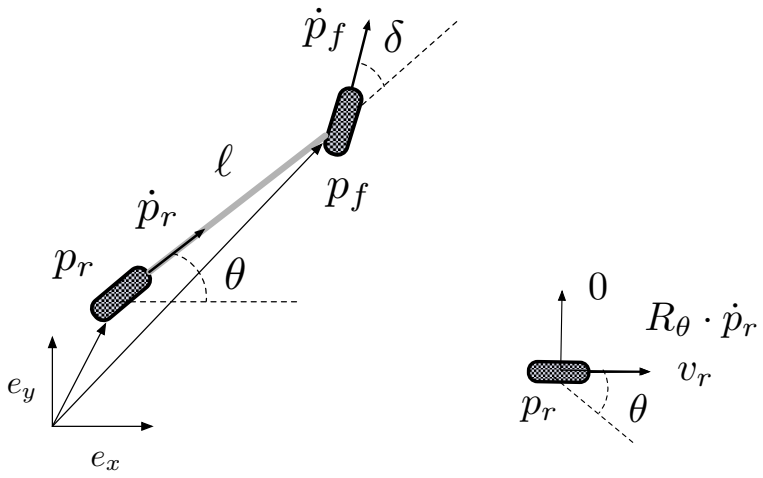

Figure 3.5
Kinematic (bicycle) model of a vehicle. *Left:* Stationary coordinates. *Right:* The rear wheel no-slip condition in local coordinates rotated by θ.

3.4.2 Example: Kinematic vehicle model

Models for ground or aerial vehicles are widely used in motion planning algorithms to approximate a vehicle's behavior under control. High-fidelity models, such as the ones used in commercial simulators like CarSim (Kinjawadekar et al.), may have hundreds of state variables modeling engine torque, tire friction, wheel slip, chassis dynamics, etc. These models may accurately predict the response of the vehicle, but their formal analysis may be intractable. Here we present a basic kinematic model of a vehicle also called the bicycle model or the single-track vehicle model. Creating this type of a model from first principles would be a nontrivial exercise in an undergraduate-level mechanics course. Here we present a swift derivation. The survey article by Paden et al. (2016) discusses several other models, and their relative merits, in designing motion control algorithms for self-driving vehicles.

Our model assumes that the vehicle has two wheels connected by a rigid link, wheels do not slip, and that the front wheel can be turned for steering. The key state variables are as follows: $p_f = (x_f, y_f)$ and $p_r = (x_r, y_r)$ are the positions of the front and rear wheels on the plane in a stationary coordinate system with unit vectors e_x and e_y. The front and rear wheel velocity vectors are $\dot{p}_f$ and $\dot{p}_r$, and the heading angle θ is the angle between x-axis and $(p_f - p_r)$. The steering angle of the front wheel, which is a controlled input variable, is δ.

The main assumption for this model is the *no slip assumption*, which states that the wheels only move in the direction of the plane they reside in. To state this, first, we express

the velocities $\dot{p}_r$ and $\dot{p}_f$ in local coordinates that are aligned with the wheels. For the rear wheel:

$$\dot{p}_{r,local} = R_\theta \dot{p}_r = \begin{bmatrix} (\dot{p}_r \cdot e_x)\cos(\theta) + (\dot{p}_r \cdot e_y)\sin(\theta) \\ -(\dot{p}_r \cdot e_x)\sin(\theta) + (\dot{p}_r \cdot e_y)\cos(\theta) \end{bmatrix}, \; \textit{where } R_\theta = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

is the rotation matrix. Next we equate the second component of $\dot{p}_{r,local}$, which is the component perpendicular to the plane in which the wheel lives, to be 0. Which leads to:

$$(\dot{p}_r \cdot e_x)\sin(\theta) = (\dot{p}_r \cdot e_y)\cos(\theta) \tag{3.17}$$

We denote the first component of $\dot{p}_{r,local}$, which is the (positive or negative) speed of the rear wheel along the direction $p_f - p_r$, by v_r. That is, $v_r = \dot{p}_r \cdot (p_f - p_r)/\|(p_f - p_r)\|$. Similarly, the the no-slip condition for the front wheel in its local coordinate can be derived. The front wheel moves with speed v_f at an angle $\theta + \delta$ with respect to the stationary coordinate system. The component-wise evolution for the wheel positions can be written as follows.

$$\dot{x}_r = v_r \cos(\theta) \qquad \dot{y}_r = v_r \sin(\theta) \qquad \dot{\theta} = (v_r/\ell)\tan(\delta) \tag{3.18}$$

$$\dot{x}_f = v_f \cos(\theta + \delta) \qquad \dot{y}_f = v_f \sin(\theta + \delta) \qquad \dot{\theta} = (v_f/\ell)\sin(\delta). \tag{3.19}$$

The ODE for θ is based on deriving the rate of rotation of the vector $(p_f - p_r)$ when the steering angle is δ (see Figure 3.5). The rear and front wheel velocities are related as $v_r = v_f \cos(\delta)$. The dynamics of the heading angle θ is sometimes simplified and written as $\dot{\theta} = \omega$, in terms of a new controlled input variable ω, with the additional constraint $\omega \in [(v_r/\ell)\tan(\delta_{min}), (v_r/\ell)\tan(\delta_{max})]$. Here $[\delta_{min}, \delta_{max}]$ is the constant range limit of the steering input (δ). The second controlled input variable, which is also range-limited, is the forward speed $v_r \in [v_{min}, v_{max}]$. In this setting, the model is sometimes referred to as the *unicycle model* as valuations of all the state variables can be derived by considering the motion of a single wheel. Figure 3.6 shows this kinematic model explicitly written out in our language. Note that we cannot yet simulate this model without specifying the behavior of the control input variables v_r and δ.

```
1  automaton Vehicle(ℓ, δmin, δmax, vmin, vmax : ℝ≥0)
     variables
3      xr, xf, yr, yf : Real
       θ : Real
5      vr : [vmin, vmax]
       ω : [(vmin/ℓ) tan(δmin), (vmax/ℓ) tan(δmax)]
7      vf : Real
6  trajectories
     evolve
8      vf := vr 1/ cos(δ)
       d(xr) := vr cos(θ)
10     d(yr) := vr sin(θ)
       d(xf) := vf cos(θ + δ)
12     d(yf) := vf sin(θ + δ)
       d(θ) := ω
```

Figure 3.6
Specification of open-loop kinematic vehicle model.

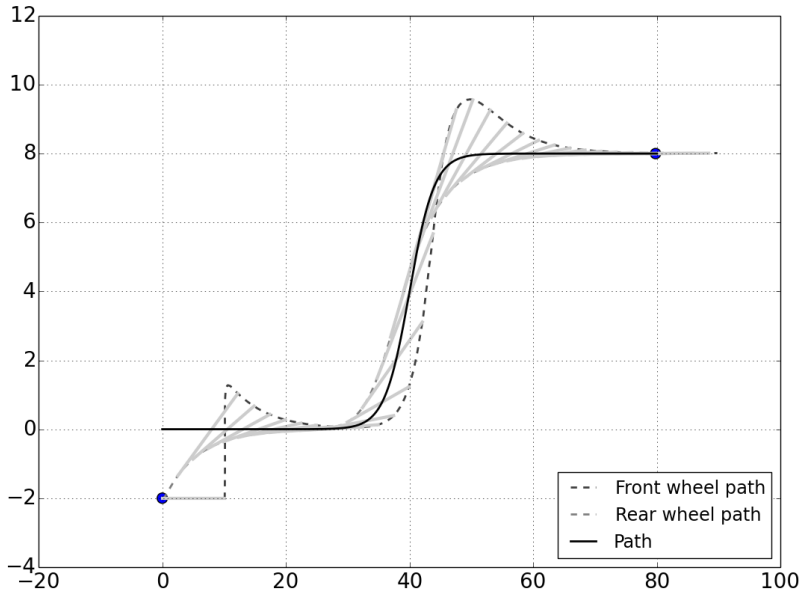

Figure 3.7
Path following trajectory of kinematic Vehicle model.

3.5 Lyapunov's direct method for proving stability

Proving stability can be challenging. Numerical simulations cannot cover all possible initial conditions, cannot be extended to infinite time, and are prone to accumulation of errors. Lyapunov's direct method bypasses explicit computation of trajectories altogether! It gives a *sufficient condition* for proving stability in terms of a special type of function called a *Lyapunov function*. These sufficient conditions can be checked by computing derivatives of the candidate Lyapunov function.

Theorem 3.3. *(Lyapunov stability) Consider the system of (3.7) with state space val(x), and suppose there exists a positive definite, continuous function* $V : val(X) \to \mathbb{R}$. *The system is:*

1. *Lyapunov stable if*

$$\dot{V}(\xi(t)) \triangleq \frac{\partial V}{\partial x} f(x) \leq 0. \tag{3.20}$$

2. *asymptotically stable if, for all* $x \neq 0$,

$$\dot{V}(\xi(t)) < 0. \tag{3.21}$$

It is globally asymptotically stable if V is also radially unbounded.

The function V is called a *Lyapunov function* of (3.7) if it satisfies the strict inequality. Otherwise, it is called a *weak Lyapunov function.* For general nonlinear models, finding a Lyapunov function can be difficult, but for LTI systems, one can find them by solving an optimization problem (see Theorem 3.5).

Exercise 3.6. Suppose V is a Lypaunov function for (3.13). Show that any sublevel set of V containing the initial set Θ is an invariant of the system.

3.5.1 Stability of linear dynamical systems

For an autonomous linear time-invariant (LTI) system

$$\dot{x} = Ax, \tag{3.22}$$

we can check stability or instability by computing certain characteristic properties of the matrix A without actually computing the trajectories of the system or finding a Lyapunov function. The next Theorem is well-known and it characterizes stability of an LTI in terms of the eigenvalues and the Jordan decomposition of the matrix A. A brief overview of these concepts is given in Appendix A.

Theorem 3.4. *The system (3.22) is:*

1. *Lyapunov stable iff all the eigenvalues of A have real parts that are either zero or negative and the Jordan blocks corresponding to the eigenvalues with zero real parts are of size* 1.
2. *Asymptotically stable iff all the eigenvalues of A have strictly negative real parts. Such a matrix is called* Hurwitz.
3. *Unstable iff at least one eigenvalue of A has a positive real part or zero real part, but the size of the corresponding Jordan block is larger than* 1.

The second condition in the above theorem implies that $\xi(t) = e^{At}$ converges to zero exponentially fast, and it follows that asymptotic and exponential stability are equivalent for LTI systems.

We can apply Lyapunov's method to LTI systems as well, and in this case, we can compute a Lyapunov function by solving a special type of an optimization problem called a *semi-definite program.*

Theorem 3.5. *The system (3.22) is asymptotically stable iff for every positive definite matrix Q, there exists a unique symmetric positive definite matrix P that solves the Lyapunov*

equation:

$$A^T P + PA = -Q. \tag{3.23}$$

To apply this theorem, we choose a positive definite Q, for example the identity matrix, and solve for P. If a positive definite P is then found, the quadratic Lyapunov function derived is $V(x) \triangleq x^T Px$. We can see that $\dot{V}(x) = (Ax)^T Px + x^T P(Ax)$, which can be written as $x^T(A^T P + PA)x = -xQx$. Since Q is positive definite, V satisfies (3.21).

In summary, Lyapunov's method for proving stability of ODEs does not require explicit solutions, but it does require one to find a Lyapunov function. In general, for nonlinear systems, it may not be easy to find such a function, but for linear systems, we can use Theorem 3.5. Does a stable system necessarily always have a Lyapunov function? That is, does the converse of Theorem 3.4 also hold? Indeed, under some additional assumptions, such converse Lyapunov theorems also hold. In particular, stable linear systems always have a quadratic Lyapunov function.

3.6 Differential equations as automata

In this section, we relate ODEs to the automata models of Chapter 2. By choosing a sampling time $\delta > 0$, we can construct an automaton $\mathcal{A}(\delta)$ (Definition 2.1) whose every transition models an advance of time by δ amount in the ODE model. Consider, for example, an automaton $\mathcal{A}(\delta) = (V, \Theta, A, \mathcal{D})$ that corresponds to the system in Equation (3.13). Since $x \in \mathbb{R}^n$, we can define $V = \{x\}$ with $type(x) = \mathbb{R}^n$. An alternative choice would be to define $V' = \{x_1, \ldots, x_n\}$, with $type(x_i) = \mathbb{R}$ for each $i \in \{1, \ldots, n\}$. The state space $val(V)$ is the set of all possible valuations of x, which is isomorphic to $\mathbb{R}^n$. A set of initial states Θ could be defined as a predicate on x. The set of action names is immaterial here; we choose $A = \{a\}$. Finally, we define the set $\mathcal{D}$ of transitions as $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ iff there exists a solution $\xi : [0, \delta] \to val(V)$ of 3.13 from $\mathbf{x}$ to $\mathbf{x}'$, that is, that $\xi(0) = \mathbf{x}$ and $\xi(\delta) = \mathbf{x}'$. This sampled time model is akin to the automaton model described in Section 2.3.3. Instead of working with a sampled version of the solutions of an ODE, one often finds it convenient to work with discrete-time dynamical systems described in terms of *difference equations* that take the general form:

$$x(t + 1) = f(x(t), t).$$

For an in-depth treatment of verification (and synthesis) for discrete-time models, we refer the reader to the book by Belta et al. (2017).

3.7 Example: Simple economy

A simple linear model of a country's economy is given by:

$$\begin{aligned} \dot{x} &= x - \alpha y, \\ \dot{y} &= \beta(x - y - g), \end{aligned} \tag{3.24}$$

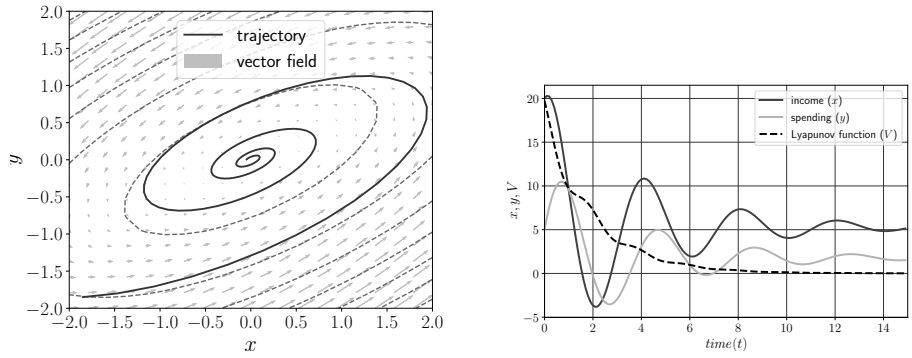

Figure 3.8
Trajectories of simple economy model. *Left:* Phase portrait for the model of Equation 3.24 for $\alpha = 3, \beta = 1.5, k = .1$, and $g_0 = 3$. the sublevel sets of the Lyapunov function ares shown in dotted lines. *Right:* Evolution of the solutions over time.

where x is the national income; y is the rate of consumer spending; g is the rate of government expenditure, which is related to the national income by the algebraic rule $g = g_0 + kx$; and α, β, k are positive constants ($\alpha, \beta > 1$). If we set the right-hand side to zero, we find that the valuation $\mathbf{x}^*$ with $\mathbf{x}^* \lceil x = \frac{g_0 \alpha}{\alpha - 1 - k\alpha}$, $\mathbf{x}^* \lceil y = \frac{g_0}{\alpha - 1 - k\alpha}$ is an equilibrium point of the system. Note that the income-independent part of the government expenditure g_0 influences the equilibrium. If we shift the coordinates to $\mathbf{x}^*$, then the dynamics in the new coordinate system is the following LTI model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \end{bmatrix} = \begin{bmatrix} 1 & -\alpha \\ \beta(1-k) & -\beta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}. \tag{3.25}$$

For $\alpha = 3, \beta = 1$, and $k = 0$, the eigenvalues of matrix A are $(-0.25 + i1.714)$ and $(-0.25 - i1.714)$, and since the real parts of both are negative, by Theorem 3.4 it follows that the system is asymptotically stable and that the national income and consumer spending stabilize to $\mathbf{x}^* \lceil x = 1.764$ and to $\mathbf{x}^* \lceil y = 5.294$, respectively. By choosing a positive definite matrix Q, say $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, we can solve the Lyapunov equation 3.23 to obtain $P = \begin{bmatrix} 2.5971 & -2.2941 \\ -2.2941 & 4.9216 \end{bmatrix}$. By Theorem 3.5, the resulting Lyapunov function is the quadratic function is $V(\mathbf{x}) = (\mathbf{x} - \mathbf{x}^*) P (\mathbf{x} - \mathbf{x}^*)^T$ and this function decays along all trajectories of (3.25). Figure 3.8 on the right shows how the valuation of the Lyapunov function decays

over time, and the left shows some of its sublevel sets. It can be shown that any sublevel set containing the initial set of the system is an invariant (Exercise 3.11).

3.8 Numerical simulations for ordinary differential equations

In general, finding or computing exact solutions to nonlinear differential equations is hard. Numerical ODE solvers provide numerical approximations of solutions from particular initial states and for a bounded time (Nedialkov (2006); CAPD). In fact, there are two types of numerical simulation problems. *Initial value problems,* are what we just described: given an initial state $\mathbf{x}_0$, we have to compute the valuations of state variables along a trajectory $\xi(t)$ at a finite discrete sequence of time points, for example, $t = 0, \delta, 2\delta, \ldots, n\delta$. In general, the time points need not be evenly spaced. In a *boundary value problem,* the boundary conditions on ξ are specified at more than one point. In what follows, we discuss approaches for solving the initial value problem for ODEs.

A numerical solution or simulation is a finite sequence of state-time pairs $(\mathbf{x}_0, t_0)$, $(\mathbf{x}_1, t_1)$, $\ldots (\mathbf{x}_n, t_n)$, such that for each pair $(\mathbf{x}_i, t_i)$, the error $|\mathbf{x}_i - \xi(t_i)|$ between the actual trajectory ξ starting from $\mathbf{x}_0$ at t_i and the numerically computed state $\mathbf{x}_i$ is small. An alternative set-valued definition of numerical solutions is used in Section 11.1.

For computing numerical solutions, the basic idea is to convert the differential equation to an algebraic update rule for the dependent variables for finite changes in the independent variable t. Consider the autonomous differential equation of (3.12). If we choose the step size of the independent variable (time) to be $h > 0$, then a simple update rule starting from initial state $\mathbf{x}_0$ is the Euler method:

$$\begin{aligned} t_{k+1} &= t_k + h \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + hf(\xi(t_k), t_k). \end{aligned} \tag{3.26}$$

Obviously, that is inaccurate, as it updates the state by advancing time by h units, but only uses the stale derivative value from the beginning of the interval. If we instead use the derivative value from the midpoint of the interval, then the resulting symmetrization cancels the first-order error term and gives a second-order Runge-Kutta method, which is also called the *midpoint method*.

$$\begin{aligned} t_{k+1} &= t_k + h, \\ d_1 &= hf(\mathbf{x}_k, t_k), \\ d_2 &= hf(\mathbf{x}_k + \frac{d_1}{2}, t_k + \frac{h}{2}), \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + d_2. \end{aligned} \tag{3.27}$$

It can be shown by power series expansion that the error $|\mathbf{x}_k - \xi(t_k)|$ introduced in each step is $O(h^3)$. The above method uses one additional intermediate evaluation of the right-hand side between t_k and t_{k+1}. Going further in that direction we arrive at the commonly used

fourth-order Runge-Kutta method:

$$\begin{aligned} t_{k+1} &= t_k + h, & d_1 &= hf(\mathbf{x}_k, t_k), \\ d_2 &= hf(\mathbf{x}_k + \frac{d_1}{2}, t_k + \frac{h}{2}), & d_3 &= hf(\mathbf{x}_k + \frac{d_2}{2}, t_k + \frac{h}{2}), \\ d_4 &= hf(\mathbf{x}_k + d_3, t_k + h), & \mathbf{x}_{k+1} &= \mathbf{x}_k + \frac{d_1}{6} + \frac{d_2}{3} + \frac{d_3}{3} + \frac{d_4}{6}. \end{aligned}$$

This method requires four evaluations of the right-hand side, and the error in each step is

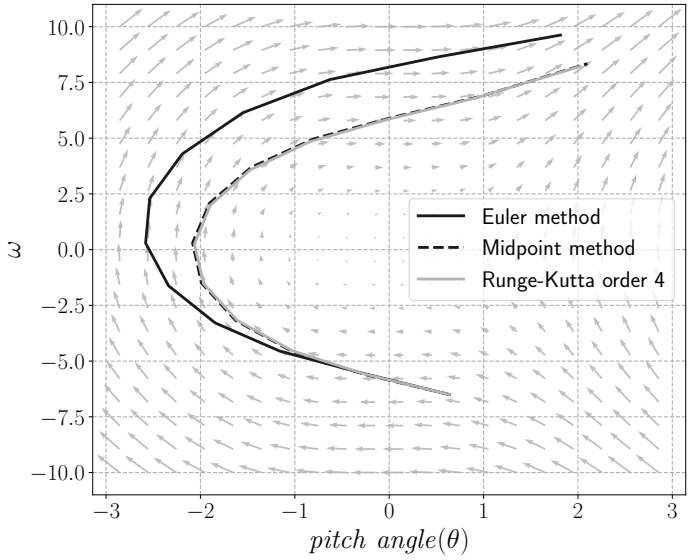

Figure 3.9
Simulated trajectories for the helicopter system of Equation (3.28). Here the midpoint method and the fourth-order Runge-Kutta method give almost indistinguishable simulated points, while the results from the Euler method differ significantly.

$O(h^5)$. The fourth-order method is usually more accurate than the midpoint method, but in general, higher order does not necessarily imply higher accuracy. There are faster and more accurate methods for generating simulations of differential equations, and some of them use variable or adaptive step sizes **?**.

Consider a nonlinear ODE model of a tabletop helicopter system:

$$\frac{d^2\theta}{dt^2} = -\Omega^2 \cos\theta + u, \tag{3.28}$$

which describes the dynamic relationship between the pitch angle (θ) and the input rotor thrust (u) of a. Here Ω is a constant: the moment of inertia of the helicopter. Numerical solutions of the system with different methods are shown in Figure 3.9.

3.9 Closing the loop and control synthesis

The semantics of *open ODEs* or *open-loop systems* with inputs, as in Equation (3.7), is defined in the same way as for autonomous (or closed) systems (see Section 3.4). From Theorem 3.1, we know that for any initial state $\mathbf{x}_0 \in \Theta$ and any piecewise continuous input trajectory η, (3.7) has a unique state trajectory or solution. Let $\xi_{\mathbf{x}_0,\eta} : [0, T] \rightarrow val(x)$ denote that solution. The set of all such trajectories, from all initial states and with all possible inputs, defines the set of possible open-loop behaviors of the system. A state $\mathbf{x}$ is *reachable* if there is an initial state $\mathbf{x}_0$, an input η, and a time t, such that $\xi_{\mathbf{x}_0,\eta}(t) = \mathbf{x}$. Open-loop reachable sets and bounded reachable sets are defined in an analogous fashion.

Given an open-loop system or a plant, a natural problem is to *design a controller* for closing the loop such that the resulting closed-loop control system meets the desired requirements. Figure 3.9 shows the archetype of a closed-loop system. The plant dynamics follows Equation (3.7) but with an additional disturbance input $d(t)$. In general, the plant state $x(t)$ is not available to the controller, but instead an observation $y(t) = h(x(t))$ of the state is. Here $h(\cdot)$ is the observation function that models the mapping from actual states to sensed outputs, possibly including quantization and other effects. The desired output $y_d(t)$ is also called the *set point*, and the actual input to the controller is the *error* between $y(t)$ and $y_d(t)$. Finally, $g(\cdot)$ is the controller function that decides the input $u(t)$ to the plant based on the current observation $y(t)$ or the current error $e(t)$.

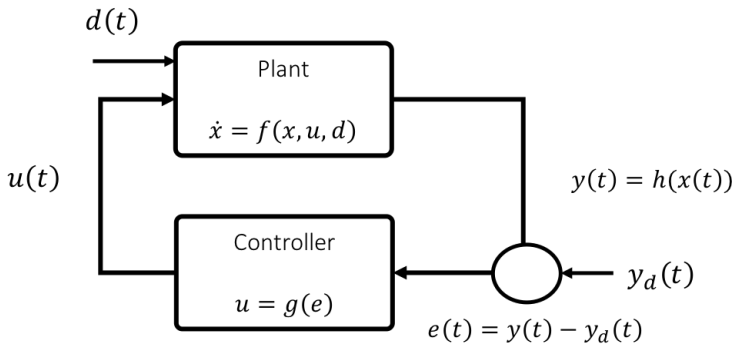

Figure 3.10
A closed-loop control system.

Controller design encompasses different approaches for constructing the controller function $g(\cdot)$ such that all resulting trajectories of the closed-loop system meet the desired design requirements. Requirements could be formally stated in terms of stability, invariants, performance relative to disturbances, minimization of certain gains, etc. We will discuss requirements in more detail in Chapter 6. In the next section we discuss the PID controller, which is one of the predominant types of controllers used in the industry.

3.9.1 PID controller

PID controller is by far the industry's most common way of designing a practical controller. According to a survey of over eleven thousand controllers in the refining, chemicals, and pulp and paper industries, 97% of controllers utilize PID. The form of the PID control function is:

$$u(t) = K_P e(t) + K_I \int_0^t e(s)ds + K_D \frac{d}{dt} e(t), \tag{3.29}$$

where K_P, K_I, and K_D are the proportional (P), integral (I), and differential (D) gains for the controller. Roughly speaking, the proportional term $K_P e(t)$ is proportional to the magnitude of the error and by itself may not be able to eliminate steady-state error in the face of disturbances. The integral term is proportional to both the magnitude and the duration of the error and eliminates steady-state errors. However, as this term responds to accumulated errors from the past, it can cause the system to overshoot the set point value. The derivative term $K_D \frac{d}{dt} e(t)$ predicts the error change and improves settling time and stability. Tuning the gain terms K_P, K_I and K_D involves trading-off different performance criteria of the control system.

As a simple illustration, consider a plant with single-integrator dynamics:

$$\dot{y}(t) = u(t) + d(t),$$

that is, $d(t)$ is an unknown disturbance and $u(t)$ is the control input. Using proportional control $u(t) = -K_P e(t) = -K_P(y(t) - y_d(t))$, the closed loop system equation becomes:

$$\dot{y}(t) = -K_P y(t) + K_P y_d(t) + d(t).$$

The negative proportional gain $(-K_P)$ ensures that the output $y(t)$ is stable. Assuming the steady-state disturbance value of d_{ss} and a constant set point y_{dss}, the steady-state output is $y_{ss} = y_{dss} + \frac{d}{K_P}$, and the steady-state error is $\frac{d}{K_P}$. The transient solution is:

$$y(t) = y(0)e^{-t/T} + y_{ss}(1 - e^{-t/T})),$$

where the time constant $T = 1/K_P$. Thus, to make the steady-state error small, we can increase the proportional gain K_P at the expense of longer settling times and higher risk of instability.

We refer the interested readers to the textbooks by Ogata (2009) and Astrom and Murray (2008) for detailed discussion of PID gain tuning methods such as Ziegler-Nichols method, the relay method, and the Cohen-Coon method.

3.9.2 Controller synthesis problem

Beyond PID controllers, there are many other approaches for designing controllers and this is an active area of research both within control theory and in computer science.

Definition 3.6. Given the dynamical system of Equation (3.7) and a set of requirements R, the *control design* or *controller synthesis* problem is that of computing inputs η (if possible) such that every trajectory meets the requirement R. If no such controller exists, then this fact should be established.

There are several connections between the synthesis problem and verification problem. First, if the number of possible controllers is finite, then, each of these finitely many possibilities can be verified against the requirement, until we either find a satisfying controller or we establish that none of the controllers meet the requirement. Secondly, the requirements are a common input to both problems. We will see in Chapter 6 that many types of requirements beyond stability and invariance can be specified, for example using temporal logics, both for verification and synthesis. The broad categories of synthesis approaches for temporal logic specification are outlined in Section 6.6.2. Finally, many verification and synthesis algorithms utilize common concepts that we will encounter in this book, such as discrete abstractions, Lyapunov analysis, and satisfiability modulo theory encodings.

Summary and outlook. We have seen how physical processes can be mathematically modeled using ordinary differential equations (ODEs). We added a new feature in the language of Chapter 2 to include ODEs. We defined the semantics of ODEs in terms of solutions and discussed sufficient conditions for their existence and uniqueness. We also discussed reachable states, stability, their relationship, and Lyapunov's method for proving stability. In the next chapter, we will unify the concepts from Chapters 2 and 3 and see cyberphysical systems mathematically modeled using automata and differential equations.

3.10 Problems

Exercise 3.7. Consider the kinematic car model Vehicle of Section 3.4.2 and fix rear wheel speed to a constant $v_r > 0$. Design a controller function that takes as input the current state $\mathbf{x}$ of the Vehicle automaton, a target heading direction θ^* and produces as output the steering ω, so that the vehicle eventually heads in the direction θ^*. Write a program implementing this closed-loop system. [Hint: Use a proportional controller.]

Exercise 3.8. Define and simulate a waypoint following controller for the Vehicle automaton. Given a target state $\mathbf{x}^*$ and gain constants $k_1, k_2, k_3 > 0$, define the control inputs at

state $\mathbf{x}$ as follows:

$$v_r = (\mathbf{x}^* \lceil v_r) \cos(\theta_e) + k_1 x_e$$
$$\omega = \mathbf{x}^* \lceil \omega + (\mathbf{x}^* \lceil v_r)(k_2 y_e + k_3 \sin(\theta_e)),$$

where the "error" vector $[x_e\ y_e\ \theta_e]$ at state $\mathbf{x}$ relative to the target state $\mathbf{x}^*$ is defined as:

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos(\mathbf{x} \lceil \theta) & \sin(\mathbf{x} \lceil \theta) & 0 \\ -\sin(\mathbf{x} \lceil \theta) & \cos(\mathbf{x} \lceil \theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (\mathbf{x}^* \lceil x_r) - (\mathbf{x} \lceil x_r) \\ (\mathbf{x}^* \lceil y_r) - (\mathbf{x} \lceil y_r) \\ (\mathbf{x}^* \lceil \theta) - (\mathbf{x} \lceil \theta) \end{bmatrix}.$$

Complete the code in Figure 3.6 to create a model of a controlled vehicle. Write a program implementing this closed-loop system.

Exercise 3.9. Give an example of an unstable linear time-varying dynamical system $\dot{x} = A(t)x$, such that for every $t \geq 0$, $A(t)$ is Hurwitz (each eigenvalue has negative real parts).

Exercise 3.10. For a linear autonomous system $\dot{x} = Ax$, show that if the set of initial states is convex, then the set of reachable states at any time $t > 0$ is also convex. If the set of initial set Θ is a bounded polyhedron, then $\mathsf{Reach}(\Theta, t)$ is the convex hull of the points reached from the vertices of Θ.

Exercise 3.11. Consider an autonomous dynamical system $\dot{x} = f(x)$ with initial set $\Theta \subseteq val(x)$ and a Lyapunov function $V : val(x) \to \mathbb{R}$. Show that any sublevel set of V containing Θ is an invariant of the system.

Exercise 3.12. Consider the Vehicle model with the controller defined in Exercise 3.8. Show that the following function is a Lyapunov function for the closed loop system.

$$V = \frac{1}{2}(x_e^2 + y_e^2) + (1 - \cos(\theta_e))/k_2.$$