z}_2(t)\dot{z}_1(t) - \dot{z}_2(t)\ddot{z}_1(t)}{\sqrt{\dot{z}_1^2(t) + \dot{z}_2^2(t)}}\right). \tag{8c}$$

Note that in the heading component of the state appear 1st order derivatives and in the roll angle input appear 2nd order derivatives of the flat outputs. Denoting $\xi(t) = \begin{bmatrix} x^T(t) & y^T(t) & \Psi^T(t) \end{bmatrix}^T \in \mathbb{R}^3$ the state vector, $u = \begin{bmatrix} V_a^T(t) & \Phi^T(t) \end{bmatrix}^T \in \mathbb{R}^2$ the input vector we have that:

$$\begin{array}{rcl} \xi^{ref}(t) & = & \Phi_0(z(t), \dot{z}(t)), \\ u^{ref}(t) & = & \Phi_1(z(t), \dot{z}(t), \ddot{z}(t)), \end{array} \tag{9}$$

Following the previous section the implementation algorithm for the trajectory generations is resumed:

- assume an interval $t \in [0, T]$
- boundary conditions $\xi^{ref}(0) = \xi_0, \xi^{ref}(T) = \xi_f$, $u^{ref}(0) = u_0, u^{ref}(T) = u_f$
- choose a basis function $\Lambda(t) = [\dots \Lambda^i(t) \dots]$
- parametrize the flat output $z(t) = \sum_{i=1}^{N_\alpha} \alpha_i \Lambda^i(t)$
- and its derivatives $z^{(q)}(t) = \sum_{i=1}^{N_\alpha} \alpha_i \Lambda^{(q)}(t)$
- obtain coefficients α_i from the boundary conditions
- go back to $\xi^{ref}(t)$ and $u^{ref}(t)$ and generate the reference trajectory.

2.1 Some hints of implementation

- You can first save into workspace all the collections of waypoints you would like to consider (e.g., create the cell array WPLists, to select the i-th set of WP points, call WPLists $\{i\}$).
- Next, you can construct a Matlab function which gives $\xi^{ref}(t)$ and $u^{ref}(t)$ as in (9). Here you can use the algorithm provided in Section 2:

```
% initialize the system parameters
param.g=9.8;
param.h=0.05;
param.dbl_Alt=150;

%initialize the time stamp, the state and input reference:
t=[];
xi_ref=[];
u_ref=[];

% define the intial and final conditions:
 xi=...;
 ui=...;
 xf=...;
 uf=...;
 T=...;

% construct the matrix M \alpha = \bar{z} as in eq. (6):

 M=[ 1 0 0   0     0       0       0 0 0   0     0       0; ...
     0 1 0   0     0       0       0 0 0   0     0       0; ...
     0 0 2   0     0       0       0 0 0   0     0       0; ...
     0 0 0   0     0       0       1 0 0   0     0       0; ...
     0 0 0   0     0       0       0 1 0   0     0       0; ...
     0 0 0   0     0       0       0 0 2   0     0       0; ...
     1 T T^2 T^3   T^4     T^5     0 0 0   0     0       0; ...
     0 1 2*T 3*T^2 4*T^3   5*T^4   0 0 0   0     0       0; ...
     0 0 2   2*3*T 3*4*T^2 4*5*T^3 0 0 0   0     0       0; ...
     0 0 0   0     0       0       1 T T^2 T^3   T^4     T^5; ...
     0 0 0   0     0       0       0 1 2*T 3*T^2 4*T^3   5*T^4;...
     0 0 0   0     0       0       0 0 2   2*3*T 3*4*T^2 4*5*T^3];

 b=[ xi(1) ui(1)*cos(xi(3)) -sin(xi(3))*param.g*tan(ui(2))...
     xi(2) ui(1)*sin(xi(3)) cos(xi(3))*param.g*tan(ui(2))...
     xf(1) uf(1)*cos(xf(3)) -sin(xf(3))*param.g*tan(uf(2))...
     xf(2) uf(1)*sin(xf(3)) cos(xf(3))*param.g*tan(uf(2))]';

 a=inv(M)*b;

% define the flat output and its derivatives

 z1=@(t)(a(1)+a(2)*t+a(3)*t.^2+a(4)*t.^3+a(5)*t.^4+a(6)*t.^5);
 z2=@(t)(a(7)+a(8)*t+a(9)*t.^2+a(10)*t.^3+a(11)*t.^4+a(12)*t.^5);

 zd1=@(t)(a(2)+2*a(3)*t+3*a(4)*t.^2+4*a(5)*t.^3+5*a(6)*t.^4);
 zd2=@(t)(a(8)+2*a(9)*t+3*a(10)*t.^2+4*a(11)*t.^3+5*a(12)*t.^4);

 zdd1=@(t)(2*a(3)+2*3*a(4)*t+3*4*a(5)*t.^2+4*5*a(6)*t.^3);
 zdd2=@(t)(2*a(9)+2*3*a(10)*t+3*4*a(11)*t.^2+4*5*a(12)*t.^3);
```

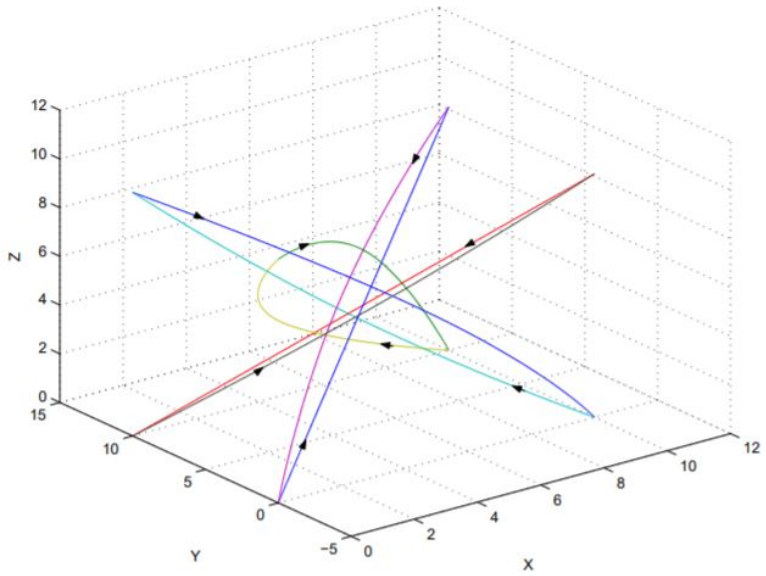
```
X=@(t)([feval(z1,t); feval(z2,t); atan2(feval(zd2,t),feval(zd1,t))]);
U=@(t)([sqrt((feval(zd1,t).^2)+(feval(zd2,t).^2));...
        atan2((feval(zdd2,t).*feval(zd1,t)-feval(zd2,t).*feval(zdd1,t)),
              (param.g.*(sqrt((feval(zd1,t).^2)+(feval(zd2,t).^2)))))]);

%compute the state and input reference:
 tt=param.h:param.h:T;
 xi_ref=[xi_ref feval(X,tt)];
 u_ref=[u_ref feval(U,tt)];
```

3 Exercises

The next exercises demand a longer time for solving them than a typical TP duration. However, these exercises can be considered as exam subjects.

Exercise 3.1. Consider the a priori generated trajectory as a reference to be followed in a constrained MPC problem. Provide some simulations.

Exercise 3.2. Generate flat trajectories for the UAV model in (7) that must execute a "Swapping Cube" maneuver. This means that 8 vehicles placed in the corners of a cube must change their places simultaneously without collision (see the figure).



Exercise 3.3. Enumerate some other possibilities for flat output parametrization. Matlab implementations for each possibility will highlight their advantages and disadvantages.

Exercise 3.4. In a multi-agent context explain in your own words how the trajectory generation modifies for centralized, decentralized or distributed control (overall formation stability and collision avoidance issues). Provide some simulations on specific scenarios in order to argument your answer.

References

[1] J. De Doná et al. "A flatness-based iterative method for reference trajectory generation in constrained NMPC". In: *Int. Workshop on Assesment and Future Direction of Nonlinear Model Predictive Control* (2009), pp. 325–333.

[2] G.E. Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*. Morgan Kaufmann, 5th edition, 2001.

[3] M. Fliess et al. "Flatness and defect of non-linear systems: introductory theory and examples". In: *International journal of control* 61.6 (1995), pp. 1327–1361.

[4] J. Lévine. *Analysis and control of nonlinear systems: A flatness-based approach*. Springer Verlag, 2009.

[5] R.M. Murray. "Optimization-Based Control". In: *Technical Report, California Institute of Technology, CA* (2009).

[6] L.L. Schumaker. *Spline functions: basic theory*. Cambridge Univ Pr, 2007.

[7] F. Suryawan. "Constrained Trajectory Generation and Fault Tolerant Control Based on Differential Flatness and B-splines". PhD thesis. 2012.

[8] J.H. Wilkinson. *The algebraic eigenvalue problem*. Vol. 155. Oxford Univ. Press, 1965.

[9] F. Yamaguchi and F. Yamaguchi. *Curves and surfaces in computer aided geometric design*. Springer-Verlag Berlin, 1988.