

Optimization-Based Control

Richard M. Murray
Control and Dynamical Systems
California Institute of Technology

Version v2.3c (08 Jan 2023)

© California Institute of Technology
All rights reserved.

This manuscript is for personal use only and may not be reproduced,
in whole or in part, without written consent from the author.

Chapter 2

Trajectory Generation and Tracking

This chapter expands on Section 8.5 of FBS2e, which introduces the use of feed-forward compensation in control system design. We begin with a review of the two degree of freedom design approach and then focus on the problem of generating feasible trajectories for a (nonlinear) control system. We make use of the concept of differential flatness as a tool for generating feasible trajectories.

Prerequisites. Readers should be familiar with modeling of input/output control systems using differential equations, linearization of a system around an equilibrium point, and state space control of linear systems, including reachability and eigenvalue assignment. Although this material supplements concepts introduced in the context of output feedback and state estimation, no knowledge of observers is required.

2.1 Two Degree of Freedom Design

A large class of control problems consists of planning and following a trajectory in the presence of noise and uncertainty. Examples include autonomous vehicles maneuvering in city streets, mobile robots performing tasks on factory floors (or other planets), manufacturing systems that regulate the flow of parts and materials through a plant or factory, and supply chain management systems that balance orders and inventories across an enterprise. All of these systems are highly nonlinear and demand accurate performance.

To control such systems, we make use of the notion of *two degree of freedom* controller design. This is a standard technique in linear control theory that separates a controller into a feedforward compensator and a feedback compensator. The feedforward compensator generates the nominal input required to track a given reference trajectory. The feedback compensator corrects for errors between the desired and actual trajectories. This is shown schematically in Figure 2.1.

In a nonlinear setting, two degree of freedom controller design decouples the trajectory generation and asymptotic tracking problems. Given a desired output

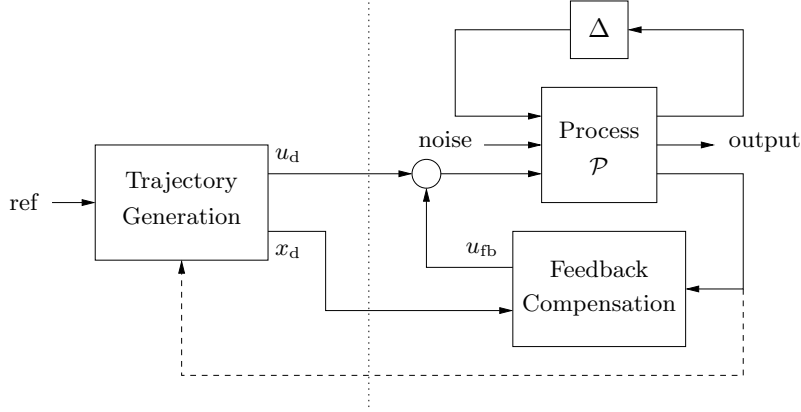


Figure 2.1: Two degree of freedom controller design for a process P with uncertainty Δ . The controller consists of a trajectory generator and feedback controller. The trajectory generation subsystem computes a feedforward command u_d along with the desired state x_d . The state feedback controller uses the measured (or estimated) state and desired state to compute a corrective input u_{fb} . Uncertainty is represented by the block Δ , representing unmodeled dynamics, as well as disturbances and noise. The dashed line represents the use of the current system state for real-time trajectory generation (described in more detail in Chapter 4).

trajectory, we first construct a state space trajectory x_d and a nominal input u_d that satisfy the equations of motion. The error system can then be written as a time-varying control system in terms of the error, $e = x - x_d$. Under the assumption that that tracking error remains small, we can linearize this time-varying system about $e = 0$ and stabilize the $e = 0$ state. (Note: in FBS2e the notation u_{ff} is used for the desired [feedforward] input. We use u_d here to match the desired state x_d .)

More formally, we assume that our process dynamics can be described by a nonlinear differential equation of the form

$$\begin{aligned} \dot{x} &= f(x, u), & x &\in \mathbb{R}^n, u \in \mathbb{R}^m, \\ y &= h(x, u), & y &\in \mathbb{R}^p, \end{aligned} \quad (2.1)$$

where x is the system state, u is a vector of inputs, and f is a smooth function describing the dynamics of the process. The smooth function h describes the output y that we wish to control. We are particularly interested in the class of control problems in which we wish to track a time-varying reference trajectory $r(t)$, called the *trajectory tracking* problem. In particular, we wish to find a control law $u = \alpha(x, r(\cdot))$ such that

$$\lim_{t \rightarrow \infty} (y(t) - r(t)) = 0.$$

We use the notation $r(\cdot)$ to indicate that the control law can depend not only on the reference signal $r(t)$ but also derivatives of the reference signal.

A *feasible trajectory* for the system (2.1) is a pair $(x_d(t), u_d(t))$ that satisfies the differential equation and generates the desired trajectory:

$$\dot{x}_d(t) = f(x_d(t), u_d(t)), \quad r(t) = h(x_d(t), u_d(t)).$$

The problem of finding a feasible trajectory for a system is called the *trajectory generation* problem, with x_d representing the desired state for the (nominal) system and u_d representing the desired input or the feedforward control. If we can find a feasible trajectory for the system, we can search for controllers of the form $u = \alpha(x, x_d, u_d)$ that track the desired reference trajectory.

In many applications, it is possible to attach a cost function to trajectories that describe how well they balance trajectory tracking with other factors, such as the magnitude of the inputs required. In such applications, it is natural to ask that we find the *optimal* controller with respect to some cost function. We can again use the two degree of freedom paradigm with an optimal control computation for generating the feasible trajectory. This subject is examined in more detail in Chapter 3. In addition, we can take the extra step of updating the generated trajectory based on the current state of the system. This additional feedback path is denoted by a dashed line in Figure 2.1 and allows the use of so-called *receding horizon control* techniques: a (optimal) feasible trajectory is computed from the current position to the desired position over a finite time T horizon, used for a short period of time $\delta < T$, and then recomputed based on the new system state. Receding horizon control is described in more detail in Chapter 4.

A key advantage of optimization-based approaches is that they allow the potential for customization of the controller based on changes in *mission*, *condition* and *environment*. Because the controller is solving the optimization problem online, updates can be made to the cost function, to change the desired operation of the system; to the model, to reflect changes in parameter values or damage to sensors and actuators; and to the constraints, to reflect new regions of the state space that must be avoided due to external influences. Thus, many of the challenges of designing controllers that are robust to a large set of possible uncertainties become embedded in the online optimization.

2.2 Trajectory Tracking and Gain Scheduling

We begin by considering the problem of tracking a feasible trajectory. Assume that a trajectory generator is able to generate a trajectory (x_d, u_d) that satisfies the dynamics (2.1) and satisfies $r(t) = h(x_d(t), u_d(t))$. To design the controller, we construct the *error system*. Let $e = x - x_d$ and $v = u - u_d$ and compute the dynamics for the error:

$$\begin{aligned} \dot{e} &= \dot{x} - \dot{x}_d = f(x, u) - f(x_d, u_d) \\ &= f(e + x_d, v + u_d) - f(x_d, u_d) =: F(e, v, x_d(t), u_d(t)). \end{aligned}$$

The function F represents the dynamics of the error, with control input v and external inputs x_d and u_d . In general, this system is time-varying through the desired state and input.

For trajectory tracking, we can assume that e is small (if our controller is doing a good job), and so we can linearize around $e = 0$:

$$\frac{de}{dt} \approx A(t)e + B(t)v, \quad A(t) = \left. \frac{\partial F}{\partial e} \right|_{(x_d(t), u_d(t))}, \quad B(t) = \left. \frac{\partial F}{\partial v} \right|_{(x_d(t), u_d(t))}.$$

It is often the case that $A(t)$ and $B(t)$ depend only on x_d , in which case it is convenient to write $A(t) = A(x_d)$ and $B(t) = B(x_d)$.

We start by reviewing the case where $A(t)$ and $B(t)$ are constant, in which case our error dynamics become

$$\dot{e} = Ae + Bv.$$

This occurs, for example, if the original nonlinear system is linear. We can then search for a control system of the form

$$v = -Ke.$$

We can now apply the results of Chapter 7 of FBS2e and solve the problem by finding a gain matrix K that gives the desired closed loop dynamics (e.g., by eigenvalue assignment). It can be shown that this formulation is equivalent to a two degree of freedom design where x_d and u_d are chosen to give the desired reference output (Exercise 2.1).

Returning to the full nonlinear system, assume now that x_d and u_d are either constant or slowly varying (with respect to the performance criterion). This allows us to consider just the (constant) linearized system given by $(A(x_d), B(x_d))$. If we design a state feedback controller $K(x_d)$ for each x_d , then we can regulate the system using the feedback

$$v = -K(x_d)e.$$

Substituting back the definitions of e and v , our controller becomes

$$u = u_d - K(x_d)(x - x_d).$$

Note that the controller $u = \alpha(x, x_d, u_d)$ depends on (x_d, u_d) , which themselves depend on the desired reference trajectory. This form of controller is called a *gain scheduled* linear controller with feedforward input u_d .

More generally, the term gain scheduling is used to describe any controller that depends on a set of measured parameters in the system. So, for example, we might write

$$u = u_d - K(x, \mu) \cdot (x - x_d),$$

where $K(x, \mu)$ depends on the *current* system state (or some portion of it) and an external parameter μ . The dependence on the current state x (as opposed to the desired state x_d) allows us to modify the closed loop dynamics differently depending on our location in the state space. This is particularly useful when the dynamics of the process vary depending on some subset of the states (such as the altitude for an aircraft or the internal temperature for a chemical reaction). The dependence on μ can be used to capture the dependence on the reference trajectory, or they can reflect changes in the environment or performance specifications that are not modeled in the state of the controller.

Example 2.1 Steering control with velocity scheduling

Consider the problem of controlling the motion of an automobile so that it follows a given trajectory on the ground, as shown in Figure 2.2a. We use the model derived in FBS2e, Example 3.11, choosing the reference point to be the center of the rear wheels. This gives dynamics of the form

$$\dot{x} = \cos \theta v, \quad \dot{y} = \sin \theta v, \quad \dot{\theta} = \frac{v}{l} \tan \phi, \quad (2.2)$$

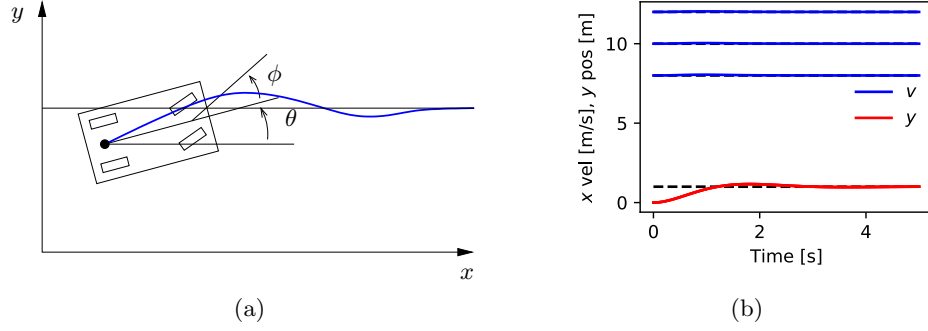


Figure 2.2: Vehicle steering using gain scheduling.

where (x, y, θ) is the position and orientation of the vehicle, v is the velocity and ϕ is the steering angle, both considered to be inputs, and l is the wheelbase.

A simple feasible trajectory for the system is to follow a straight line in the x direction at lateral position y_r and fixed velocity v_r . This corresponds to a desired state $x_d = (v_r t, y_r, 0)$ and nominal input $u_d = (v_r, 0)$. Note that (x_d, u_d) is not an equilibrium point for the system, but it does satisfy the equations of motion.

Linearizing the system about the desired trajectory, we obtain

$$A_d = \left. \frac{\partial f}{\partial x} \right|_{(x_d, u_d)} = \begin{bmatrix} 0 & 0 & -\sin \theta v \\ 0 & 0 & \cos \theta v \\ 0 & 0 & 0 \end{bmatrix} \bigg|_{(x_d, u_d)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v_r \\ 0 & 0 & 0 \end{bmatrix},$$

$$B_d = \left. \frac{\partial f}{\partial u} \right|_{(x_d, u_d)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & v_r/l \end{bmatrix}.$$

We form the error dynamics by setting $e = x - x_d$ and $w = u - u_d$:

$$\dot{e}_x = w_1, \quad \dot{e}_y = e_\theta, \quad \dot{e}_\theta = \frac{v_r}{l} w_2.$$

We see that the first state is decoupled from the second two states and hence we can design a controller by treating these two subsystems separately. Suppose that we wish to place the closed loop eigenvalues of the longitudinal dynamics (e_x) at $-\lambda_1$ and place the closed loop eigenvalues of the lateral dynamics (e_y, e_θ) at the roots of the polynomial equation $s^2 + a_1 s + a_2 = 0$.

This can be accomplished by setting

$$w_1 = -\lambda_1 e_x$$

$$w_2 = -\frac{l}{v_r} \left(\frac{a_2}{v_r} e_y + a_1 e_\theta \right).$$

Note that the gains depend on the velocity v_r (or equivalently on the nominal input u_d), giving us a gain scheduled controller.

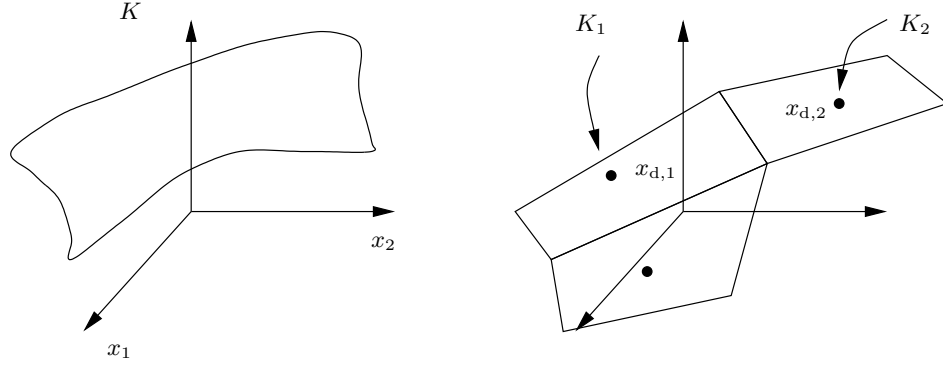


Figure 2.3: Gain scheduling. A general gain scheduling design involves finding a gain K at each desired operating point. This can be thought of as a gain surface, as shown on the left (for the case of a scalar gain). An approximation to this gain can be obtained by computing the gains at a fixed number of operating points and then interpolated between those gains. This gives an approximation of the continuous gain surface, as shown on the right.

In the original inputs and state coordinates, the controller has the form

$$\begin{bmatrix} v \\ \phi \end{bmatrix} = \underbrace{\begin{bmatrix} v_r \\ 0 \end{bmatrix}}_{u_d} - \underbrace{\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \frac{a_2 l}{v_r^2} & \frac{a_1 l}{v_r} \end{bmatrix}}_{K(x_d, u_d)} \underbrace{\begin{bmatrix} x - v_r t \\ y - y_r \\ \theta \end{bmatrix}}_e.$$

The form of the controller shows that at low speeds the gains in the steering angle will be high, meaning that we must turn the wheel harder to achieve the same effect. As the speed increases, the gains become smaller. This matches the usual experience that at high speed a very small amount of actuation is required to control the lateral position of a car. Note that the gains go to infinity when the vehicle is stopped ($v_r = 0$), corresponding to the fact that the system is not reachable at this point.

Figure 2.2b shows the response of the controller to a step change in lateral position at three different reference speeds. Notice that the rate of the response is constant, independent of the reference speed, reflecting the fact that the gain scheduled controllers each set the closed loop poles to the same values. ∇

One limitation of gain scheduling as we have described it is that a separate set of gains must be designed for each operating condition x_d . In practice, gain scheduled controllers are often implemented by designing controllers at a fixed number of operating points and then interpolating the gains between these points, as illustrated in Figure 2.3. Suppose that we have a set of operating points $x_{d,j}$, $j = 1, \dots, N$. Then we can write our controller as

$$u = u_d - K(x)e \quad K(x) = \sum_{j=1}^N \rho_j(x) K_j,$$

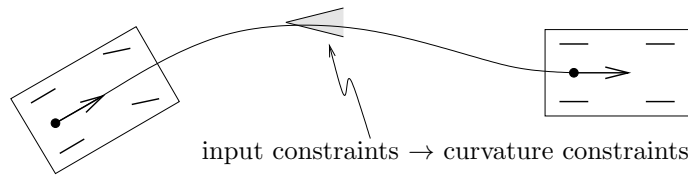


Figure 2.4: Simple model for an automobile. We wish to find a trajectory from an initial state to a final state that satisfies the dynamics of the system and constraints on the curvature (imposed by the limited travel of the front wheels).

where K_j is a set of gains designed around the operating point $x_{d,j}$ and $\rho_j(x)$ is a weighting factor. For example, we might choose the weights $\rho_j(x)$ such that we take the gains corresponding to the nearest two operating points and weight them according to the Euclidean distance of the current state from that operating point; if the distance is small then we use a weight very near to 1 and if the distance is far then we use a weight very near to 0.

While the intuition behind gain scheduled controllers is fairly clear, some caution is required in using them. In particular, a gain scheduled controller is not guaranteed to be stable even if $K(x, \mu)$ locally stabilizes the system around a given equilibrium point. Gain scheduling can be proven to work in the case when the gain varies sufficiently slowly (Exercise 2.4).

2.3 Trajectory Generation and Differential Flatness

We now return to the problem of generating a trajectory for a nonlinear system. Consider first the case of finding a trajectory $x_d(t)$ that steers the system from an initial condition x_0 to a final condition x_f . We seek a feasible solution $(x_d(t), u_d(t))$ that satisfies the dynamics of the process:

$$\dot{x}_d = f(x_d, u_d), \quad x_d(0) = x_0, \quad x_d(T) = x_f. \quad (2.3)$$

Formally, this problem corresponds to a two-point boundary value problem and can be quite difficult to solve in general.

In addition, we may wish to satisfy additional constraints on the dynamics. These can include input saturation constraints $|u(t)| < M$, state constraints $g(x) \leq 0$, and tracking constraints $h(x) = r(t)$, each of which gives an algebraic constraint on the states or inputs at each instant in time. We can also attempt to optimize a function by choosing $(x_d(t), u_d(t))$ to minimize

$$\int_0^T L(x, u) dt + V(x(T), u(T)).$$

As an example of the type of problem we would like to study, consider the problem of steering a car from an initial condition to a final condition, as shown in Figure 2.4. To solve this problem, we must find a solution to the differential equations (2.2) that satisfies the endpoint conditions. Given the nonlinear nature

of the dynamics, it seems unlikely that one could find explicit solutions that satisfy the dynamics except in very special cases (such as driving in a straight line).

A closer inspection of this system shows that it is possible to understand the trajectories of the system by exploiting the particular structure of the dynamics. Suppose that we are given a trajectory for the rear wheels of the system, $x_d(t)$ and $y_d(t)$. From equation (2.2), we see that we can use this solution to solve for the angle of the car by writing

$$\frac{\dot{y}}{\dot{x}} = \frac{\sin \theta}{\cos \theta} \quad \Longrightarrow \quad \theta_d = \tan^{-1}(\dot{y}_d/\dot{x}_d).$$

Furthermore, given θ we can solve for the velocity using the equation

$$\dot{x} = v \cos \theta \quad \Longrightarrow \quad v_d = \dot{x}_d / \cos \theta_d,$$

assuming $\cos \theta_d \neq 0$ (if it is, use $v = \dot{y} / \sin \theta$). And given θ , we can solve for ϕ using the relationship

$$\dot{\theta} = \frac{v}{l} \tan \phi \quad \Longrightarrow \quad \phi_d = \tan^{-1}\left(\frac{l\dot{\theta}_d}{v_d}\right).$$

Hence all of the state variables and the inputs can be determined by the trajectory of the rear wheels and its derivatives. This property of a system is known as *differential flatness*.

Definition 2.1 (Differential flatness). A nonlinear system (2.1) is *differentially flat* if there exists a function α such that

$$z = \alpha(x, u, \dot{u} \dots, u^{(p)}),$$

and we can write the solutions of the nonlinear system as functions of z and a finite number of derivatives:

$$\begin{aligned} x &= \beta(z, \dot{z}, \dots, z^{(q)}), \\ u &= \gamma(z, \dot{z}, \dots, z^{(q)}). \end{aligned} \tag{2.4}$$

The collection of variables $\bar{z} = (z, \dot{z}, \dots, z^{(q)})$ is called the *flat flag*.

For a differentially flat system, all of the feasible trajectories for the system can be written as functions of a flat output $z(\cdot)$ and its derivatives. The number of flat outputs is always equal to the number of system inputs. The kinematic car is differentially flat with the position of the rear wheels as the flat output. Differentially flat systems were originally studied by Fliess et al. [FLMR92].

Differentially flat systems are useful in situations where explicit trajectory generation is required. Since the behavior of a flat system is determined by the flat outputs, we can plan trajectories in output space, and then map these to appropriate inputs. Suppose we wish to generate a feasible trajectory for the the nonlinear system

$$\dot{x} = f(x, u), \quad x(0) = x_0, x(T) = x_f.$$

If the system is differentially flat then

$$\begin{aligned} x(0) &= \beta(z(0), \dot{z}(0), \dots, z^{(q)}(0)) = x_0, \\ x(T) &= \gamma(z(T), \dot{z}(T), \dots, z^{(q)}(T)) = x_f, \end{aligned} \tag{2.5}$$

and we see that the initial and final condition in the full state space depend on just the output z and its derivatives at the initial and final times. Thus any trajectory for z that satisfies these boundary conditions will be a feasible trajectory for the system, using equation (2.4) to determine the full state space and input trajectories.

In particular, given initial and final conditions on z and its derivatives that satisfy equation (2.5), any curve $z(\cdot)$ satisfying those conditions will correspond to a feasible trajectory of the system. We can parameterize the flat output trajectory using a set of smooth basis functions $\psi_i(t)$:

$$z(t) = \sum_{i=1}^N a_i \psi_i(t), \quad a_i \in \mathbb{R}.$$

We seek a set of coefficients a_i , $i = 1, \dots, N$ such that $z(t)$ satisfies the boundary conditions (2.5). The derivatives of the flat output can be computed in terms of the derivatives of the basis functions:

$$\begin{aligned} \dot{z}(t) &= \sum_{i=1}^N a_i \dot{\psi}_i(t) \\ &\vdots \\ \dot{z}^{(q)}(t) &= \sum_{i=1}^N a_i \psi_i^{(q)}(t). \end{aligned}$$

We can thus write the conditions on the flat outputs and their derivatives as

$$\begin{bmatrix} \psi_1(0) & \psi_2(0) & \dots & \psi_N(0) \\ \dot{\psi}_1(0) & \dot{\psi}_2(0) & \dots & \dot{\psi}_N(0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(0) & \psi_2^{(q)}(0) & \dots & \psi_N^{(q)}(0) \\ \psi_1(T) & \psi_2(T) & \dots & \psi_N(T) \\ \dot{\psi}_1(T) & \dot{\psi}_2(T) & \dots & \dot{\psi}_N(T) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(T) & \psi_2^{(q)}(T) & \dots & \psi_N^{(q)}(T) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} z(0) \\ \dot{z}(0) \\ \vdots \\ z^{(q)}(0) \\ z(T) \\ \dot{z}(T) \\ \vdots \\ z^{(q)}(T) \end{bmatrix} \quad (2.6)$$

This equation is a *linear* equation of the form $Ma = \bar{z}$. Assuming that M has a sufficient number of columns and that it is full column rank, we can solve for a (possibly non-unique) a that solves the trajectory generation problem.

Example 2.2 Nonholonomic integrator

A simple nonlinear system called a *nonholonomic integrator* [Bro81] is given by the differential equations

$$\dot{x}_1 = u_1, \quad \dot{x}_2 = u_2, \quad \dot{x}_3 = x_2 u_1.$$

This system is differentially flat with flat output $z = (x_1, x_3)$. The relationship between the flat variables and the states is given by

$$x_1 = z_1, \quad x_2 = \dot{x}_3 / \dot{x}_1 = \dot{z}_2 / \dot{z}_1, \quad x_3 = z_2. \quad (2.7)$$

Using simple polynomials as our basis functions,

$$\begin{aligned} \psi_{1,1}(t) = 1, \quad \psi_{1,2}(t) = t, \quad \psi_{1,3}(t) = t^2, \quad \psi_{1,4}(t) = t^3, \\ \psi_{2,1}(t) = 1 \quad \psi_{2,2}(t) = t, \quad \psi_{2,3}(t) = t^2, \quad \psi_{2,4}(t) = t^3, \end{aligned}$$

the equations for the feasible (flat) trajectory become

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & T & T^2 & T^3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2T & 3T^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T & T^2 & T^3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2T & 3T^2 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \end{bmatrix} = \begin{bmatrix} x_{1,0} \\ 1 \\ x_{3,0} \\ x_{2,0} \\ x_{1,f} \\ 1 \\ x_{3,f} \\ x_{2,f} \end{bmatrix}.$$

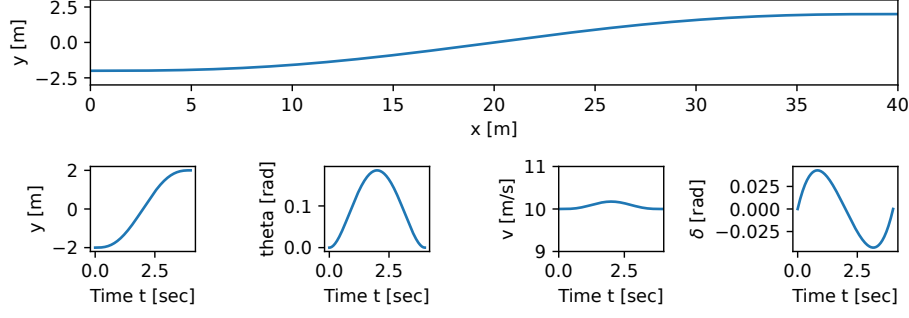
This is a set of 8 linear equations in 8 variables. It can be shown that the matrix M is full rank when $T \neq 0$ and the system can be solved numerically. ∇

Note that no ODEs need to be integrated in order to compute the feasible trajectories for a differentially flat system (unlike optimal control methods that we will consider in the next chapter, which involve parameterizing the *input* and then solving the ODEs). This is the defining feature of differentially flat systems. The practical implication is that nominal trajectories and inputs that satisfy the equations of motion for a differentially flat system can be computed in a computationally efficient way (solving a set of algebraic equations). Since the flat output functions do not have to obey a set of differential equations, the only constraints that must be satisfied are the initial and final conditions on the endpoints, their tangents, and higher order derivatives. Any other constraints on the system, such as bounds on the inputs, can be transformed into the flat output space and (typically) become limits on the curvature or higher order derivative properties of the curve.

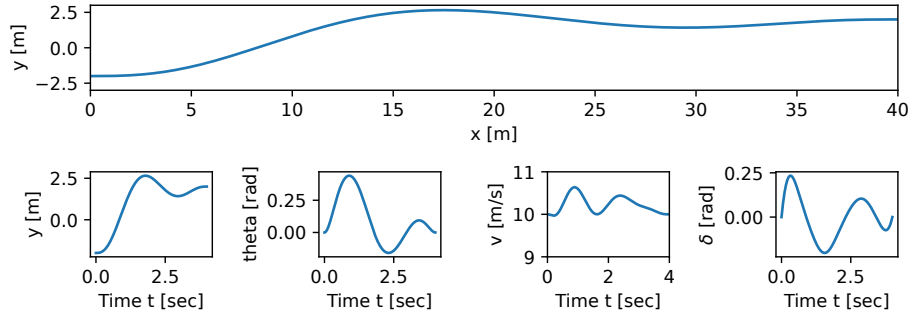
In the example above we had exactly the same number of basis functions as the total number of initial and final conditions, but it is also possible to choose a larger number of basis functions and use the remaining degrees of freedom for other purposes. For example, if there is a performance index for the system, this index can be transformed and becomes a functional depending on the flat outputs and their derivatives up to some order. By approximating the performance index we can achieve paths for the system that are suboptimal but still feasible. This approach is often much more appealing than the traditional method of approximating the system (for example by its linearization) and then using the exact performance index, which yields optimal paths but for the wrong system.

Example 2.3 Vehicle steering

Consider the vehicle steering example described at the start of this section and illustrated in Figure 2.4. The system consists of 3 states and 2 inputs and so we could potentially find solutions for the point-to-point trajectory generation problem using only 3 basis function in each of the two inputs. Suppose instead that we use a larger number of basis functions in each of the two inputs, allowing additional degrees of freedom.



(a) Least squares solution



(b) Penalize lateral error, input cost

Figure 2.5: Trajectory generation for vehicle steering example.

One possible solution for the now underdetermined linear set of equations that we obtain in equation (2.6) is to use the least squares solution of the linear equation $Ma = \bar{z}$, which provides the smallest possible a (coefficient vector) that satisfies the equation. The results of applying this to the problem of changing lanes using a polynomial basis, are shown in Figure 2.5a.

Suppose instead that we wish to change lanes faster, but also take into account the size of the inputs that are required. For example, we could seek to minimize the cost function

$$J(x, u) = \int_0^T ((y(\tau) - y_f)^2 + (v(\tau) - v_f)^2 + 10\delta^2(\tau)) d\tau,$$

where y is the lateral position of the vehicle, v is the vehicle velocity, δ is the steering wheel angle, and the subscript ‘f’ represents the final value. Using the free coefficients so as to minimize this cost, we obtain the results shown in Figure 2.5b. We see that the resulting trajectory transitions between the lanes more quickly, thought at the expense of larger inputs. ∇

In light of the techniques that are available for differentially flat systems, the characterization of flat systems becomes particularly important. General conditions for flatness are complicated to apply [Lév10], but many important classes

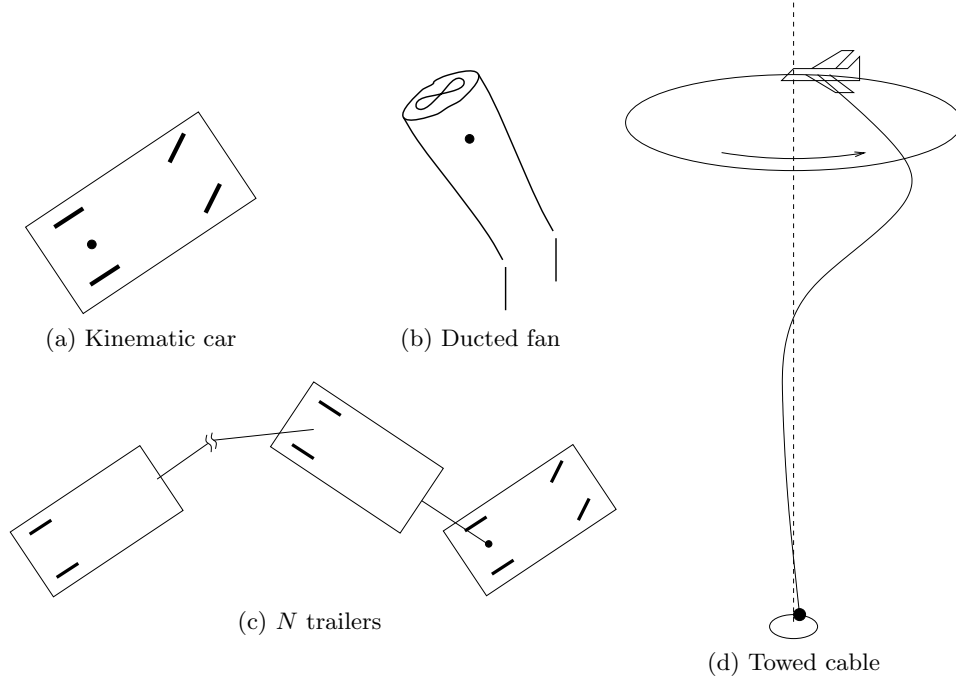


Figure 2.6: Examples of flat systems.

of nonlinear systems, including feedback linearizable systems, can be shown to be differentially flat. One large class of flat systems are those in “pure feedback form”:

$$\begin{aligned}
 \dot{x}_1 &= f_1(x_1, x_2) \\
 \dot{x}_2 &= f_2(x_1, x_2, x_3) \\
 &\vdots \\
 \dot{x}_n &= f_n(x_1, \dots, x_n, u).
 \end{aligned} \tag{2.8}$$

Under certain regularity conditions these systems are differentially flat with output $y = x_1$. These systems have been used for so-called “integrator backstepping” approaches to nonlinear control by Kokotovic et al. [KKM91]. Figure 2.6 shows some additional systems that are differentially flat.

Example 2.4 Vectored thrust aircraft

Consider the dynamics of a planar, vectored thrust flight control system as shown in Figure 2.7. This system consists of a rigid body with body fixed forces and is a simplified model for a vertical take-off and landing aircraft (see Example 3.12 in FBS2e). Let (x, y, θ) denote the position and orientation of the center of mass of the aircraft. We assume that the forces acting on the vehicle consist of a force F_1 perpendicular to the axis of the vehicle acting at a distance r from the center of mass and a force F_2 parallel to the axis of the vehicle. Let m be the mass of the vehicle, J the moment of inertia, and g the gravitational constant. We ignore aerodynamic forces for the purpose of this example.

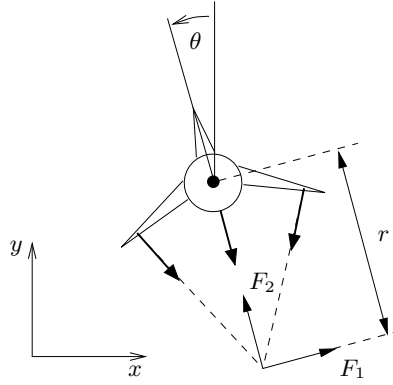


Figure 2.7: Vectored thrust aircraft (from FBS2e). The net thrust on the aircraft can be decomposed into a horizontal force F_1 and a vertical force F_2 acting at a distance r from the center of mass.

The dynamics for the system are

$$\begin{aligned} m\ddot{x} &= F_1 \cos \theta - F_2 \sin \theta - c\dot{x}, \\ m\ddot{y} &= F_1 \sin \theta + F_2 \cos \theta - mg - c\dot{y}, \\ J\ddot{\theta} &= rF_1. \end{aligned} \quad (2.9)$$

Martin et al. [MDP94] showed that when $c = 0$ this system is differentially flat and that one set of flat outputs is given by

$$\begin{aligned} z_1 &= x - (J/mr) \sin \theta, \\ z_2 &= y + (J/mr) \cos \theta. \end{aligned} \quad (2.10)$$

Using the system dynamics, it can be shown that

$$\ddot{z}_1 \cos \theta + (\ddot{z}_2 + g) \sin \theta = 0, \quad (2.11)$$

and thus given $z_1(t)$ and $z_2(t)$ we can find $\theta(t)$ except for an ambiguity of π and away from the singularity $\ddot{z}_1 = \ddot{z}_2 + g = 0$. The remaining states and the forces $F_1(t)$ and $F_2(t)$ can then be obtained from the dynamic equations, all in terms of z_1 , z_2 , and their higher order derivatives. ∇

Additional remarks on differential flatness



Determining whether a system is differentially flat is a challenging problem. Necessary and sufficient conditions have been developed by Lévine [Lév10], but the conditions are not constructive in nature. We briefly summarize here some known conditions under which a system is differentially flat as well as some additional concepts related to flatness.

Flatness of linear systems and feedback linearizable systems. All single-input reachable linear systems are differentially flat, which can be shown by putting the system into reachable canonical form (FBS2e, equation (7.6)) and choosing the last state

as the flat output. Multi-input, reachable linear systems are often differentially flat, though the construction of the flat outputs can be more complicated. Similarly, systems that are feedback linearizable (see FBS2e, Figure 6.15) are commonly differentially flat, as are systems in pure feedback form (2.8) (under appropriate regularity conditions). More details on some of the conditions under which feedback linearizable systems are differentially flat can be found in [vNRM98].

For mechanical systems, where the equations of motion satisfy Lagrange's equations and the state variables consist of configuration variables and their velocities (or momenta), it is often the case that the flat outputs are functions only of the configuration variables. In many cases, these flat outputs also have some geometric interpretation. For example, for the vectored thrust aircraft in Example 2.4 the flat output is a point on the axis of the aircraft whose position is based on the physical parameters of the system, as seen in equation (2.10). For the N -trailer system the flat output is the center of rear wheels of the final trailer (Exercise 2.8), and for the towed cable system in Figure 2.6d the bottom of the cable is the flat output [Mur96]. Other examples of configuration flat systems, and some geometric conditions characterizing configuration flatness, can be found in [RM98].

Structure of the flat flag. In Definition 2.1, the flat flag was defined as having the form $\bar{z} = (z, \dot{z}, \dots, z^{(q)})$, which implies that in the multi-input case the number of derivatives for each flat variable is the same. This need not be the case and it may be that a different number derivatives are required for different flat outputs, so that the flat flag has the structure $\bar{z}_i = (z_i, \dot{z}_i, \dots, z_i^{(q_i)})$, where i is the flat output and q_i is the number of derivatives for that output (which may not be the same for all i). The number of derivatives required in each flat output provide insights into the structure of the underlying system, with the dynamics of the system being equivalent to a set of chains of integrators of different lengths.

Related to this, in some instances, it can be the case that when finding the mapping from states and inputs of the system to the flat flag, derivatives of the inputs can appear. While this is allowable in the context of flatness (with appropriate extensions of the definition), in practice it often turns out that one can take the inputs as constants when computing the flat flag. This is allowable in many situations because the only time we use the mapping from the states and inputs to the flat flag is when determining the endpoints for a point-to-point trajectory generation problem. In that setting, constraining the derivative of the input to be zero at the start and end of the trajectory is often acceptable.

Partially flat systems and defect. For systems that are not differentially flat, it is sometimes possible to find a set of outputs for which a portion of the states can be determined from the flat outputs and their derivatives, but some set of states must still satisfy a set of differential equations. For example, we may be able to write the states of the system in the form

$$\begin{aligned} x &= \beta(z, \dot{z}, \dots, z^{(q)}), \\ u &= \gamma(z, \dot{z}, \dots, z^{(q)}), \\ \Phi &= \Phi(z, \dot{z}, \dots, z^{(n-r)}), \end{aligned}$$

where $z \in \mathbb{R}^p$, $p \geq m$ represents a set of outputs that parameterize the trajectory and Φ represents a set of remaining differential constraints on the output. The

minimum number of possible remaining differential constraints is called the *defect* of the system. Even though such a system is not differentially flat, the problem of trajectory generation may still be considerably simplified since the number of differential equation constraints may be small. In some cases, it even can turn out that even if a system is differentially flat the computations required to find the states based on the flat outputs may be sufficiently complicated that it is beneficial to use a set of outputs that only partially determine the system states.

A more complete description of the concept of defect for nonlinear systems is provided in [FLMR95]. (The concept of defect is also related to the notion of the relative degree and zero dynamics in the feedback linearization literature.)

*Flatness versus feedback linearization.*¹ Feedback linearizable systems, in which a system is rendered linear through a (nonlinear) change of state variables and transformation of the input, are a fairly common class of systems. One case that comes up relatively frequently, and is hence worth special mention, is the set of mechanical systems of the form

$$M(q)\ddot{q} + C(q, \dot{q}) = B(q)u.$$

Here $q \in \mathbb{R}^n$ is the configuration of the mechanical system, $M(q) \in \mathbb{R}^{n \times n}$ is the configuration-dependent inertia matrix, $C(q, \dot{q}) \in \mathbb{R}^n$ represents the Coriolis forces and additional nonlinear forces (such as stiffness and friction), and $B(q) \in \mathbb{R}^{n \times p}$ is the input matrix. If $p = n$, then we have the same number of inputs and configuration variables, and if we further have that $B(q)$ is an invertible matrix for all configurations q , then we can choose

$$u = B^{-1}(q)(M(q)v + C(q, \dot{q})). \quad (2.12)$$

The resulting dynamics become

$$M(q)\ddot{q} = M(q)v \quad \implies \quad \ddot{q} = v,$$

which is a linear system. We can now use the tools of linear system theory to analyze and design control laws for the linearized system, remembering to apply equation (2.12) to obtain the actual input that will be applied to the system.

A natural question in considering feedback linearizable systems is whether one should simply feedback linearize the system or whether it is better to instead generate feasible trajectories for the (differentially flat) nonlinear system and then make use of linear controllers to stabilize the system to that trajectory. In many cases it can be advantageous to generate a trajectory for the system (using differentially flatness), where one can take into account constraints on the inputs and the costs associated with state errors and input magnitudes in the original coordinates of the model. A downside of this approach is that the gains of the system must now be modified depending on the operating point (e.g., using gain scheduling), whereas for a system that has been feedback linearized a single linear controller (in the transformed coordinates) can be used.

¹The material in this section is drawn from FBS2e, Section 6.4.

2.4 Implementation in Python²

The Python Control Systems Library (python-control) contains modules that help support trajectory generation using differential flatness and gain-scheduling controller designs.

The `control.flatsys` package contains a set of classes and functions that can be used to compute trajectories for differentially flat systems. It allows both “simple” trajectory generation (no constraints, no cost function) as well as constrained, optimal trajectory generation (with the same basic structure as the optimal control problems described in the next chapter). The primary advantage of solving trajectory generation problems using differentially flat structure, when it applies, is that the all operations are algebraic in nature, with no need to integrate the differential equations describing the dynamics of the system. This can substantially speed up the computation of trajectories.

A differentially flat system is defined by creating an object using the `FlatSystem` class, which has member functions for mapping the system state and input into and out of flat coordinates. The `point_to_point()` function can be used to create a trajectory between two endpoints, written in terms of a set of basis functions defined using the `BasisFamily` class. The resulting trajectory is returned as a `SystemTrajectory` object and can be evaluated using the `eval()` member function.

To create a trajectory for a differentially flat system, a `FlatSystem` object must be created. This is done by specifying the forward and reverse mappings between the system state/input and the differentially flat outputs and their derivatives (“flat flag”).

The `forward()` method computes the flat flag $\bar{z} = (z, \dot{z}, \dots, z^{(q)})$ given a state and input:

```
zflag = sys.forward(x, u)
```

The `reverse()` method computes the state and input given the flat flag:

```
x, u = sys.reverse(zflag)
```

The flag \bar{z} is implemented as a list of flat outputs z_i and their derivatives up to order q_i :

$$\text{zflag}[i][j] = z_i^{(j)}$$

The number of flat outputs must match the number of system inputs.

For a linear system, a flat system representation can be generated using the `LinearFlatSystem` class:

```
sys = control.flatsys.LinearFlatSystem(linsys)
```

For more general systems, the `FlatSystem` object must be created manually:

```
sys = control.flatsys.FlatSystem(forward, reverse, inputs=m, states=n)
```

In addition to the flat system description, a set of basis functions $\phi_i(t)$ must be chosen. The `FlatBasis` class is used to represent the basis functions. A polynomial basis function of the form $1, t, t^2, \dots$ can be computed using the `PolyBasis` class, which is initialized by passing the desired order of the polynomial basis set:

²The material in this section is drawn from [FGM⁺21].

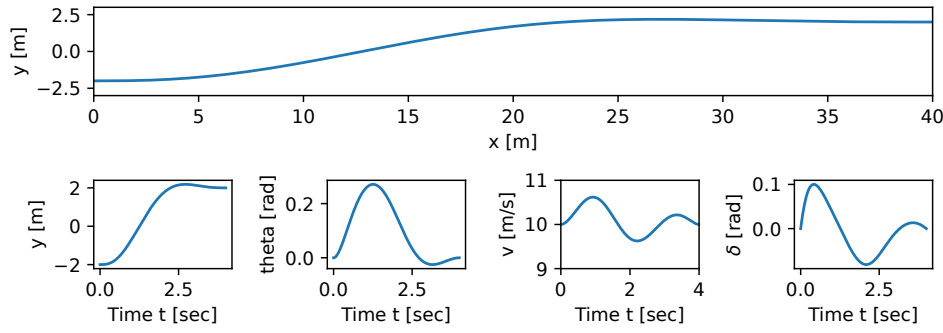


Figure 2.8: Trajectory generation using differential flatness.

```
polybasis = control.flatsys.PolyFamily(N)
```

Once the system and basis function have been defined, the `point_to_point()` function can be used to compute a trajectory between initial and final states and inputs:

```
traj = control.flatsys.point_to_point(
    sys, Tf, x0, u0, xf, uf, basis=polybasis)
```

The returned object has class `SystemTrajectory` and can be used to compute the state and input trajectory between the initial and final condition:

```
xd, ud = traj.eval(timepts)
```

where `timepts` is a list of times on which the trajectory should be evaluated (e.g., `timepts = np.linspace(0, Tf, M)`).

The `point_to_point()` function also allows the specification of a cost function and/or constraints, in the same format as `solve_ocp()`. An example is shown in Figure 2.8, where we have further modified the problem from Example 2.3 by adding constraints on the inputs.

The python-control package also has functions to help simplify the implementation of state feedback-based controllers. The `create_statefbk_iosystem` function can be used to create an I/O system using state feedback, including simple forms of gain scheduling.

A basic state feedback controller of the form

$$u = u_d - K(x - x_d)$$

can be created with the python-control command

```
ctrl, clsys = ct.create_statefbk_iosystem(sys, K)
```

where `sys` is the process dynamics and `K` is the state feedback gain. The function returns the controller `ctrl` and the closed loop systems `clsys`, both as I/O systems. The input to the controller is the vector of desired states x_d , desired inputs u_d , and system states x .

Gain scheduling on the desired state, desired input, or system state can be implemented by setting the gain to a 2-tuple consisting of a list of gains and a list of points at which the gains were computed, as well as a description of the scheduling variables::

```
ctrl, clsys = ct.create_statefbk_iosystem(
    sys, ([g1, ..., gN], [p1, ..., pN]), gainsched_indices=[s1, ..., sq])
```

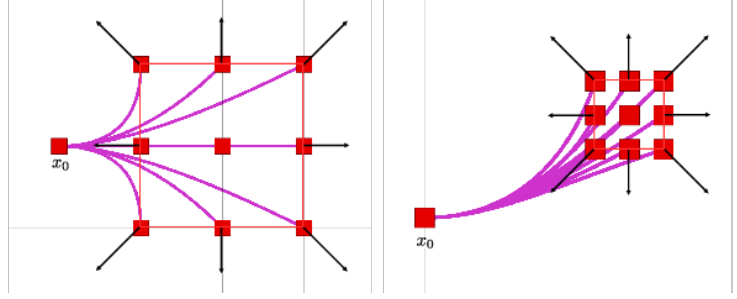


Figure 2.9: Examples of motion primitives for a planar vehicle. Example of 9 planar motion primitives from initial state x_0 for an acceleration-controlled system (left) and a jerk-controlled system (right). The black arrow indicates corresponding control input. The red boundary shows the feasible region for the end states (red squares), which is induced by the control limit u_{\max} . Figure and caption courtesy Liu et al. [LAMK17] (CC-BY).

The list of indices can either be integers indicating the offset into the controller input vector (x_d, u_d, x) or a list of strings matching the names of the input signals. The controller implemented in this case has the form

$$u = u_d - K(\mu)(x - x_d)$$

where μ represents the scheduling variables. See the python-control documentation and `steering-gainsched.py` (in the documentation examples) for more information on the implementation of a gain scheduled controllers.

2.5 Other Methods for Generating Trajectories

In this section we briefly survey some other methods of generating trajectories for nonlinear systems, building on the basic ideas already presented.

Motion primitives and graph search. Rather than solve for an entire trajectory x_d that satisfies the equations of motion and a trajectory goal (e.g., moving from point to point), another common approach to trajectory generation is to create small segments of trajectories that can be concatenated into a longer trajectory. Each segment is called a *motion primitive*.

An example of two sets of motion primitives is shown in Figure 2.9. In the left figure the primitives are generated by using constant acceleration trajectories and in the right figure the primitives are generated by using trajectories that are constant in the third derivative.

Motion primitives can often be combined with other methods for path planning, such as graph search. For example, a grid of target states can be established where points in the grid are connected by motion primitives. This approach creates a graph structure, with each vertex representing a position in the state space and each edge representing a path segment. The problem of trajectory generation then becomes one of graph search: for example, we seek to find a path between two points in the (discretized) state space that minimizes a cost function (represented

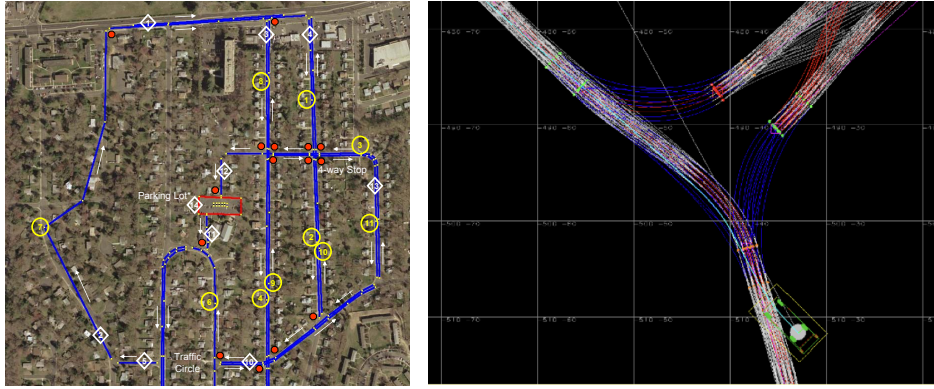


Figure 2.10: Graph-based planning. (a) Road network definition file (RNDF), used for high level route planning. (b) Graph of feasible trajectories at an intersection.

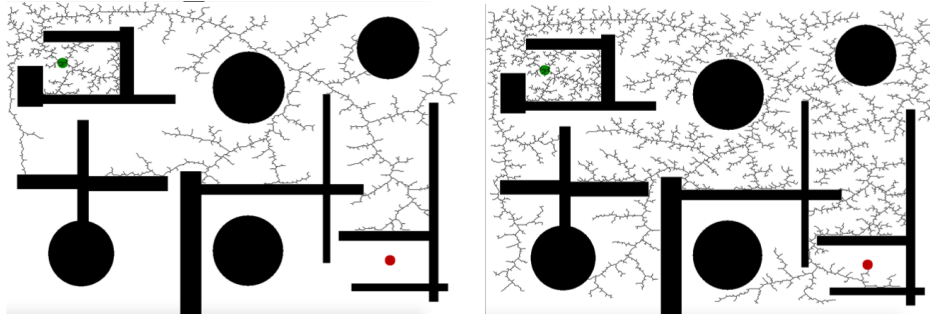


Figure 2.11: Rapidly exploring random tree (RRT) path planning. Random exploration of a 2D search space by randomly sampling points and connecting them to the graph until a feasible path between start and goal is found. Figure and caption courtesy Correll et al. [CHHR22] (CC-BY-ND-NC).

as weights on the nodes and/or edges in the graph). Figure 2.10 illustrates one such approach, used in an autonomous vehicle setting [BdTH⁺07].

Rapidly-exploring random tree (RRT). While graph-based search methods can be very fast, they can become difficult to implement in high dimensional state spaces, since the number of points in the grid scales exponentially in the dimension of the state space. An alternative to “filling” the state space with grid points is to sample feasible trajectories from the primitive set and then “explore” the state space by constructing a tree consisting of concatenated segments.

A popular algorithm for this type of sample-based planning is rapidly-exploring random tree (RRT) search, as illustrated in Figure 2.11. The idea in RRT search is that we start from the initial point in the trajectory and construct a tree of possible trajectories by constantly adding new segments to existing points on the tree. When we add a segment that gets close to the desired final point, we can use the path back to the root to establish a feasible trajectory for the system.

2.6 Further Reading

The two degree of freedom controller structure introduced in this chapter is described in a bit more detail in FBS2e, Chapter 8 (in the context of output feedback control) and a description of some of the origins of this structure are provided in the “Further Reading” section of Chapter 8. Gain scheduling is a classical technique that is often omitted from introductory control texts, but a good description can be found in the survey article by Rugh [Rug90] and the work of Shamma [Sha90]. Differential flatness was originally developed by Fliess, Levine, Martin and Rouchon [FLMR92]. See [Mur97] for a description of the role of flatness in control of mechanical systems and [vNM98, MFHM05] for more information on flatness applied to flight control systems.

Exercises

2.1 (Feasible trajectory for constant reference). Consider a linear input/output system of the form

$$\dot{x} = Ax + Bu, \quad y = Cx \quad (2.13)$$

in which we wish to track a constant reference r . A feasible (steady state) trajectory for the system is given by solving the equation

$$\begin{bmatrix} 0 \\ r \end{bmatrix} = \begin{bmatrix} A & B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_d \\ u_d \end{bmatrix}$$

for x_d and u_d .

(a) Show that these equations always have a solution as long as the linear system (2.13) is reachable.

(b) In Section 7.2 of FBS2e we showed that the reference tracking problem could be solved using a control law of the form $u = -Kx + k_r r$. Show that this is equivalent to a two degree of freedom control design using x_d and u_d and give a formula for k_r in terms of x_d and u_d . Show that this formula matches that given in FBS2e.

2.2. A simplified model of the steering control problem is described in FBS2e, Example 6.13. The lateral dynamics can be approximated by the linearized dynamics

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} \gamma \\ 1 \end{bmatrix} u, \quad y = x_1,$$

where $x = (y, \theta) \in \mathbb{R}^2$ is the state of the system, γ is a parameter that depends on the forward speed of the vehicle, and y is the lateral position of the vehicle.

Suppose that we wish to track a piecewise constant reference trajectory that consists of moving left and right by 1 meter:

$$x_d = \begin{bmatrix} \text{square}(2\pi t/20) \\ 0 \end{bmatrix}, \quad u_d = 0,$$

where `square` is the square wave function in `scipy.signal`. Suppose further that the speed of the vehicle varies such that the parameter γ satisfies the formula

$$\gamma(t) = 2 + 2\sin(2\pi t/50).$$

(a) Show that the desired trajectory given by x_d and u_d satisfy the dynamics of the system at all points in time except the switching points of the square wave function.

(b) Suppose that we fix $\gamma = 2$. Use eigenvalue placement to design a state space controller $u = u_d - K(x - x_d)$ where the gain matrix K is chosen such that the eigenvalues of the closed loop poles are at the roots of $s^2 + 2\zeta\omega_0 s + \omega_0^2$, where $\zeta = 0.7$ and $\omega_0 = 1$. Apply the controller to the time-varying system where $\gamma(t)$ is allowed to vary and plot the output of the system compared to the desired output.

(c) Find gain matrices K_1 , K_2 , and K_3 corresponding to $\gamma = 0, 2, 4$ and design a gain-scheduled controller that uses linear interpolation to compute the gain for values of γ between these values. Compare the performance of the gain scheduled controller to your controller from part (b).

2.3. Solve Example 2.2 using python-control.

2.4 (Stability of gain scheduled controllers for slowly varying systems). Consider a nonlinear control system with gain scheduled feedback

$$\dot{e} = f(e, v) \quad v = k(\mu)e,$$

where $\mu(t) \in \mathbb{R}$ is an externally specified parameter (e.g., the desired trajectory) and $k(\mu)$ is chosen such that the linearization of the closed loop system around the origin is stable for each fixed μ .

Show that if $|\dot{\mu}|$ is sufficiently small then the equilibrium point is locally asymptotically stable for the full nonlinear, time-varying system. (Hint: find a Lyapunov function of the form $V = x^T P(\mu)x$ based on the linearization of the system dynamics for fixed μ and then show this is a Lyapunov function for the full system.)

2.5 (Flatness of systems in reachable canonical form). Consider a single input system in reachable canonical form [FBS2e, Section 7.1]:

$$\begin{aligned} \frac{dx}{dt} &= \begin{bmatrix} -a_1 & -a_2 & -a_3 & \dots & -a_n \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & & 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u, \\ y &= [b_1 \quad b_2 \quad b_3 \quad \dots \quad b_n] x + du. \end{aligned} \quad (2.14)$$

Suppose that we wish to find an input u that moves the system from x_0 to x_f . This system is differentially flat with flat output given by $z = x_n$ and hence we can parameterize the solutions by a curve of the form

$$x_n(t) = \sum_{k=0}^N \alpha_k t^k, \quad (2.15)$$

where N is a sufficiently large integer.

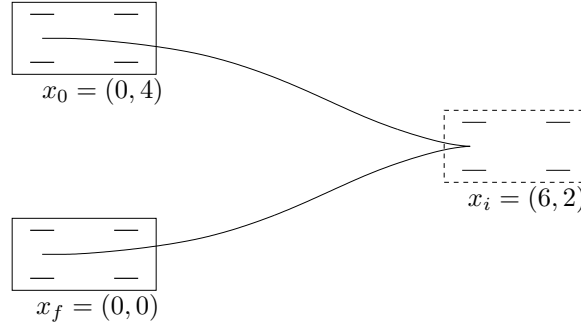
(a) Compute the state space trajectory $x(t)$ and input $u(t)$ corresponding to equation (2.15) and satisfying the differential equation (2.14). Your answer should be an equation similar to equation (2.7) for each state x_i and the input u .

(b) Find an explicit input that steers a double integrator system between any two equilibrium points $x_0 \in \mathbb{R}^2$ and $x_f \in \mathbb{R}^2$.

(c) Show that all reachable systems are differentially flat and give a formula for finding the flat output in terms of the dynamics matrix A and control matrix B .

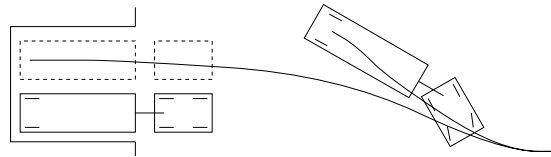
2.6. Show that the servomechanism example from Exercise 1.3 is differentially flat and compute the forward and reverse mappings between the system states and inputs and the flat flag.

2.7. Consider the lateral control problem for an autonomous ground vehicle as described in Example 2.1 and Section 2.3. Using the fact that the system is differentially flat, find an explicit trajectory that solves the following parallel parking maneuver:



Your solution should consist of two segments: a curve from x_0 to x_i with $v > 0$ and a curve from x_i to x_f with $v < 0$. For the trajectory that you determine, plot the trajectory in the plane (x versus y) and also the inputs v and ϕ as a function of time.

2.8. Consider first the problem of controlling a truck with trailer, as shown in the figure below:



$$\begin{aligned}\dot{x} &= \cos \theta u_1 \\ \dot{y} &= \sin \theta u_1 \\ \dot{\phi} &= u_2 \\ \dot{\theta} &= \frac{1}{l} \tan \phi u_1 \\ \dot{\theta}_1 &= \frac{1}{d} \cos(\theta - \theta_1) \sin(\theta - \theta_1) u_1,\end{aligned}$$

The dynamics are given above, where (x, y, θ) is the position and orientation of the truck, ϕ is the angle of the steering wheels, θ_1 is the angle of the trailer, and l and d are the length of the truck and trailer. We want to generate a trajectory for the truck to move it from a given initial position to the loading dock. We ignore the role of obstacles and concentrate on generation of feasible trajectories.

- (a) Show that the system is differentially flat using the center of the rear wheels of the trailer as the flat output.
- (b) Generate a trajectory for the system that steers the vehicle from an initial condition with the truck and trailer perpendicular to the loading dock into the loading dock.
- (c) Write a simulation of the system stabilizes the desired trajectory and demonstrate your two-degree of freedom control system maneuvering from several different initial conditions into the parking space, with either disturbances or modeling errors included in the simulation.

