

Chapter 5

Dynamical Systems

Controllers such as a thermostat for regulating the temperature in a room or a cruise controller to track the speed of an automobile, are interacting with the physical world via sensors and actuators. The physical quantities involved such as temperature, pressure, and speed evolve continuously obeying laws of physics. Design and analysis of control systems, as a result, requires construction of models of the physical system. In this chapter, we focus on continuous-time models of dynamical systems. This is a mathematically rich area that is explored in details in a course on control systems. The purpose of this chapter is to give a brief introduction to the core concepts.

5.1 Continuous-time Models

The typical set-up of a control system is depicted in Figure 5.1. The physical world that is to be controlled is modeled by a component called *plant*. The evolution of the plant can be influenced by the controller using actuators, and the controller can base its decisions based on measurements provided by the sensors. For example, in a thermostat design, the plant is the house whose temperature is to be controlled. The sensor is a thermometer that can measure the current temperature (upto certain accuracy). The task of the controller is to regulate the temperature so that it stays “close” to the temperature set by the occupant on the thermostat. The controller can influence the temperature by adjusting the heat flow from the furnace. The plant model, in this case, needs to capture how the temperature evolves as a function of the heat-flow from the furnace and the difference between the house temperature and outside temperature (which may cause heat loss).

Modeling of dynamical systems also relies heavily on components with inputs and outputs that are connected to one another using block diagrams. The underlying model of computation is *synchronous* as in Chapter 1 with one key difference: the values of variables are updated continuously as opposed to in

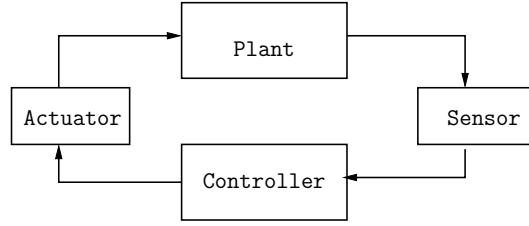


Figure 5.1: Information flow in a control system

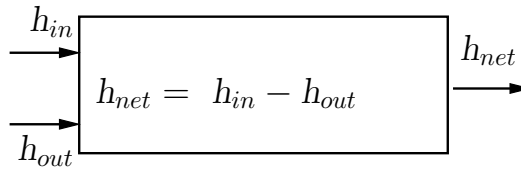


Figure 5.2: Continuous-time component NetHeat

a sequence of discrete rounds. We call such components *continuous-time components*. The variables of a continuous-time component typically range over a compact set such as an interval of the set of reals with specified units of measurement. For example, the velocity v of the car can range over the interval $[0, 150]$ miles-per-hour. For analysis purpose, we can assume every variable has the type `real`. The values for the variables are functions from the time domain. Such functions from the time domain to the set of reals are called *signals*. Throughout we will assume that the time domain consists of the set of non-negative real numbers, and denote this set by `time`. For a variable x , we will use ρ_x to denote a signal that assigns values to the variable x : for time t , $\rho_x(t)$ denotes the value of the variable x at time t . Given a set V of variables, we use ρ_V to denote a signal that assigns values to each of the variables in V as a function of time. If V contains k variables, ρ_V is a function from `time` to k -tuples over `real`.

As a first example, Figure 5.2 shows a continuous-time component `NetHeat` that has two inputs h_{in} and h_{out} , and an output h_{net} , which denote the heat inflow, heat outflow, and net heatflow. The component `NetHeat` is a mapping from two input signals to an output signal, and its dynamics is expressed by the expression

$$h_{net} = h_{in} - h_{out},$$

which says that at every time t , the value of h_{net} equals the expression $h_{in} - h_{out}$. Given input signals $\rho_{h_{in}}$ and $\rho_{h_{out}}$, the output signal $\rho_{h_{net}}$ is defined by $\rho_{h_{net}}(t) = \rho_{h_{in}}(t) - \rho_{h_{out}}(t)$ for all times $t \in \text{time}$. This unique output signal is called the *response* of the component to input signals $\rho_{h_{in}}$ and $\rho_{h_{out}}$.

The component `NetHeat` is stateless. As an example of a stateful continuous-

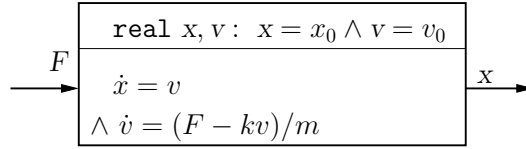


Figure 5.3: Continuous-time component modeling car motion

time component, let us build a model of how the speed of a car changes as a result of the force applied to it by the engine. For the purpose of designing a cruise controller, it typically suffices to make a number of simplifying assumptions. In particular, let us assume that the rotational inertia of the wheels is negligible and that the friction resisting the motion is proportional to the car's speed. If x denotes the position of the car (measured with respect to an inertial reference), and F denotes the force applied to the car, then using the classical Newtonian model for motion, we can capture the dynamics of the car by the equation:

$$F - k \dot{x} = m \ddot{x}.$$

Here k is the coefficient of the frictional force, and m denotes the mass of the car. The quantity \dot{x} denotes the first-order time derivative of the signal x , and thus captures the velocity of the car, and \ddot{x} denotes the second-order derivative of x , that is, the acceleration of the car. This equation of motion is modeled by the continuous-time component **Car** of Figure 5.3. It uses two state variables: x modeling the position of the car and v modeling the velocity of the car. For every state variable, the component needs to specify its initial value. For every state variable, the dynamics is given by specifying the first-order time-derivative of the value of the state variable as a function of the inputs and state. The output of the car is its position. For every output variable, the component specifies the value of the output as a function of the inputs and state; in this example the output simply equals one of the state variables.

In general, the dynamics of the component is specified by a real-valued expression h_y for every output variable y , and a real-valued expression f_x for every state variable x . Each of these expressions is an expression over the input and state variables. The value of the output variable y at time t is obtained by evaluating the expression h_y using values of state and input variables at time t , and the signal for a state variable x should be such that its rate of change at time t should equal the expression f_x evaluated using values of state and input variables at time t . For given input signals, the signals for state and output variables are uniquely determined. Thus, the execution of a continuous-time component is similar to the execution of a deterministic synchronous component, except the notion of a round is now infinitesimal: at every t , the outputs at time t are determined as a function of the inputs at time t and the state of the component at time t , and then the state is updated using the rate of change specified by the derivative evaluated using the inputs and state at time t .

The definition of a continuous-time component is summarized below:

CONTINUOUS-TIME COMPONENT

A continuous-time component C has a finite set I of real-valued input variables, a finite set O of real-valued output variables, a finite set S of real-valued state variables, such that these three sets are pair-wise disjoint; an initial value $x_0 \in \mathbf{real}$ for every state variable $x \in S$; a real-valued expression h_y over $I \cup S$ for every output variable $y \in O$, and a real-valued expression f_x over $I \cup S$ for every state variable $x \in S$. Given an input signal ρ_I , the corresponding execution of the component C is the state signal ρ_S and the output signal ρ_O such that (1) for every state variable x , $\rho_x(0) = x_0$; (2) for every output variable y and time t , $\rho_y(t) = [\rho_I(t), \rho_S(t)](h_y)$; and (3) for every state variable x and time t , $d/dt \rho_x(t) = [\rho_I(t), \rho_S(t)](f_x)$.

Integral Models

As another example, let us consider the classical problem of controlling a helicopter so as to keep it from spinning. A helicopter has 6 degrees of motion, 3 for position and 3 for rotation. In our simplified version, let us assume that the helicopter position is fixed and it remains vertical. The only freedom of motion is the angular rotation around the Y-axis, which is in the horizontal plane perpendicular to the body axis. This rotation is called *yaw*. The friction of the main rotor (at the top of the helicopter) causes the yaw to change. The tail rotor then needs to apply a torque to counteract this rotational force to keep the helicopter from spinning. In this setting, the helicopter model has a continuous-time input signal T , denoting the torque around the Y-axis. The moment of inertia of the helicopter in this simplified setting can be modeled by a single scalar I . The output signal of the model is the angular velocity around the vertical axis, and is modeled by the spin $s = \dot{\theta}$, where θ gives the yaw. The equation of motion is given by

$$\ddot{\theta} = T/I$$

The corresponding continuous-time component is shown in Figure 5.4. The dynamics can alternatively be expressed using an integral equation: the value of the spin at time t equals the sum of its initial value and the integral of the torque upto time t .

$$s(t) = s_0 + (1/I) \int_0^t F(\tau) d\tau.$$

Modeling tools such as Simulink allow integrators as components. Note that the internal state is implicit in the integral model. The models expressed as continuous-time components with state variables and differential equations are sometimes called the *state-space representation* of dynamical systems.

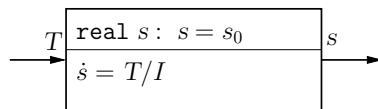


Figure 5.4: Continuous-time component modeling helicopter spin

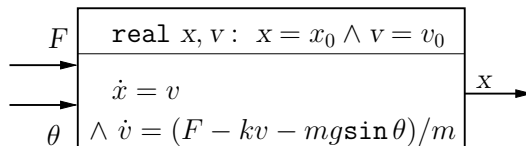


Figure 5.5: Continuous-time component modeling car motion on a graded road

Adding Noise

Let us revisit the model of the motion of a car. This model assumes that the car is moving in a single dimension on a flat road. Suppose we now want to account for the *grade* of the road: on an up-hill, the weight of the car works against the force applied by the engine, and on a down-hill, the weight of the car adds to this force. The cruise controller needs to adjust the force to keep the net velocity in the direction along the road constant. We can model the grade of the road by an additional input, θ , that captures the angle of the road with the horizontal (positive angle indicates an up-hill slope and negative angle indicates a down-hill slope). The weight of the car equals mg in the vertically downwards direction, where $g = 9.8 \text{ m/s}^2$ is the gravitational acceleration. The modified dynamical system is shown in Figure 5.5. The forces acting on the car in the direction along the road are: F in the forward direction controlled by the engine, kv in the backward direction modeling the friction, and $mg \sin \theta$ capturing the gravitational force along the road in the backward direction. The control design problem for the revised model is different in a crucial way: the input signal θ modeling the grade of the road is not controlled by the controller and is not known in advance. The controller should work for a reasonable range of θ values (for instance, all values in the range $[-30, +30]$).

Linear Systems

Consider a continuous-time component C with an input variable x and an output variable y . Given a input signal $\rho_x : \text{time} \mapsto \text{real}$, there is a unique output signal $\rho_y : \text{time} \mapsto \text{real}$ corresponding to the execution of C corresponding to the input signal. Thus, a continuous-time component is a function from input signals to output signals. If this function is linear, the corresponding dynamical system is called *linear*. Linearity means the following two properties:

- If the input signal is scaled by a constant factor, then the output signal also gets scaled by the same factor. Given an input signal ρ_x and a constant α , let $\alpha\rho_x$ be the input signal whose value at time t is $\alpha\rho_x(t)$. Then, for a linear component C , if ρ_y is the output signal corresponding to ρ_x , then $\alpha\rho_y$ is the output signal corresponding to $\alpha\rho_x$.
- If an input signal can be expressed as a sum of two input signals, then the corresponding output signal can be obtained by adding the output signals corresponding to the component input signals. That is, if $\rho_x(t) = \rho_{x_1}(t) + \rho_{x_2}(t)$ for all time t , and $\rho_y, \rho_{y_1}, \rho_{y_2}$ are the output signals corresponding to the input signals ρ_x, ρ_{x_1} , and ρ_{x_2} , then $\rho_y(t) = \rho_{y_1}(t) + \rho_{y_2}(t)$ for all times t .

In general, the component has multiple inputs and multiple outputs, and we need to consider signals that are mappings from the time domain to real-valued vectors. Linearity is defined by considering sum of vectors and scaling of vectors.

LINEAR COMPONENT

A continuous-time component C with input variables I and output variables O is called *linear*, if for all input signals ρ_{I_1} and ρ_{I_2} , and constants $\alpha, \beta \in \text{real}$, if the output signals generated by C in response to the input signals ρ_{I_1} and ρ_{I_2} are ρ_{O_1} and ρ_{O_2} , respectively, then the output signal generated by C in response to the input signal $\alpha\rho_{I_1} + \beta\rho_{I_2}$ is $\alpha\rho_{O_1} + \beta\rho_{O_2}$.

In our examples, the components of Figure 5.2, Figure 5.3, and Figure 5.4 are linear, but the component of Figure 5.5 with the non-linear term $mg \sin \theta$ in the dynamics is not linear. However, notice that the input θ is not controlled, and simply represents disturbance or noise that the controller must handle. As a result we can replace the input θ by another variable d with the meaning that d captures the value of $mg \sin \theta$. Now the dynamics becomes $\dot{v} = (F - kv - d)/m$, and is linear. The range for input disturbance $\theta \in [-30, +30]$ must be replaced by $d \in [mg \sin(-30), mg \sin(+30)]$.

The standard form for expressing the dynamics for linear systems uses matrices. Consider a linear component with m input variables $I = \{u_1, \dots, u_m\}$, n state variables $S = \{x_1, \dots, x_n\}$, and k output variables $O = \{y_1, \dots, y_k\}$. In this case, we can view the input as a vector of dimension $m \times 1$, state as a vector of dimension $n \times 1$, and output as a vector of dimension $k \times 1$. The dynamics is expressed by four matrices each with real-valued coefficients: matrix A of dimension $n \times n$, matrix B of dimension $n \times m$, matrix C of dimension $k \times n$, and matrix D of dimension $k \times m$. The dynamics is given by

$$\dot{S} = AS + BI, \quad O = CS + DI.$$

That is, for each state variable x_i , the differential equation modeling its rate of change is given by

$$\dot{x}_i = A_{i1}x_1 + \dots + A_{in}x_n + B_{i1}u_1 + \dots + B_{im}u_m$$

For each output variable y_j , its value is defined in terms of state and input variables by the linear expression:

$$y_j = C_{i1}x_1 + \cdots + C_{in}x_n + D_{i1}u_1 + \cdots + D_{im}u_m$$

In our example of the model of the car of Figure 5.3, $m = 2$ and $n = k = 1$. The matrices are given by:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -k/m \end{bmatrix}; \quad B = [0 \quad 1/m]; \quad C = [1 \quad 0]; \quad D = [0]$$

For linear systems, a number of analysis techniques are available to understand how the output signal is related to the input signal. In particular, consider the dynamics $\dot{S} = AS + BI$. Suppose the initial state is given by the vector x_0 . Given an input signal ρ_I , the resulting state signal is given by the equation:

$$\rho_S(t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)} B \rho_I(\tau) d\tau.$$

The response of the system to a given input signal can be computed using this equation.

Composing Components

Continuous-time components can be composed using block diagrams in a way similar to synchronous components. We can define the operations of instantiation, output hiding, and parallel composition in the same way. To ensure determinism and well-formed composition, while composing components, a sufficient condition is absence of cyclic awaits dependencies. The awaits-dependency of an output variable on input variables is defined the same way as in synchronous components. For a continuous-time component, an output variable y awaits a subset J of input variables, if the expression h_y is over the variables $S \cup J$, and thus, value of output y at time t depends on state at t and values of input variables in J at time t .

Stability

Intuitively, a stable system is one whose output signal is of small magnitude when the input signal is small in magnitude. More specifically, a signal ρ is bounded if there exists a constant Δ such that $|\rho(t)| \leq \Delta$ for all time t . For instance, the linearly increasing signal defined by $\rho(t) = a + bt$, for constants a and $b > 0$, is not bounded; the exponentially increasing signal defined by $\rho(t) = a + e^{bt}$, for constants a and $b > 0$, is not bounded; the exponentially decaying signal defined by $\rho(t) = a + e^{-bt}$, for constants a and $b > 0$, is bounded; the step-signal defined by $\rho(t) = a$ for $t < t_0$ and $\rho(t) = b$ for $t \geq t_0$, for constants t_0, a, b , is bounded; and the sinusoidal signal defined by $\rho(t) = a \cos bt$, for constants a, b , is bounded.

In a stable system, whenever the input signal is bounded then so is the output signal produced by the component in response. The bound on the output signal can be different from the bound on the input signal. Stability is a desirable feature for dynamical systems.

STABLE SYSTEM

A continuous component C with input variables I and output variables O is *Bounded-Input-Bounded-Output stable* if for every bounded input signal ρ_I , the output signal ρ_O produced by C in response to the input ρ_I , is also bounded.

Let us consider the model of the car from Figure 5.4:

$$\dot{s} = T/I$$

This system is unstable. Suppose we apply a constant torque T_0 to the system, then the rate of change of the spin s is constant. The spin will keep increasing linearly: the output signal is described by $\rho_s(t) = s_0 + (T_0/I)t$.

For linear systems, stability can be analyzed using standard mathematical tools. Consider the system given by $\dot{S} = AS + BI$ where the initial value for the state variables is given by the vector x_0 . Stability of such a system can be analyzed by studying its dynamics when the input signal is always 0. The system is stable if, irrespective of the initial state, the state of the system goes to 0 as time advances: if for every initial state $x_0 \in \mathbf{time}^n$, if ρ_S is the state response of the system from the initial state x_0 to the zero input signal $\rho_I = 0$, then $\rho_S(t) \rightarrow 0$ as $t \rightarrow \infty$.

Note that for the helicopter model, if we set the input torque to 0, dynamics is given by $\dot{s} = 0$. In this dynamics, if the initial spin is s_0 , it will stay at s_0 , and thus system is unstable.

Consider a system with one state variable x and one input variable u , and suppose the dynamics is given by the linear equation

$$\dot{x} = ax + bu$$

When the input signal for u is 0, the dynamics becomes $\dot{x} = ax$. If the coefficient a is negative, then no matter what the initial value of x is, x decays exponentially, and will become 0 in the limit, and the system is stable. If the coefficient a is 0, then x stays equal to its initial value, and the system is unstable. If the coefficient a is positive, then x increases exponentially and grows in an unbounded manner; the system is unstable. Thus, the stability of one-dimensional linear system depends on the sign of the coefficient of the term capturing dependence of the rate of change on the state. In the general case of linear systems, the eigenvalues of the matrix A in the dynamics can be examined to determine the stability. An eigenvalue of a matrix is, in general, a complex number. Stability requires that the real part of such an eigenvalue should be

negative. If the matrix A is a diagonal matrix (all entries A_{ij} with $i \neq j$, are 0s) then the eigenvalues are simply the coefficients A_{ii} along the diagonal, and they all should be negative.

Theorem 5.1 [Stability Test for Linear Dynamics] *A linear system with state dynamics $\dot{S} = AS + BI$ is stable if the real component of every eigenvalue of the matrix A is negative.*

5.2 Designing Controllers

Given a dynamical system model of the plant, the controller is designed to provide the controlled input signals to maintain the output of the system close to the desired output irrespective of changes in uncontrolled input signals corresponding to disturbances. Designing controllers in a principled manner is a well-developed discipline. Let us review some basic terminology in control design, and get familiar with the most commonly used class of controllers in industrial practice.

Open-loop vs. Feedback Controller

An open-loop controller does not use measurements of the state or outputs of the plant to make its decisions. Such a controller relies on the model of the plant to decide on the controlled input for the plant, and its implementation does not require sensors. For example, consider the first model of the car from Figure 5.3. Suppose the controller's objective is to maintain constant velocity (that is, $\dot{v} = 0$). Then, if v_0 is the initial velocity, the desired value of force F equals kv_0 : the controller can simply apply this constant force to the car to maintain the velocity at v_0 . Such a controller is called open-loop: when compared to the set-up shown in Figure 5.1, the block for sensors and flow of information from the plant to the controller is missing. Obviously, such a controller would be less expensive. However, it is assuming that the behavior of the plant is entirely predictable. In practice, operation of such a controller is acceptable with manual intervention. If the driver finds the speed of the car unacceptable, she would simply increment or decrement the desired speed triggering recalculation by the open-loop controller.

A feedback controller uses sensors to measure the output, and thus, indirectly the current state of the plant, to change the controlled input. For example, in the revised model of the car in Figure 5.5, the model accounts for the change in the grade of the road. Suppose that the controller is applying the correct amount of force to maintain the velocity of the car at desired cruising speed. A positive change in θ causes the car to slow down, while a negative change in θ causes the car to speed up. The speed of the car, as measured by the actuators, is an input to the controller. It notices the change in speed and adjusts the force to make the velocity again equal to the desired cruising speed. Feedback controller not only can cope with disturbances (such as the grade) whose variation with time

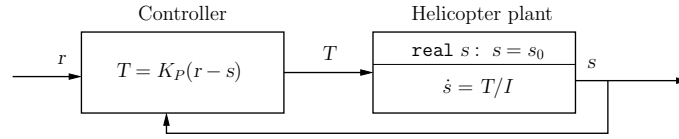


Figure 5.6: Stabilizing controller for the helicopter model

is not predictable in advance, but can work well even when the mathematical model of the plant is only a rough approximation of the real-world dynamics. Implementation of a feedback controller requires sensors, and its performance is related to the accuracy of measurements by sensors.

Stabilizing Controller

We now describe a simple and typical pattern for designing a controller using the helicopter example. The controller is shown in Figure 5.6. It takes two input signals: the input r represents the *reference signal* that captures the desired spin, and the signal s is the (measured) spin of the helicopter. Given two such inputs corresponding to the desired and actual values, we can define the *error signal* e by the equation $e = r - s$. The goal of the controller is to keep the magnitude of the error as small as possible, and also ensure that the closed-loop system obtained by composing the helicopter model and the controller is stable. Note that a positive value of e means that the controller should try to increase the actual spin s by applying a positive torque, and a negative value of e means that the controller should try to decrease the actual spin s by applying a negative torque. The controller of Figure 5.6 computes the torque by simply scaling the error signal by a positive constant factor K_P . Such a controller is called a *proportional controller* and the constant K_P is called the *gain* of the controller.

For the closed-loop system consisting of the composition of the controller and the helicopter, the input signal is the reference r and the output is the spin s . The dynamics is given by the equation

$$\dot{s} = K_P(r - s)/I.$$

This is a one-dimensional linear system, and is stable as long as the gain K_P is positive (note that the coefficient capturing the dependence of the rate of change of s on s is $-K_P/I$).

When the reference input is 0, that is, when the objective of the controller is to keep the helicopter from spinning, the controller applies the torque equal to $-K_P s/I$. No matter what the initial spin s_0 is, this causes the spin to decay to 0 exponentially. Higher the value of K_P , faster is the rate of convergence.

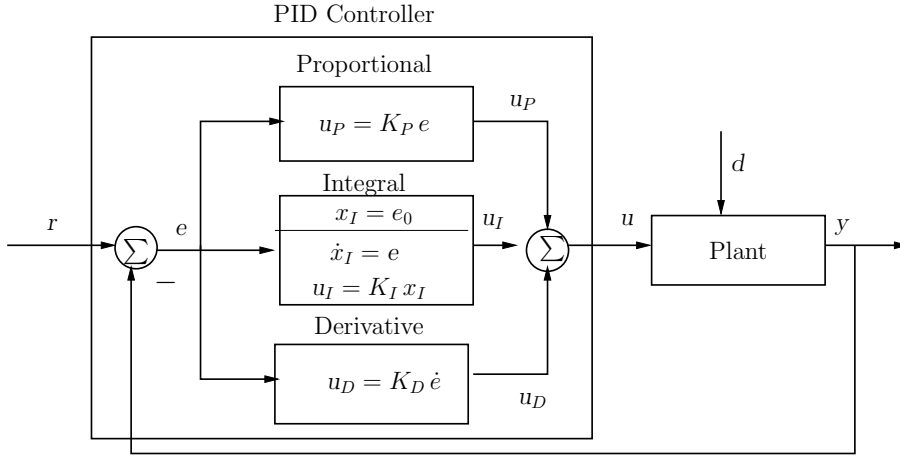


Figure 5.7: A generic PID controller

PID Controllers

In industrial control systems, the most commonly used design of a controller to correct the discrepancy between the desired reference signal and the measured output signal uses combination of three terms: a *proportional* term capturing reaction to the current error, an *integral* term capturing reaction to the cumulative error, and a *derivative* term capturing the response to the rate of change of error. Such a controller is called a PID controller.

The controller is shown in Figure 5.7. The controller takes two signals as inputs: the reference signal r and the measured output y of the plant. Note that r and y can be vectors. For example, in the general setting for controlling the motion of a helicopter, the reference signal will include 3-dimensional motion plan from the source to the destination, along with constraints on the angular rotations. Let e denote the *error signal* capturing the difference between the reference signal r and the measured plant output y . Then, the controller’s output u , which is fed to the plant, is the sum of three terms.

- Proportional term $K_P e$: The contribution of this term is directly proportional to the current error. The constant K_P is called the proportional gain, and the controller scales the error by this factor.
- Integral term $K_I \int_0^t e(\tau) d\tau$: Note that the integral of the error signal upto time t gives the cumulative error upto time t , and thus, the contribution of this term accounts for the cumulative error so far. The constant K_I is called the integral gain, and the controller scales the accumulated error by this factor.
- Derivative term $K_D \dot{e}$: The contribution of this term is correlated to the

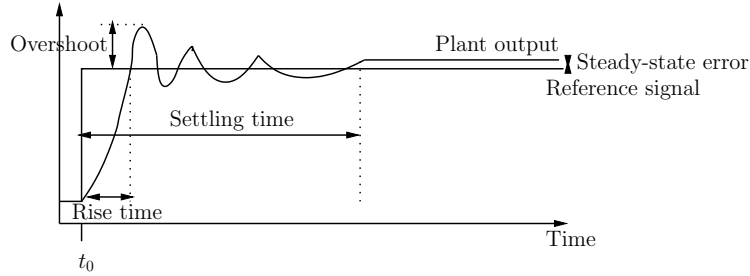


Figure 5.8: Typical output response to a step reference signal

rate at which the error changes. The constant K_D is called the derivative gain, and the controller scales the rate of change of the error by this factor.

Note that the proportional component of the PID controller is stateless. The integral component maintains a state variable x_I that captures the accumulated error, and the rate of change of this state variable equals the error e . The derivative component has a state variable x_D that stores the error e , and the output of the derivative component is the first-order derivative of this state variable. In Figure 5.7, the Sum block simply outputs the sum of its input signals, where the negative sign on an input signal indicates that the corresponding input should be subtracted. Some of the components may be missing in a specific control design. For instance, a P controller has only the proportional block, and a PI controller has only the proportional and the integral blocks. Both of these are also common in practice.

To understand how the control performance changes with the contributions of various terms, let us consider how the output of a linear plant changes in response to a step signal given as a reference signal. Figure 5.8 shows a typical change in the output signal y . At the time instance t_0 when the reference signal jumps from the initial value to the set-point, the error e is highest. As e changes, so does the output of the controller, which impacts the state and output of the plant. The output approaches 1, but overshoots the desired level. The smoothness of the dynamics of the physical world makes such an overshoot unavoidable. The overshoot makes the error negative causing the controller to change the direction of its output. The same phenomenon repeats causing the output to oscillate for a while before it settles into a steady-state value (with no change in absence of external disturbances). For such a response in the plant output, the following metrics capture the performance of the controller:

1. **Overshoot:** The difference between the maximum value of the plant output and the desired reference value. Ideally, the overshoot should be as small as possible. In particular, a safety requirement can assert that the overshoot should be below some threshold value.

2. Rise time: The time difference between the instance t_0 when the reference signal changes and the time at which the output signal crosses the desired reference value. Ideally, smaller rise time means better responsiveness of the system, and typically, an attempt to reduce the rise time will increase the overshoot.
3. Steady state error: The difference between the steady-state value of the output signal and the set-point value of the reference signal. Ideally, steady-state error should be 0, but a small error may be acceptable.
4. Settling time: The time difference between the instance t_0 when the reference signal changes and the time at which the output signal reaches its steady-state value. Ideally settling time should also be small, and the system should reach the desired output value with few oscillations.

The values of the three parameters K_P , K_I , and K_D of the PID controller are adjusted so that the controller's performance is acceptable on all the above metrics. Higher values of the proportional gain K_P mean that the rise time will be smaller, but with higher overshoot. Too high a value of K_P can cause large oscillations delaying settling time. A purely proportional controller also has a steady-state error. The effect of the integral term gets rid of the steady-state error. Higher values of the integral gain K_I improves responsiveness, but also contributes to overshoot. The overshoot is reduced by using the derivative component. Higher values of the derivative gain K_D contribute to reducing the overshoot and the settling time. A number of tools are available for automatic tuning of these three parameters for desired performance.

5.3 Analysis Techniques

Given a model of a continuous-time component, which may include the plant model, the feedback controller, and constraints on initial state and disturbances, we want to understand the behavior of the system. We first discuss the traditional approach based on numerical simulation, and then some emerging techniques for verifying safety requirements.

5.3.1 Numerical Simulation

Consider a continuous-time component whose state is captured by the variable x which may be a vector, and whose input is described by a variable u which also can be a vector. The function f gives the rate of change of x as a function of state x and input u . Given an input signal ρ_u and an initial state x_0 , the evolution of the state of the system, then, can be computed by solving the differential equation $\dot{x} = f(x, u)$ with $\rho_x(0) = x_0$. While for some specific forms of the function f , a closed-form solution for the state $\rho_x(t)$ at time t can be computed, the general method for computing this signal relies on numerical simulation. For numerical simulation, the user provides a discretization step

parameter Δ , and the simulator attempts to compute the values of the state at times $\Delta, 2\Delta, 3\Delta, \dots$ that closely approximate the values of the desired response signal ρ_x as closely as possible.

Euler's method

Euler's method relies on the observation that the rate of change of x , that is, dx/dt , at time t , is simply the limit of the quantity $(x(t + \Delta) - x(t))/\Delta$ as Δ goes to 0. As a result, for small values of Δ , it is natural to approximate the value of $x(t + \Delta)$ by the following equation

$$x(t + \Delta) = x(t) + \Delta f(x, u)(t).$$

This approximation assumes that the rate of change of x is constant during the interval $[t, t + \Delta)$ and equals the rate of change at the beginning of interval. The rate of change at the beginning of interval is obtained by evaluating the function f using the values of x and u at time t . The change in the value of x is obtained by multiplying this rate by the size Δ of the interval. Thus, given an initial value x_0 for the state, and a sequence of values u_0, u_1, \dots for the input, the Euler's method for simulation of differential equation $\dot{x} = f(x, u)$ computes the values, for every $i \geq 0$,

$$x_{i+1} = x_i + \Delta f(x_i, u_i).$$

This sequence of values is linearly-extrapolated to give the response signal that defines the state $\rho_x(t)$ at every time $t \in \mathbf{time}$: for every $i \geq 0$, and time $t \in [i\Delta, (i + 1)\Delta)$, $\rho_x(t) = x_i + (t - i\Delta) f(x_i, u_i)$.

Runge-Kutta methods

Euler's method estimates the state at the end of interval by assuming that the rate of change of state stays constant, and this rate is based only on the state at the beginning of the interval. A better approximation can be obtained if estimated change in the state is used to estimate a change in the derivative, and use this to readjust the state estimate. Runge-Kutta methods comprise a popular class of numerical integration methods based on this idea. In particular, the *second-order Runge-Kutta method* computes the state x_{i+1} by the following calculation:

$$\begin{aligned} k_1 &= f(x_i, u_i) \\ k_2 &= f(x_i + \Delta k_1, u_{i+1}) \\ x_{i+1} &= x_i + \Delta(k_1 + k_2)/2 \end{aligned}$$

Given the current state x_i and input u_i , the first step computes k_1 to be the current rate of change. However, instead of setting the state x_{i+1} at the end of the interval to be $x_i + \Delta k_1$ as in Euler's method, it uses this estimated state to calculate the estimated rate of change k_2 at the end of the interval (the

new input value u_{i+1} is used for this estimate). The third step calculates x_{i+1} assuming that the rate of change is the average of k_1 and k_2 .

The higher-order Runge-Kutta methods use the same basic idea, but use estimates of derivatives at the midpoint as well as at the end-points to compute a weighted average. The most commonly used method in practice is the Fourth-order Runge-Kutta method. The quality of approximation afforded by numerical simulation depends on the step size Δ . Most simulation tools automatically tune this parameter to keep the error small. In particular, adaptive techniques are used to change the value of Δ dynamically, possibly at every step of simulation, to adjust to the current rate of change.

5.3.2 Computing Reachable States

Numerical simulation is an effective technique to understand the behavior of a dynamical system starting from a specific initial state. When we know that initial state belongs to a set, which can be described by constraints on initial values, the simulation tool needs to choose different values for the initial state from the given set, and run multiple simulations. Such an approach cannot be exhaustive, and thus, we need some alternative techniques.

In this section, let us focus on the following safety verification problem. We are given a dynamical system C with state x and dynamics $\dot{x} = f(x)$. Given a set $Init$ of initial states and a state property φ , we want to know if for every state signal ρ_x with $\rho_x(0) \in Init$, $\rho_x(t)$ satisfies φ for every t . That is, if the system is initialized to start in a state in $Init$, we want to check if the state always stays within the region described by φ . There is no external input: we are assuming that the controller has already been designed to supply the controlled inputs, and if there are uncontrolled inputs, we already have a model for such inputs. The set $Init$ describes the set of possible initial states, and violation of φ denotes an error. For example, the property φ can assert that the error e between the reference signal and the plant output is less than some constant δ ; a violation of this property would indicate an unacceptable overshoot.

A natural strategy is to develop a *symbolic simulation* algorithm in the style of the symbolic reachability algorithm of Section 2.4. To implement such a symbolic reachability analysis, we need a way to represent regions (that is, set of states) that supports operations such as intersection, emptiness test, and subset test. For continuous-time systems, a natural candidate for such a representation is *polyhedra*. An n -dimensional polyhedron is specified by the inequality $Rx \leq b$, where R is $r \times n$ -matrix and b is $n \times 1$ -vector. Each row R_j of the matrix corresponds to a linear constraint over the state variables: $R_j x \leq b_j$, and the number r of rows corresponds to the number of constraints. In one dimension, a polyhedron is an interval, and in two dimensions, a polyhedron is a polygon where the number of constraints needed to represent the polygon corresponds to the number of sides. An alternative to such a constraint-based representation uses the extremal vertices of the polyhedron: each such vertex corresponds to

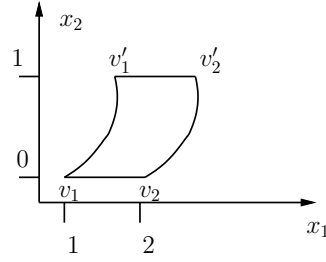


Figure 5.9: Post-image of a line segment under linear dynamics

an n -dimensional vector, and the polyhedron is simply the convex-hull of all the vertices. Algorithms are available to transform one representation to another.

Let us review some operations on polyhedra:

- *Emptiness test*: given a polyhedron represented by $Rx \leq b$, is it empty? This amounts to finding a vector x that satisfies a set of linear constraints, and can be solved using standard linear programming.
- *Intersection*: given two polyhedra $R_1x \leq b_1$ and $R_2x \leq b_2$, compute their intersection. First note that intersection of two polyhedra is indeed a polyhedron. Algorithms are available for computing intersection of polyhedra. However, intersection adds to complexity of representation. For instance, intersection of two triangles can be a hexagon.
- *Union*: Union of two polyhedra need not be a polyhedron. As a result, we can either switch to a representation that consists of a list of polyhedra, or replace union by *convex-hull* of polyhedra. Convex hull of two polyhedra is the smallest polyhedron that contains both, and can be computed effectively from the representation of the input polyhedra.
- *Subset test*: given two polyhedra $R_1x \leq b_1$ and $R_2x \leq b_2$, to check whether the first polyhedron is entirely contained within the second, we can compute the vertices of the first, and check that each such vertex v satisfies $R_2v \leq b_2$.

For our symbolic analysis, we can assume that the initial region $Init$, and the desired invariant property φ are represented using polyhedra. We are ready to use the symbolic reachability algorithm, provided we know how to implement the image computation step Post . More specifically, given a time step Δ , a polyhedron P , and dynamics $\dot{x} = f(x)$, we want to compute the set $\text{Post}_\Delta(P, f)$ of states that can appear on state signals of the system before time Δ when started in a state in set P . This is thus the set of reachable states upto the time horizon Δ . Formally, the set $\text{Post}_\Delta(P, f)$ is

$$\{x \mid \exists t \leq \Delta . \exists \rho : [0, t] \mapsto \text{time}^n . \rho(0) \in P \wedge \rho(t) = x \wedge \forall \tau \in [0, t] . \dot{\rho}(\tau) = f(\rho(\tau))\}$$

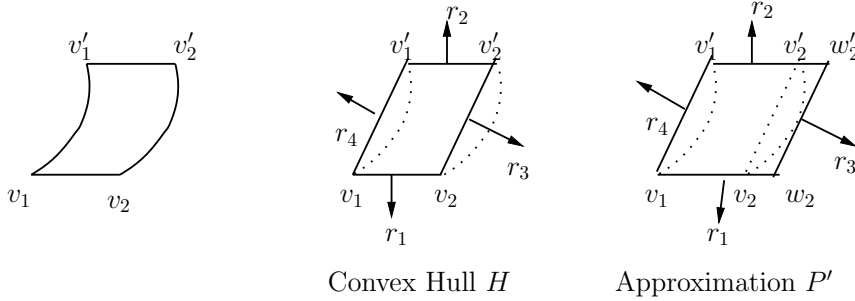


Figure 5.10: Post-image computation by flow-pipe approximation

Unfortunately, even when the system dynamics is linear, the post-image of a polyhedron need not be a polyhedron. For example, consider the 2-dimensional system with the dynamics $\dot{x}_1 = x$ and $\dot{x}_2 = 1$. For the initial set given by $1 \leq x_1 \leq 2 \wedge x_2 = 0$, Figure 5.9 shows the set of reachable states upto time 1. The initial set is the line segment joining v_1 and v_2 , and the set of reachable states upto time 1 is the shape with vertices v_1, v_2, v'_1 , and v'_2 . This is not a polyhedral set, and is not even convex.

The best we can hope for is to over-approximate the post-image: we compute a polyhedron P' that is a superset of $\text{Post}_\Delta(P, f)$. If we can establish that the over-approximation of the reachable set is contained in the desired invariant φ , we have established safety. If states violating φ are found reachable, this may not be real bug: bugs introduced by over-approximation in the analysis tool are called *false negatives*.

We describe a procedure, called *flowpipe computation*, for over-approximating the set $\text{Post}_\Delta(P, f)$ with a polyhedron. This computation proceeds in the following steps:

1. For every vertex v of the initial polyhedron P , compute the state $v' = \text{Post}_\Delta(v, f)$ of the system at time Δ assuming the state is v at time 0. This can be done by numerical simulation for a non-linear system, or from the closed form solution for a linear system (if A is the state matrix for a linear system, then $v' = e^{A\Delta}v$).
2. Compute the convex-hull H of all the vertices v of P and the vertices $v' = \text{Post}_\Delta(v, f)$ obtained in step 1. The region H is a polyhedron, but may not contain all of $\text{Post}_\Delta(P, f)$.
3. Let the representation of the convex-hull H be $Rx \leq d$, where R is $r \times n$ matrix. For each $1 \leq j \leq r$, we compute the smallest d'_j such that the constraint $R_jx \leq d'_j$ holds at all times upto time Δ for all choices of initial states $x_0 \in P$. The values d'_j can be obtained by integrating optimization

in numerical simulation, or for linear systems, by a direct calculation. The desired polyhedral over-approximation is $Rx \leq d'$.

Intuitively, in step 3, we keep the number and normal vectors for facets of the desired polyhedron are same as those for the convex-hull H . The facets are moved outwards so as to include all executions starting in P by adjusting the coefficients d_j .

The computation of a flowpipe approximation is illustrated in Figure 5.10. The vertices of the initial polyhedron are the points v_1 and v_2 . Their post-images v'_1 and v'_2 are computed in the step 1. The convex hull H is the parallelogram connecting these four points, and the vectors r_1, r_2, r_3 , and r_4 are the normal vectors to the sides of this polyhedron H . In step 3, the faces are pushed outside to include all reachable states. The resulting over-approximation is the parallelogram with vertices v_1, w_2, v'_1 , and w'_2 , and has the same normal vectors as the convex hull H .