Feedback Systems

An Introduction for Scientists and Engineers
SECOND EDITION

Karl Johan Åström
Richard M. Murray

Version v3.0g (1 Nov 2015)

PRINCETON UNIVERSITY PRESS
PRINCETON AND OXFORD

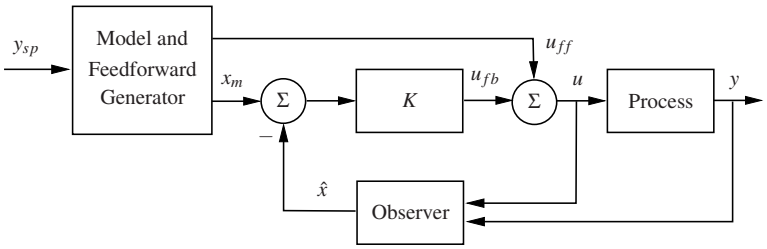


Figure 14.18: Block diagram of a controller based on model following, state feedback, and an observer.

be used in the individual subsystems, instead we can select one controller with integral action or we can use *a central integrator* and distribut its output to the controler of the subsystems.

14.5 Top-Down Architectures

- Introduction: controllers, logic
- Correctness by design
- Logic and FMS
- State Feedback and Observers
- Theorem proving
- State Based Control, FSM
- Model Predictive Control

Top-down paradigms start with a problem formulation in terms of an optimization problem. Paradigms that support a top-down approach are optimization, state feedback, observers, predictive control, and linearization. In the top-down approach it is natural to deal with many inputs and many outputs simultaneously. Since this is not the main topic of this book we will only give a brief discussion. The top-down approach often leads to the controller structure shown in Figure 14.18. In this system all measured process variables y together with the control variables u are sent to an observer, which uses the sensor information and a mathematical model to generate a vector $\hat{x}$ of good estimates of internal process variables and important disturbances. The estimated state $\hat{x}$ is then compared with the ideal state x_m produced by the feedforward generator, and the difference is fed back to the process. The feedforward generator also gives a feedforward signal u_{ff}, which is sent directly to the process inputs. The controller shown in Figure 14.18 is useful for process segments where there are several inputs and outputs that interact, but the system becomes very complicated when there is a large number of inputs

and outputs. In such a case it may be better to decompose the system into several subsystems.

An advantage with the top-down approach is that the total behavior of the system is taken into account. A systematic approach based on mathematical modeling and simulation makes it easy to understand the fundamental limitations. Commissioning of the system is, however, difficult because many feedback loops have to be closed simultaneously. When using the top-down approach it is therefore good practice to first tune loops based on simulation, possibly also hardware in the loop simulation.

Real-time trajectory generation [from OBC]

In Section 8.5 we introduced the use of feedforward compensation in control system design. We briefly review those concepts here and provide an overview of approaches to solving the trajectory generation problem and linking it to feedback control problems.

Two degree of freedom design

A large class of control problems consist 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 factor 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 14.19.

In a nonlinear setting, two degree of freedom controller design decouples the trajectory generation and asymptotic tracking problems. Given a desired output 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 early chapters the notation u_{ff} was 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

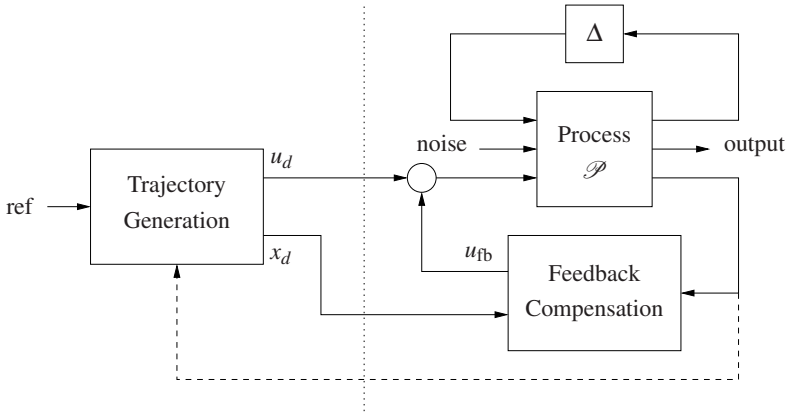


Figure 14.19: Two degree of freedom controller design for a process P with uncertainty Δ. The controller consists of a trajectory generator and feedback controller. The trajectory generation subsystem computes a feedforward command u_d along with the desired state x_d. The state feedback controller uses 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.

nonlinear differential equation of the form

$$\begin{aligned} \dot{x} &= f(x,u), \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^m, \\ y &= h(x,u), \qquad y \in \mathbb{R}^p, \end{aligned} \tag{14.19}$$

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

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

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 (14.19) is a pair $(x_d(t), u_d(t))$ that satisfies the differential equation and generates the desired trajectory:

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

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 **??**. 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 14.19 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 **??**.

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.

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 (14.19) 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:

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

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 $e = 0$:

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

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

$$v = -Ke + k_r r.$$

In the case where r is constant, we can apply the results of Chapter 7 and solve the problem by finding a gain matrix K that gives the desired closed loop dynamics (e.g., by eigenvalue assignment) and choosing k_r to give the desired output value at equilibrium. The equilibrium point is given by

$$x_e = -(A-BK)^{-1}Bk_r r \quad \Longrightarrow \quad y_e = -C(A-BK)^{-1}Bk_r r$$

and if we wish the output to be $y = r$ it follows that

$$k_r = -1/\big(C(A-BK)^{-1}B\big).$$

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 **??**). Exercise **??**

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

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

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* u_d.

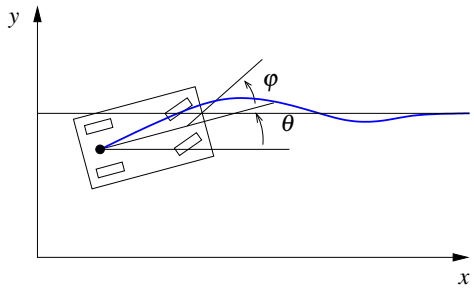
More generally, the term gain scheduling is used to describe any controller that depends on a set of measured parameters in the system. So, for example, we might write

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

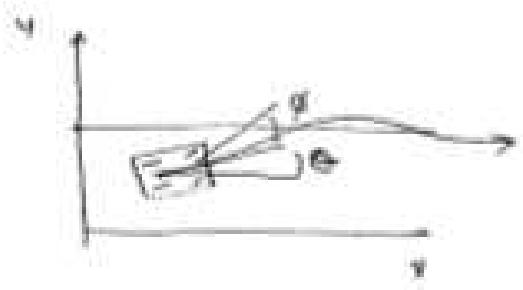
where $K(x, \mu)$ depends on the *current* system state (or some portion of it) and an external parameter μ. The dependence on the current state x (as opposed to the desired state x_d) allows us to modify the closed loop dynamics differently depending on our location in the state space. This is particularly 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 14.3 Steering control with velocity scheduling

Consider the problem of controlling the motion of a automobile so that it follows a given trajectory on the ground, as shown in Figure 14.20a. We use the model



(a)



(b)

Figure 14.20: Vehicle steering using gain scheduling.

derived in Example **??**, choosing the reference point to be the center of the rear wheels. This gives dynamics of the form

$$\dot{x} = \cos\theta\, v, \qquad \dot{y} \;= \sin\theta\, v, \qquad \dot{\theta} = \frac{v}{l}\tan\varphi, \tag{14.20}$$

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

$$A_d = \left.\frac{\partial f}{\partial x}\right|_{(x_d,u_d)} = \left.\begin{pmatrix} 0 & 0 & -\sin\theta \\ 0 & 0 & \cos\theta \\ 0 & 0 & 0 \end{pmatrix}\right|_{(x_d,u_d)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

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

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

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

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 λ_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 accomplished by setting

$$w_1 = -\lambda_1 e_x$$
$$w_2 = \frac{l}{v_r}(a_1 e_y + a_2 e_\theta).$$

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

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

$$\begin{pmatrix} v \\ \varphi \end{pmatrix} = -\underbrace{\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \frac{a_1 l}{v_r} & \frac{a_2 l}{v_r} \end{pmatrix}}_{K_d} \underbrace{\begin{pmatrix} x - v_r t \\ y - y_r \\ \theta \end{pmatrix}}_{e} + \underbrace{\begin{pmatrix} v_r \\ 0 \end{pmatrix}}_{u_d}.$$

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 14.20b 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 14.21. Suppose that we have a set of operating points $x_{d,j}$, $j = 1, \ldots, N$. Then we can write our controller as

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

where K_j is a set of gains designed around the operating point $x_{d,j}$ and $\alpha_j(x)$ is a weighting factor. For example, we might choose the weights $\alpha_j(x)$ 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.

RMM: Redraw

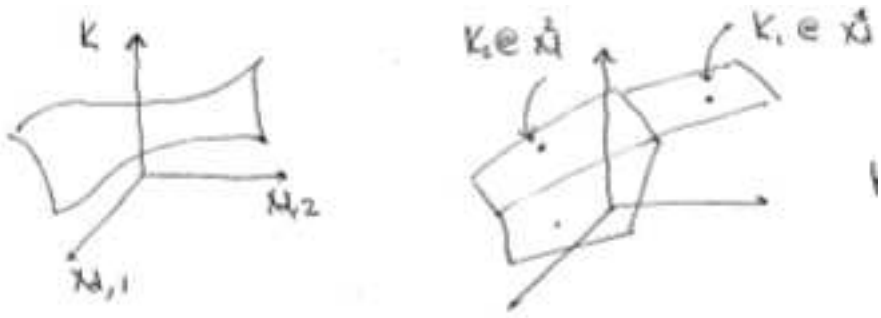


Figure 14.21: Gain scheduling. A general gain scheduling design involves finding a gain K at each desired operating point. This can be thought of as a gain surface, as shown on the left (for the case of a scalar gain). An approximation to this gain can be obtained by computing the gains at a fixed number of operating points and then interpolated between those gains. This gives an approximation of the continuous gain surface, as shown on the right.

RMM Add example showing weighting functions. Use a different example (non-mechanical).

While the intuition behind gain scheduled controllers is fairly clear, some caution in 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 **??**).

Exercise **??**

Trajectory tracking 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:

$$\dot{x}_d = f(x_d,u_d), \qquad x_d(0)=x_0, x_d(T)=x_f. \tag{14.21}$$

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 show in Figure 14.22. To solve this problem, we must find a solution to the differential equations (14.20) that satisfies the endpoint conditions. Given the nonlinear nature

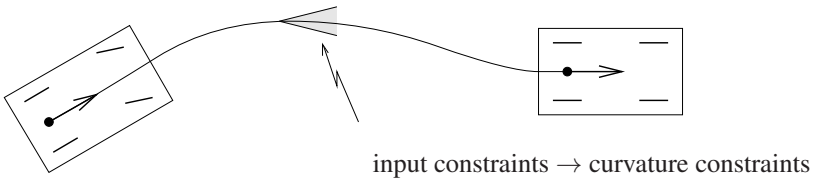


Figure 14.22: Simple model for an automobile. We wish to find a trajectory from an initial state to a final state that satisfies the dynamics of the system and constraints on the curvature (imposed by the limited travel of the front wheels).

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 (14.20), we see that we can use this solution to solve for the angle of the car by writing

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

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

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

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

$$\dot{\theta} = \frac{v}{l}\tan\varphi \qquad \Longrightarrow \qquad \varphi_d = \tan^{-1}\left(\frac{l\dot{\theta}_d}{v_d}\right).$$

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 14.1 (Differential flatness)**.** A nonlinear system (14.19) is *differentially flat* if there exists a function α such that

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

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

$$\begin{aligned} x &= \beta(z, \dot{z}, \ldots, z^{(q)}), \\ u &= \gamma(z, \dot{z}, \ldots, z^{(q)}). \end{aligned} \tag{14.22}$$

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

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

If the system is differentially flat then

$$\begin{aligned} x(0) &= \beta\big(z(0), \dot{z}(0), \ldots, z^{(q)}(0)\big) = x_0, \\ x(T) &= \gamma\big(z(T), \dot{z}(T), \ldots, z^{(q)}(T)\big) = x_f, \end{aligned} \tag{14.23}$$

and we see that the initial and final condition in the full state space depends 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 (14.22) to determine the full state space and input trajectories.

In particular, given initial and final conditions on z and its derivatives that satisfy equation (14.23), 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 $\psi_i(t)$:

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

We seek a set of coefficients α_i, $i = 1, \ldots, N$ such that $z(t)$ satisfies the boundary conditions (14.23). The derivatives of the flat output can be computed in terms of the derivatives of the basis functions:

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

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

$$\begin{pmatrix} \psi_1(0) & \psi_2(0) & \ldots & \psi_N(0) \\ \dot{\psi}_1(0) & \dot{\psi}_2(0) & \ldots & \dot{\psi}_N(0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(0) & \psi_2^{(q)}(0) & \ldots & \psi_N^{(q)}(0) \\ \psi_1(T) & \psi_2(T) & \ldots & \psi_N(T) \\ \dot{\psi}_1(T) & \dot{\psi}_2(T) & \ldots & \dot{\psi}_N(T) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(T) & \psi_2^{(q)}(T) & \ldots & \psi_N^{(q)}(T) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix} = \begin{pmatrix} z(0) \\ \dot{z}(0) \\ \vdots \\ z^{(q)}(0) \\ z(T) \\ \dot{z}(T) \\ \vdots \\ z^{(q)}(T) \end{pmatrix}$$

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

Example 14.4 Nonholonomic integrator
A simple nonlinear system called a *nonholonomic integrator* [**?**] is given by the differential equations

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

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

$$x_1 = z_1, \qquad x_2 = \dot{x}_3/\dot{x}_1 = \dot{z}_2/\dot{z}_1, \qquad x_3 = z_2. \tag{14.24}$$

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

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

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. ∇

Add remarks: **RMM**

- There are several remaining degrees of freedom (T, 1, etc) that have to be specified $\Longrightarrow$ solutions are not unique.
- Possible to over-parameterize and solve using least squares

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.

RMM Add a text + equations for the tracking problem, plus a worked out example (double integrator)

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.

RMM Add example (non-mech) showing optimization

In light of the techniques that are available for differentially flat systems, the characterization of flat systems becomes particularly important. Unfortunately, general conditions for flatness are not known, but many important class of nonlinear systems, including feedback linearizable systems, are differential flat. One large class of flat systems are those in "pure feedback form":

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

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. [**?**] and constructive controllability techniques for nonholonomic systems in chained form [**?**].† Figure 14.23 shows some additional systems that are differentially flat.

RMM: Check to make sure this is the correct reference

RMM The example has been rewritten to use the PVTOL example of early chapters. Check to make sure it reads correctly.

Example 14.5 Vectored thrust aircraft

Consider the dynamics of a planar, vectored thrust flight control system as shown in Figure 14.24. 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 **??**). 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.

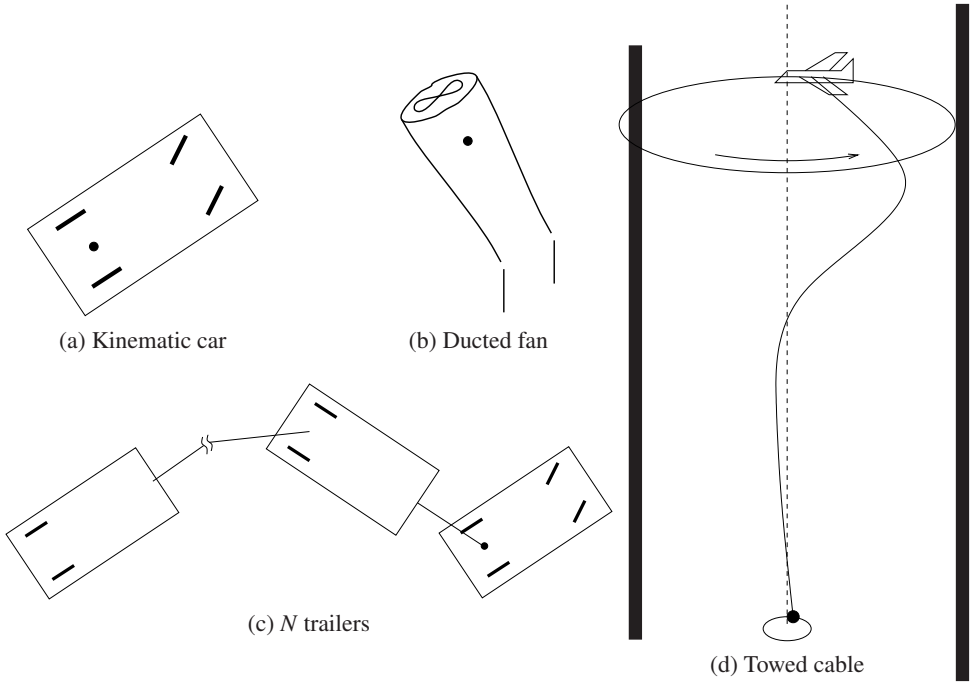


Figure 14.23: Examples of flat systems.

The dynamics for the system are

$$\begin{aligned} m\ddot{x} &= F_1 \cos\theta - F_2 \sin\theta, \\ m\ddot{y} &= F_1 \sin\theta + F_2 \cos\theta - mg, \\ J\ddot{\theta} &= rF_1. \end{aligned} \tag{14.25}$$

Martin et al. [**?**] showed that this system is differentially flat and that one set of flat outputs is given by

$$\begin{aligned} z_1 &= x - (J/mr)\sin\theta, \\ z_2 &= y + (J/mr)\cos\theta. \end{aligned} \tag{14.26}$$

Using the system dynamics, it can be shown that

$$\ddot{z}_1 \cos\theta + (\ddot{z}_2 + g)\sin\theta = 0 \tag{14.27}$$

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. ∇

Other methods of trajectory generation

a. Primitives b. Graph search

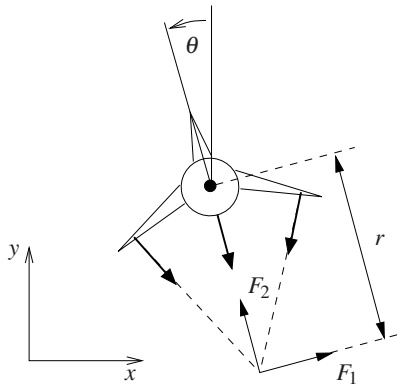


Figure 14.24: Vectored thrust aircraft (from Example **??**). The net thrust on the aircraft can be decomposed into a horizontal force F_1 and a vertical force F_2 acting at a distance r from the center of mass.

Receding Horizon Control

This set of notes builds on the previous two chapters and explores the use of online optimization as a tool for control of nonlinear control. We begin with a high-level discussion of optimization-based control, refining some of the concepts initially introduced in Chapter **??**. We then describe the technique of receding horizon control (RHC), including a proof of stability for a particular form of receding horizon control that makes use of a control Lyapunov function as a terminal cost. We conclude the chapter with a detailed design example, in which we can explore some of the computational tradeoffs in optimization-based control.

Optimization-Based Control

Optimization-based control refers to the use of online, optimal trajectory generation as a part of the feedback stabilization of a (typically nonlinear) system. The basic idea is to use a *receding horizon control* technique: 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 starting at time $t + \delta$ until time $t + T + \delta$. Development and application of receding horizon control (also called model predictive control, or MPC) originated in process control industries where the processes being controlled are often sufficiently slow to permit its implementation. An overview of the evolution of commercially available MPC technology is given in [**?**] and a survey of the state of stability theory of MPC is given in [**?**].

RMM: Move to further reading

Design approach. The basic philosophy that we propose is illustrated in Figure 14.25. We begin with a nonlinear system, including a description of the constraint set. We linearize this system about a representative equilibrium point and perform a linear control design using standard control design tools. Such a design can provide provably robust performance around the equilibrium point and, more importantly,

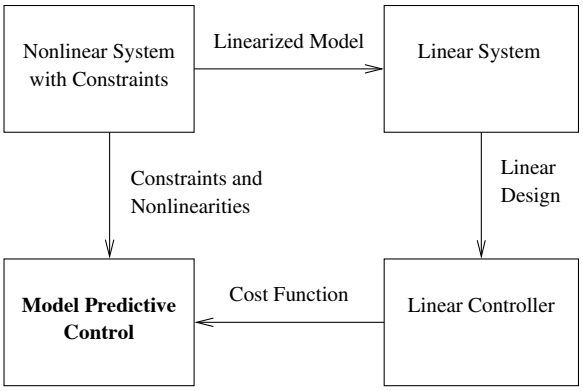


Figure 14.25: Optimization-based control approach.

allows the designer to meet a wide variety of formal and informal performance specifications through experience and the use of sophisticated linear design tools.

The resulting linear control law then serves as a *specification* of the desired control performance for the entire nonlinear system. We convert the control law specification into a receding horizon control formulation, chosen such that for the linearized system, the receding horizon controller gives comparable performance. However, because of its use of optimization tools that can handle nonlinearities and constraints, the receding horizon controller is able to provide the desired performance over a much larger operating envelope than the controller design based just on the linearization. Furthermore, by choosing cost formulations that have certain properties, we can provide proofs of stability for the full nonlinear system and, in some cases, the constrained system.

The advantage of the proposed approach is that it exploits the power of humans in designing sophisticated control laws in the absence of constraints with the power of computers to rapidly compute trajectories that optimize a given cost function in the presence of constraints. New advances in online trajectory generation serve as an enabler for this approach and their demonstration on representative flight control experiments shows their viability [**?**]. This approach can be extended to existing nonlinear paradigms as well, as we describe in more detail below.

An advantage of optimization-based approaches is that they allow the potential for online customization of the controller. By updating the model that the optimization uses to reflect the current knowledge of the system characteristics, the controller can take into account changes in parameters values or damage to sensors or actuators. In addition, environmental models that include dynamic constraints can be included, allowing the controller to generate trajectories that satisfy complex operating conditions. These modifications allow for many state- and environment-dependent uncertainties to including the receding horizon feedback loop, providing potential robustness with respect to those uncertainties.

The paragraph below should be replaced with a paragraph on terminal constraints. **RMM**

Move the result to the further reading section.

A number of approaches in receding horizon control employ the use of terminal state equality or inequality constraints, often together with a terminal cost, to ensure closed loop stability. In Primbs et al. [**?**], aspects of a stability-guaranteeing, global control Lyapunov function (CLF) were used, via state and control constraints, to develop a stabilizing receding horizon scheme. Many of the nice characteristics of the CLF controller together with better cost performance were realized. Unfortunately, a global control Lyapunov function is rarely available and often not possible.

Motivated by the difficulties in solving constrained optimal control problems, researchers have developed an alternative receding horizon control strategy for the stabilization of nonlinear systems [**?**]. In this approach, closed loop stability is ensured through the use of a terminal cost consisting of a control Lyapunov function (defined later) that is an incremental upper bound on the optimal cost to go. This terminal cost eliminates the need for terminal constraints in the optimization and gives a dramatic speed-up in computation. Also, questions of existence and regularity of optimal solutions (very important for online optimization) can be dealt with in a rather straightforward manner.

Inverse Optimality. The philosophy presented here relies on the synthesis of an optimal control problem from specifications that are embedded in an externally generated controller design. This controller is typically designed by standard classical control techniques for a nominal process, absent constraints. In this framework, the controller's performance, stability and robustness specifications are translated into an equivalent optimal control problem and implemented in a receding horizon fashion.

One central question that must be addressed when considering the usefulness of this philosophy is: *Given a control law, how does one find an equivalent optimal control formulation?* The paper by Kalman [**?**] lays a solid foundation for this class of problems, known as *inverse optimality*. In this paper, Kalman considers the class of linear time-invariant (LTI) processes with full-state feedback and a single input variable, with an associated cost function that is quadratic in the input and state variables. These assumptions set up the well-known linear quadratic regulator (LQR) problem, by now a staple of optimal control theory.

In Kalman's paper, the mathematical framework behind the LQR problem is laid out, and necessary and sufficient algebraic criteria for optimality are presented in terms of the algebraic Riccati equation, as well as in terms of a condition on the return difference of the feedback loop. In terms of the LQR problem, the task of synthesizing the optimal control problem comes down to finding the integrated cost weights Q_x and Q_u given only the dynamical description of the process represented by matrices A and B and of the feedback controller represented by K. Kalman delivers a particularly elegant frequency characterization of this map [**?**], which we briefly summarize here.

We consider a linear system

$$\dot{x} = Ax + Bu \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{14.28}$$

with state x and input u. We consider only the single input, single output case for now ($m = 1$). Given a control law

$$u = Kx$$

we wish to find a cost functional of the form

$$J = \int_0^T x^T Q_x x + u^T Q_u u \, dt + x^T(T) P_T x(T) \tag{14.29}$$

where $Q_x \in \mathbb{R}^{n\times n}$ and $Q_u \in \mathbb{R}^{m\times m}$ define the integrated cost, $P_T \in \mathbb{R}^{n\times n}$ is the terminal cost, and T is the time horizon. Our goal is to find $P_T > 0$, $Q_x > 0$, $Q_u > 0$, and $T > 0$ such that the resulting optimal control law is equivalent to $u = Kx$.

The optimal control law for the quadratic cost function (14.29) is given by

$$u = -R^{-1}B^T P(t),$$

where $P(t)$ is the solution to the Riccati ordinary differential equation

$$-\dot{P} = A^T P + PA - PBR^{-1}B^T P + Q \tag{14.30}$$

with terminal condition $P(T) = P_T$. In order for this to give a control law of the form $u = Kx$ for a constant matrix K, we must find P_T, Q_x, and Q_u that give a constant solution to the Riccati equation (14.30) and satisfy $-R^{-1}B^T P = K$. It follows that P_T, Q_x and Q_u should satisfy

$$\begin{aligned} A^T P_T + P_T A - P_T B Q_u^{-1} B^T P_T + Q = 0 \\ -Q_u^{-1} B^T P_T = K. \end{aligned} \tag{14.31}$$

We note that the first equation is simply the normal algebraic Riccati equation of optimal control, but with P_T, Q, and R yet to be chosen. The second equation places additional constraints on R and P_T.

Equation (14.31) is exactly the same equation that one would obtain if we had considered an infinite time horizon problem, since the given control was constant and hence $P(t)$ was forced to be constant. This infinite horizon problem is precisely the one that Kalman considered in 1964, and hence his results apply directly. Namely, in the single-input single-output case, we can always find a solution to the coupled equations (14.31) under standard conditions on reachability and observability [**?**]. The equations can be simplified by substituting the second relation into the first to obtain

$$A^T P_T + P_T A - K^T R K + Q = 0.$$

This equation is linear in the unknowns and can be solved directly (remembering that P_T, Q_x and Q_u are required to be positive definite).

The implication of these results is that any state feedback control law satisfying these assumptions can be realized as the solution to an appropriately defined

receding horizon control law. Thus, we can implement the design framework summarized in Figure 14.25 for the case where our (linear) control design results in a state feedback controller.

The above results can be generalized to nonlinear systems, in which one takes a nonlinear control system and attempts to find a cost function such that the given controller is the optimal control with respect to that cost.

The history of inverse optimal control for nonlinear systems goes back to the early work of Moylan and Anderson [**?**]. More recently, Sepulchre et al. [**?**] showed that a nonlinear state feedback obtained by Sontag's formula from a control Lyapunov function (CLF) is inverse optimal. The connections of this inverse optimality result to passivity and robustness properties of the optimal state feedback are discussed in Jankovic *et al.* [**?**]. Most results on inverse optimality do not consider the constraints on control or state. However, the results on the unconstrained inverse optimality justify the use of a more general nonlinear loss function in the integrated cost of a finite horizon performance index combined with a real-time optimization-based control approach that takes the constraints into account.

Control Lyapunov Functions. For the optimal control problems that we introduce in the next section, we will make use of a terminal cost that is also a control Lyapunov function for the system. Control Lyapunov functions are an extension of standard Lyapunov functions and were originally introduced by Sontag [**?**]. They allow constructive design of nonlinear controllers and the Lyapunov function that proves their stability. A more complete treatment is given in [KKK95].

Consider a nonlinear control system

$$\dot{x} = f(x,u), \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^m. \tag{14.32}$$

Definition 14.2 (Control Lyapunov Function)**.** A locally positive function $V : \mathbb{R}^n \to \mathbb{R}_+$ is called a *control Lyapunov function (CLF)* for a control system (14.32) if

$$\inf_{u \in \mathbb{R}^m} \left(\frac{\partial V}{\partial x} f(x,u) \right) < 0 \qquad \text{for all } x \neq 0.$$

In general, it is difficult to find a CLF for a given system. However, for many classes of systems, there are specialized methods that can be used. One of the simplest is to use the Jacobian linearization of the system around the desired equilibrium point and generate a CLF by solving an LQR problem.

RMM Add a theorem about the existence of a stabilizing control law

As described in Chapter **??**, the problem of minimizing the quadratic performance index,

$$J = \int_0^\infty (x^T(t)Qx(t) + u^T Ru(t))dt \qquad \text{subject to} \qquad \begin{aligned} \dot{x} &= Ax + Bu, \\ x(0) &= x_0, \end{aligned} \tag{14.33}$$

results in finding the positive definite solution of the following Riccati equation:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{14.34}$$

The optimal control action is given by

$$u = -R^{-1}B^T Px$$

and $V = x^T Px$ is a CLF for the system.

In the case of the nonlinear system $\dot{x} = f(x,u)$, A and B are taken as

$$A = \frac{\partial f(x,u)}{\partial x}\Big|_{(0,0)} \qquad B = \frac{\partial f(x,u)}{\partial u}\Big|_{(0,0)}$$

where the pairs (A,B) and $(Q^{\frac{1}{2}},A)$ are assumed to be stabilizable and detectable respectively. The CLF $V(x) = x^T Px$ is valid in a region around the equilibrium $(0,0)$, as shown in Exercise **??**.

Exercise **??**

More complicated methods for finding control Lyapunov functions are often required and many techniques have been developed. An overview of some of these methods can be found in [**?**].

Add examples **RMM**

Finite Horizon Optimal Control. We briefly review the problem of optimal control over a finite time horizon as presented in Chapter **??** to establish the notation for the chapter and set some more specific conditions required for receding horizon control. This material is based on [**?**].

Given an initial state x_0 and a control trajectory $u(\cdot)$ for a nonlinear control system $\dot{x} = f(x,u)$, let $x^u(\cdot;x_0)$ represent the state trajectory. We can write this solution as a continuous curve

$$x^u(t;x_0) = x_0 + \int_0^t f(x^u(\tau;x_0),u(\tau))\,d\tau$$

for $t \geq 0$. We require that the trajectories of the system satisfy an *a priori* bound

$$\|x(t)\| \leq \beta(x,T,\|u(\cdot)\|_1) < \infty, \qquad t \in [0,T],$$

where β is continuous in all variables and monotone increasing in T and $\|u(\cdot)\|_1 = \|u(\cdot)\|_{L_1(0,T)}$. Most models of physical systems will satisfy a bound of this type.

The performance of the system will be measured by an integral cost $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$. We require that L be twice differentiable (C^2) and fully penalize both state and control according to

$$L(x,u) \geq c_q(\|x\|^2 + \|u\|^2), \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^m$$

for some $c_q > 0$ and $L(0,0) = 0$. It follows that the quadratic approximation of L at the origin is positive definite,

$$\frac{\partial L}{\partial x}\bigg|_{(0,0)} \geq c_q I > 0.$$

To ensure that the solutions of the optimization problems of interest are well behaved, we impose some convexity conditions. We require the set $f(x,\mathbb{R}^m) \subset \mathbb{R}^n$ to be convex for each $x \in \mathbb{R}^n$. Letting $\lambda \in \mathbb{R}^n$ represent the co-state, we also require

that the pre-Hamiltonian function $\lambda^T f(x,u) + L(x,u) =: K(x,u,\lambda)$ be strictly convex for each $(x,\lambda) \in \mathbb{R}^n \times \mathbb{R}^n$ and that there is a C^2 function $\bar{u}^* : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$ providing the global minimum of $K(x,u,\lambda)$. The Hamiltonian $H(x,\lambda) := K(x,\bar{u}^*(x,\lambda),\lambda)$ is then C^2, ensuring that extremal state, co-state, *and* control trajectories will all be sufficiently smooth (C^1 or better). Note that these conditions are automatically satisfied for control affine f and quadratic L.

The cost of applying a control $u(\cdot)$ from an initial state x over the infinite time interval $[0,\infty)$ is given by

$$J_\infty(x,u(\cdot)) = \int_0^\infty L(x^u(\tau;x),u(\tau))\,d\tau\,.$$

The optimal cost (from x) is given by

$$J_\infty^*(x) = \inf_{u(\cdot)} J_\infty(x,u(\cdot))$$

where the control function $u(\cdot)$ belongs to some reasonable class of admissible controls (e.g., piecewise continuous). The function $J_\infty^*(x)$ is often called the *optimal value function* for the infinite horizon optimal control problem. For the class of f and L considered, it can be verified that $J_\infty^*(\cdot)$ is a positive definite C^2 function in a neighborhood of the origin [**?**].†

RMM: Look for a better citation

For practical purposes, we are interested in finite horizon approximations of the infinite horizon optimization problem. In particular, let $V(\cdot)$ be a nonnegative C^2 function with $V(0) = 0$ and define the finite horizon cost (from x using $u(\cdot)$) to be

$$J_T(x,u(\cdot)) = \int_0^T L(x^u(\tau;x),u(\tau))\,d\tau + V(x^u(T;x)) \tag{14.35}$$

and denote the optimal cost (from x) as

$$J_T^*(x) = \inf_{u(\cdot)} J_T(x,u(\cdot))\,.$$

As in the infinite horizon case, one can show, by geometric means, that $J_T^*(\cdot)$ is locally smooth (C^2). Other properties will depend on the choice of V and T.

RMM The next paragraph should be rewritten

Let Γ^∞ denote the domain of $J_\infty^*(\cdot)$ (the subset of $\mathbb{R}^n$ on which J_∞^* is finite). It is not too difficult to show that the cost functions $J_\infty^*(\cdot)$ and $J_T^*(\cdot)$, $T \geq 0$, are continuous functions on Γ_∞ [**?**]. For simplicity, we will allow $J_\infty^*(\cdot)$ to take values in the extended real line so that, for instance, $J_\infty^*(x) = +\infty$ means that there is no control taking x to the origin.

We will assume that f and L are such that the minimum value of the cost functions $J_\infty^*(x)$, $J_T^*(x)$, $T \geq 0$, is attained for each (suitable) x. That is, given x and $T > 0$ (including $T = \infty$ when $x \in \Gamma^\infty$), there is a (C^1 in t) optimal trajectory $(x_T^*(t;x),u_T^*(t;x))$, $t \in [0,T]$, such that $J_T(x,u_T^*(\cdot;x)) = J_T^*(x)$. For instance, if f is such that its trajectories can be bounded on finite intervals as a function of its input size, e.g., there is a continuous function β such that $\|x^u(t;x_0)\| \leq \beta(\|x_0\|,\|u(\cdot)\|_{L_1[0,t]})$, then (together with the conditions above) there will be a

minimizing control (cf. [**?**]). Many such conditions may be used to good effect; see [**?**] for a more complete discussion.

It is easy to see that $J_\infty^*(\cdot)$ is proper on its domain so that the sub-level sets

$$\Gamma_r^\infty := \{x \in \Gamma^\infty : J_\infty^*(x) \leq r^2\}$$

are compact and path connected and moreover $\Gamma^\infty = \bigcup_{r\geq 0} \Gamma_r^\infty$. Note also that Γ^∞ may be a proper subset of $\mathbb{R}^n$ since there may be states that cannot be driven to the origin. We use r^2 (rather than r) here to reflect the fact that our integral cost is quadratically bounded from below. We refer to sub-level sets of $J_T^*(\cdot)$ and $V(\cdot)$ using

$$\Gamma_r^T := \text{path connected component of } \{x \in \Gamma^\infty : J_T^*(x) \leq r^2\} \text{ containing } 0,$$

and

$$\Omega_r := \text{path connected component of } \{x \in \mathbb{R}^n : V(x) \leq r^2\} \text{ containing } 0.$$

These results provide the technical framework needed for receding horizon control.

Receding Horizon Control with CLF Terminal Cost

In receding horizon control, a finite horizon optimal control problem is solved, generating open-loop state and control trajectories. The resulting control trajectory is applied to the system for a fraction of the horizon length. This process is then repeated, resulting in a sampled data feedback law. Although receding horizon control has been successfully used in the process control industry for many years, its application to fast, stability-critical nonlinear systems has been more difficult. This is mainly due to two issues. The first is that the finite horizon optimizations must be solved in a relatively short period of time. Second, it can be demonstrated using linear examples that a naive application of the receding horizon strategy can have undesirable effects, often rendering a system unstable. Various approaches have been proposed to tackle this second problem; see [**?**] for a comprehensive review of this literature. The theoretical framework presented here also addresses the stability issue directly, but is motivated by the need to relax the computational demands of existing stabilizing RHC formulations.

Receding horizon control provides a practical strategy for the use of information from a model through on-line optimization. Every δ seconds, an optimal control problem is solved over a T second horizon, starting from the current state. The first δ seconds of the optimal control $u_T^*(\cdot; x(t))$ is then applied to the system, driving the system from $x(t)$ at current time t to $x_T^*(\delta, x(t))$ at the next sample time $t+\delta$ (assuming no model uncertainty). We denote this receding horizon scheme as $\mathscr{RH}(T,\delta)$.

In defining (unconstrained) finite horizon approximations to the infinite horizon problem, the key design parameters are the terminal cost function $V(\cdot)$ and the horizon length T (and, perhaps also, the increment δ). We wish to characterize the

RMM: This paragraph is sort of a repeat

sets of choices that provide successful controllers.

It is well known (and easily demonstrated with linear examples), that simple truncation of the integral (i.e., $V(x) \equiv 0$) may have disastrous effects if $T > 0$ is too small. Indeed, although the resulting value function may be nicely behaved, the "optimal" receding horizon closed loop system can be unstable.

RMM Example (RHP zero?): show no terminal cost/constraint followed by terminal constraint case

A more sophisticated approach is to make good use of a suitable terminal cost $V(\cdot)$. Evidently, the best choice for the terminal cost is $V(x) = J_\infty^*(x)$ since then the optimal finite and infinite horizon costs are the same. Of course, if the optimal value function were available there would be no need to solve a trajectory optimization problem. What properties of the optimal value function should be retained in the terminal cost? To be effective, the terminal cost should account for the discarded tail by ensuring that the origin can be reached from the terminal state $x^u(T;x)$ in an efficient manner (as measured by L). One way to do this is to use an appropriate control Lyapunov function, which is also an upper bound on the cost-to-go.

The following theorem shows that the use of a particular type of CLF is in fact effective, providing rather strong and specific guarantees.

Theorem 14.1. **[?]** *Suppose that the terminal cost $V(\cdot)$ is a control Lyapunov function such that*

$$\min_{u \in \mathbb{R}^m} (\dot{V} + L)(x,u) \leq 0 \tag{14.36}$$

for each $x \in \Omega_{r_v}$ for some $r_v > 0$. Then, for every *$T > 0$ and $\delta \in (0,T]$, the resulting receding horizon trajectories go to zero exponentially fast. For each $T > 0$, there is an $\bar{r}(T) \geq r_v$ such that $\Gamma^T_{\bar{r}(T)}$ is contained in the region of attraction of $\mathscr{RH}(T,\delta)$. Moreover, given any compact subset Λ of Γ^∞, there is a T^* such that $\Lambda \subset \Gamma^T_{\bar{r}(T)}$ for all $T \geq T^*$.*

Theorem 14.1 shows that for *any* horizon length $T > 0$ and *any* sampling time $\delta \in (0,T]$, the receding horizon scheme is exponentially stabilizing over the set $\Gamma^T_{r_v}$. For a given T, the region of attraction estimate is enlarged by increasing r beyond r_v to $\bar{r}(T)$ according to the requirement that $V(x^*_T(T;x)) \leq r_v^2$ on that set. An important feature of the above result is that, for operations with the set $\Gamma^T_{\bar{r}(T)}$, there is no need to impose stability ensuring constraints which would likely make the online optimizations more difficult and time consuming to solve.

Sketch of proof. Let $x^u(\tau;x)$ represent the state trajectory at time τ starting from initial state x and applying a control trajectory $u(\cdot)$, and let $(x^*_T, u^*_T)(\cdot,x)$ represent the optimal trajectory of the finite horizon, optimal control problem with horizon T. Assume that $x^*_T(T;x) \in \Omega_r$ for some $r > 0$. Then for any $\delta \in [0,T]$ we want to show that the optimal cost $x^*_T(\delta;x)$ satisfies

$$J^*_T\big(x^*_T(\delta;x)\big) \leq J^*_T(x) - \int_0^\delta q\big(L(x^*_T(\tau;x), u^*_T(\tau;x))\, d\tau. \tag{14.37}$$

This expression says that solution to the finite-horizon, optimal control problem starting at time $t = \delta$ has cost that is less than the cost of the solution from time $t = 0$, with the initial portion of the cost subtracted off.†. In other words, we are closer to our solution by a finite amount at each iteration of the algorithm. It follows using Lyapunov analysis that we must converge to the zero cost solution and hence our trajectory converges to the desired terminal state (given by the minimum of the cost function).

RMM: Add a picture like the one in Tamas Keviczky's CDS 270-2 notes

To show equation (14.37) holds, consider a trajectory in which we apply the optimal control for the first T seconds and then apply a closed loop controller using a stabilizing feedback $u = -k(x)$ for another T seconds. (The stabilizing compensator is guaranteed to exist since V is a control Lyapunov function.) Let $(x_T^*, u_T^*)(t;x)$, $t \in [0,T]$ represent the optimal control and $(x^k, u^k)(t-T; x_T^*(T;x))$, $t \in [T,2T]$ represent the control with $u = -k(x)$ applied where k satisfies $(\dot{V} + L)(x, -k(x)) \leq 0$. Finally, let $(\tilde{x}(t), \tilde{u}(t))$, $t \in [0,2T]$ represent the trajectory obtained by concatenating the optimal trajectory (x_T^*, u_T^*) with the CLF trajectory (x^k, u^k).

We now proceed to show that the inequality (14.37) holds. The cost of using $\tilde{u}(\cdot)$ for the first T seconds starting from the initial state $x_T^*(\delta;x))$, $\delta \in [0,,T]$ is given by

$$\begin{aligned} J_T(x_T^*(\delta;x), \tilde{u}(\cdot)) &= \int_\delta^{T+\delta} L(\tilde{x}(\tau), \tilde{u}(\tau))\, d\tau + V(\tilde{x}(T+\delta)) \\ &= J_T^*(x) - \int_0^\delta L(x_T^*(\tau;x), u_T^*(\tau;x))\, d\tau - V(x_T^*(T;x)) \\ &\qquad + \int_T^{T+\delta} L(\tilde{x}(\tau), \tilde{u}(\tau))\, d\tau + V(\tilde{x}(T+\delta)). \end{aligned}$$

Note that the second line is simply a rewriting of the integral in terms of the optimal cost J_T^* with the necessary additions and subtractions of the additional portions of the cost for the interval $[\delta, T+\delta]$. We can how use the bound

$$L(\tilde{x}(\tau), \tilde{u}(\tau)) \leq \dot{V}(\tilde{x}(\tau), \tilde{u}(\tau), \qquad \tau \in [T, 2T],$$

which follows from the definition of the CLF V and stabilizing controller $k(x)$. This allows us to write

$$\begin{aligned} J_T(x_T^*(\delta;x), \tilde{u}(\cdot)) &\leq J_T^*(x) - \int_0^\delta L(x_T^*(\tau;x), u_T^*(\tau;x))\, d\tau - V(x_T^*(T;x)) \\ &\qquad - \int_T^{T+\delta} \dot{V}(\tilde{x}(\tau), \tilde{u}(\tau))\, d\tau + V(\tilde{x}(T+\delta)) \\ &= J_T^*(x) - \int_0^\delta L(x_T^*(\tau;x), u_T^*(\tau;x))\, d\tau - V(x_T^*(T;x)) \\ &\qquad - V(\tilde{x}(\tau))\Big|_T^{T+\delta} + V(\tilde{x}(T+\delta)) \\ &= J_T^*(x) - \int_0^\delta L(x_T^*(\tau;x), u_T^*(\tau;x)). \end{aligned}$$

Finally, using the optimality of u_T^* we have that $J_T^*(x_T^*(\delta;x)) \leq J_T(x_T^*(\delta;x),\tilde{u}(\cdot))$ and we obtain equation (14.37). □

An important benefit of receding horizon control is its ability to handle state and control constraints. While the above theorem provides stability guarantees when there are no constraints present, it can be modified to include constraints on states and controls as well. In order to ensure stability when state and control constraints are present, the terminal cost $V(\cdot)$ should be a local CLF satisfying $\min_{u\in\mathscr{U}} \dot{V} + L(x,u) \leq 0$ where $\mathscr{U}$ is the set of controls where the control constraints are satisfied. Moreover, one should also require that the resulting state trajectory $x^{CLF}(\cdot) \in \mathscr{X}$, where $\mathscr{X}$ is the set of states where the constraints are satisfied. (Both $\mathscr{X}$ and $\mathscr{U}$ are assumed to be compact with origin in their interior). Of course, the set Ω_{r_v} will end up being smaller than before, resulting in a decrease in the size of the guaranteed region of operation (see [**?**] for more details).

Receding Horizon Control Using Differential Flatness

In this section we demonstrate how to use differential flatness to find fast numerical algorithms for solving the optimal control problems required for the receding horizon control results of the previous section. We consider the affine nonlinear control system

$$\dot{x} = f(x) + g(x)u, \tag{14.38}$$

where all vector fields and functions are smooth. For simplicity, we focus on the single input case, $u \in \mathbb{R}$. We wish to find a trajectory of equation (14.38) that minimizes the performance index (14.35), subject to a vector of initial, final, and trajectory constraints

$$\begin{aligned} lb_0 &\leq \psi_0(x(t_0),u(t_0)) \leq ub_0, \\ lb_f &\leq \psi_f(x(t_f),u(t_f)) \leq ub_f, \\ lb_t &\leq S(x,u) \leq ub_t, \end{aligned} \tag{14.39}$$

respectively. For conciseness, we will refer to this optimal control problem as

$$\min_{(x,u)} J(x,u) \qquad \text{subject to} \qquad \begin{cases} \dot{x} = f(x) + g(x)u, \\ lb \leq c(x,u) \leq ub. \end{cases} \tag{14.40}$$

Numerical Solution Using Collocation.

RMM Rewrite this as a more general description of collocation. Perhaps also add other methods of collocation.

A numerical approach to solving this optimal control problem is to use the direct collocation method outlined in Hargraves and Paris [**?**]. The idea behind this approach is to transform the optimal control problem into a nonlinear programming problem. This is accomplished by discretizing time into a grid of $N-1$ intervals

$$t_0 = t_1 < t_2 < \ldots < t_N = t_f \tag{14.41}$$

and approximating the state x and the control input u as piecewise polynomials $\tilde{x}$ and $\tilde{u}$, respectively. Typically a cubic polynomial is chosen for the states and a linear polynomial for the control on each interval. Collocation is then used at the midpoint of each interval to satisfy equation (14.38). Let $\tilde{x}(x(t_1),...,x(t_N))$ and $\tilde{u}(u(t_1),...,u(t_N))$ denote the approximations to x and u, respectively, depending on $(x(t_1),...,x(t_N)) \in \mathbb{R}^{nN}$ and $(u(t_1),...,u(t_N)) \in \mathbb{R}^N$ corresponding to the value of x and u at the grid points. Then one solves the following finite dimension approximation of the original control problem (14.40):

$$\min_{y\in\mathbb{R}^M} F(y) = J(\tilde{x}(y),\tilde{u}(y)) \qquad \text{subject to} \qquad \begin{cases} \dot{\tilde{x}} - f(\tilde{x}(y)) + g(\tilde{x}(y))\tilde{u}(y) = 0, \\ lb \le c(\tilde{x}(y),\tilde{u}(y)) \le ub, \\ \qquad \forall t = \dfrac{t_j + t_{j+1}}{2} \quad j = 1,\dots,N-1 \end{cases} \tag{14.42}$$

where $y = (x(t_1),u(t_1),\dots,x(t_N),u(t_N))$, and $M = \dim y = (n+1)N$.

Seywald [**?**] suggested an improvement to the previous method (see also [**?**, p. 362]). Following this work, one first solves a subset of system dynamics in equation (14.40) for the the control in terms of combinations of the state and its time derivative. Then one substitutes for the control in the remaining system dynamics and constraints. Next all the time derivatives $\dot{x}_i$ are approximated by the finite difference approximations

$$\dot{\tilde{x}}(t_i) = \frac{x(t_{i+1}) - x(t_i)}{t_{i+1} - t_i}$$

to get

$$\left.\begin{array}{l} p(\dot{\tilde{x}}(t_i),x(t_i)) = 0 \\ q(\dot{\tilde{x}}(t_i),x(t_i)) \le 0 \end{array}\right\} \quad i = 0,...,N-1.$$

The optimal control problem is turned into

$$\min_{y\in\mathbb{R}^M} F(y) \qquad \text{subject to} \qquad \begin{cases} p(\dot{\tilde{x}}(t_i),x(t_i)) = 0 \\ q(\dot{\tilde{x}}(t_i),x(t_i)) \le 0 \end{cases} \tag{14.43}$$

where $y = (x(t_1),\dots,x(t_N))$, and $M = \dim y = nN$. As with the Hargraves and Paris method, this parameterization of the optimal control problem (14.40) can be solved using nonlinear programming.

The dimensionality of this discretized problem is lower than the dimensionality of the Hargraves and Paris method, where both the states and the input are the unknowns. This induces substantial improvement in numerical implementation.

Differential Flatness Based Approach. The results of Seywald give a constrained optimization problem in which we wish to minimize a cost functional subject to $n-1$ equality constraints, corresponding to the system dynamics, at each time instant. In fact, it is usually possible to reduce the dimension of the problem further. Given an output, it is generally possible to parameterize the control and a part of the state in terms of this output and its time derivatives. In contrast to the previous approach, one must use more than one derivative of this output for this purpose.

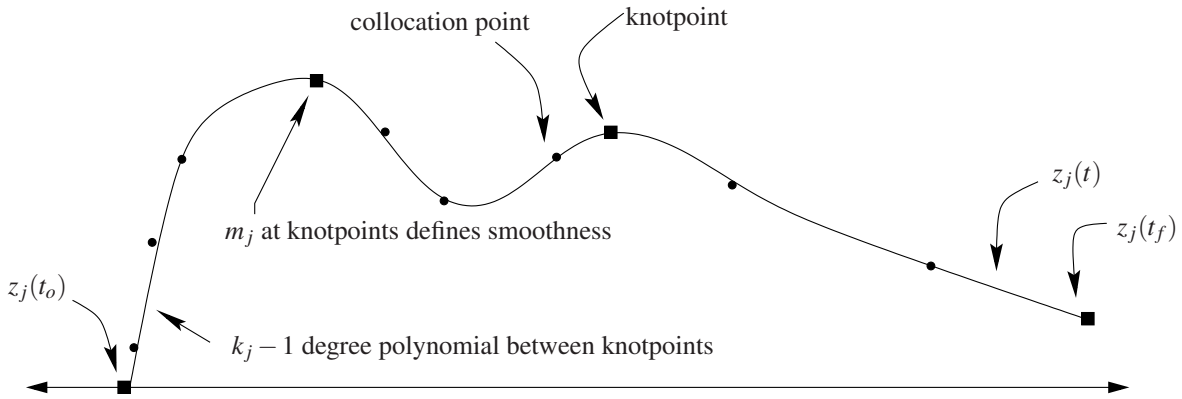


Figure 14.26: Spline representation of a variable.

When the whole state and the input can be parameterized with one output, the system is differentially flat, as described in Section 14.5. When the parameterization is only partial, the dimension of the subspace spanned by the output and its derivatives is given by r the *relative degree* of this output [Isi95]. In this case, it is possible to write the system dynamics as

$$\begin{aligned} x &= \alpha(z, \dot{z}, \ldots, z^{(q)}) \\ u &= \beta(z, \dot{z}, \ldots, z^{(q)}) \\ \Phi(z, \dot{z}, \ldots, z^{n-r}) &= 0 \end{aligned} \tag{14.44}$$

where $z \in \mathbb{R}^p$, $p > m$ represents a set of outputs that parameterize the trajectory and $\Phi : \mathbb{R}^n \times \mathbb{R}^m$ represents $n - r$ remaining differential constraints on the output. In the case that the system is flat, $r = n$ and we eliminate these differential constraints.

Unlike the approach of Seywald, it is not realistic to use finite difference approximations as soon as $r > 2$. In this context, it is convenient to represent z using B-splines. B-splines are chosen as basis functions because of their ease of enforcing continuity across knot points and ease of computing their derivatives. A pictorial representation of such an approximation is given in Figure 14.26. Doing so we get

$$z_j = \sum_{i=1}^{p_j} B_{i,k_j}(t) C_i^j, \qquad p_j = l_j(k_j - m_j) + m_j$$

where $B_{i,k_j}(t)$ is the B-spline basis function defined in [**?**] for the output z_j with order k_j, C_i^j are the coefficients of the B-spline, l_j is the number of knot intervals, and m_j is number of smoothness conditions at the knots. The set $(z_1, z_2, \ldots, z_{n-r})$ is thus represented by $M = \sum_{j \in \{1, r+1, \ldots, n\}} p_j$ coefficients.

In general, w collocation points are chosen uniformly over the time interval $[t_o, t_f]$ (though optimal knots placements or Gaussian points may also be considered). Both dynamics and constraints will be enforced at the collocation points.

The problem can be stated as the following nonlinear programming form:

$$\min_{y \in \mathbb{R}^M} F(y) \qquad \text{subject to} \qquad \begin{cases} \Phi(z(y), \dot{z}(y), \ldots, z^{(n-r)}(y)) = 0 \\ lb \leq c(y) \leq ub \end{cases} \tag{14.45}$$

where

$$y = (C_1^1, \ldots, C_{p_1}^1, C_1^{r+1}, \ldots, C_{p_{r+1}}^{r+1}, \ldots, C_1^n, \ldots, C_{p_n}^n).$$

The coefficients of the B-spline basis functions can be found using nonlinear programming.

A software package called Nonlinear Trajectory Generation (NTG) has been written to solve optimal control problems in the manner described above (see [**?**] for details). The sequential quadratic programming package NPSOL by [**?**] is used as the nonlinear programming solver in NTG. When specifying a problem to NTG, the user is required to state the problem in terms of some choice of outputs and its derivatives. The user is also required to specify the regularity of the variables, the placement of the knot points, the order and regularity of the B-splines, and the collocation points for each output.

Implementation on the Caltech Ducted Fan

To demonstrate the use of the techniques described in the previous section, we present an implementation of optimization-based control on the Caltech Ducted Fan, a real-time, flight control experiment that mimics the longitudinal dynamics of an aircraft. The experiment is show in Figure 14.27.

Description of the Caltech Ducted Fan Experiment. The Caltech ducted fan is an experimental testbed designed for research and development of nonlinear flight guidance and control techniques for Uninhabited Combat Aerial Vehicles (UCAVs). The fan is a scaled model of the longitudinal axis of a flight vehicle and flight test results validate that the dynamics replicate qualities of actual flight vehicles [**?**].

The ducted fan has three degrees of freedom: the boom holding the ducted fan is allowed to operate on a cylinder, 2 m high and 4.7 m in diameter, permitting horizontal and vertical displacements. A counterweight is connected to the vertical axis of the stand, allowing the effective mass of the fan to be adjusted. Also, the wing/fan assembly at the end of the boom is allowed to rotate about its center of mass. Optical encoders mounted on the ducted fan, counterweight pulley, and the base of the stand measure the three degrees of freedom. The fan is controlled by commanding a current to the electric motor for fan thrust and by commanding RC servos to control the thrust vectoring mechanism.

The sensors are read and the commands sent by a DSP-based multi-processor system, comprised of a D/A card, a digital I/O card, two Texas Instruments C40 signal processors, two Compaq Alpha processors, and a high-speed host PC interface. A real-time interface provides access to the processors and I/O hardware. The NTG software resides on both of the Alpha processors, each capable of running real-time optimization.

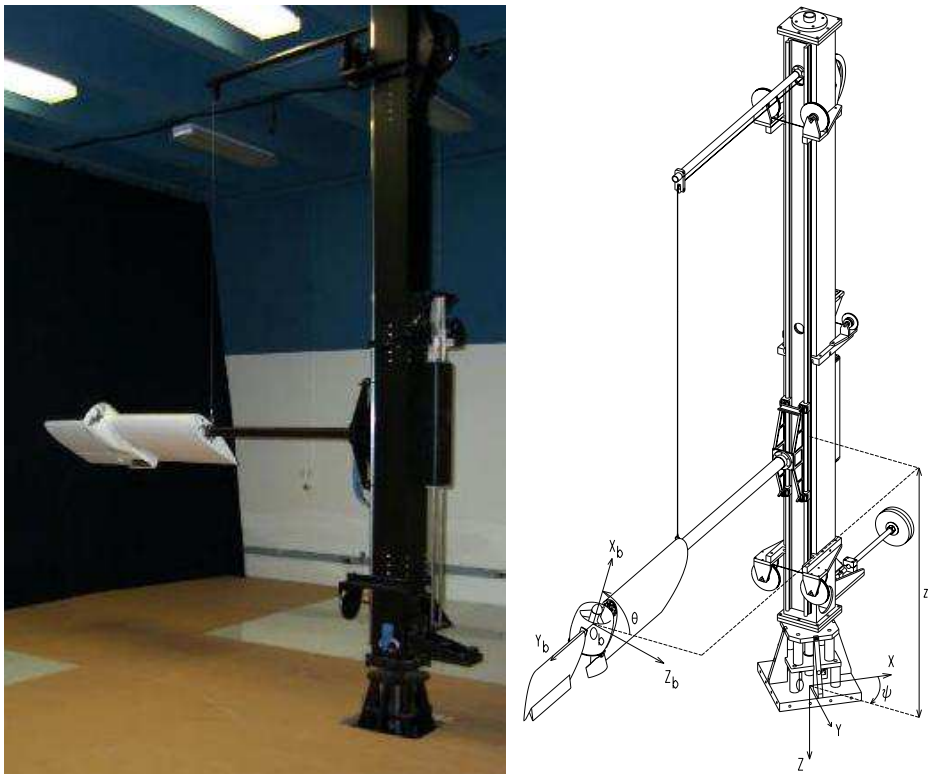


Figure 14.27: Caltech ducted fan.

The ducted fan is modeled in terms of the position and orientation of the fan, and their velocities. Letting x represent the horizontal translation, z the vertical translation and θ the rotation about the boom axis, the equations of motion are given by

$$\begin{aligned} m\ddot{x} + F_{X_a} - F_{X_b}\cos\theta - F_{Z_b}\sin\theta &= 0, \\ m\ddot{z} + F_{Z_a} + F_{X_b}\sin\theta - F_{Z_b}\cos\theta &= mg_{\text{eff}}, \\ J\ddot{\theta} - M_a + \frac{1}{r_s} I_p \Omega \dot{x}\cos\theta - F_{Z_b} r_f &= 0, \end{aligned} \tag{14.46}$$

where $F_{X_a} = D\cos\gamma + L\sin\gamma$ and $F_{Z_a} = -D\sin\gamma + L\cos\gamma$ are the aerodynamic forces and F_{X_b} and F_{Z_b} are thrust vectoring body forces in terms of the lift (L), drag (D), and flight path angle (γ). I_p and Ω are the moment of inertia and angular velocity of the ducted fan propeller, respectively. J is the moment of ducted fan and r_f is the distance from center of mass along the X_b axis to the effective application point of the thrust vectoring force. The angle of attack α can be derived from the pitch angle θ and the flight path angle γ by

$$\alpha = \theta - \gamma.$$

The flight path angle can be derived from the spatial velocities by

$$\gamma = \arctan \frac{-\dot{z}}{\dot{x}}.$$

The lift (L) ,drag (D), and moment (M) are given by

$$L = qSC_L(\alpha) \qquad D = qSC_D(\alpha) \qquad M = \bar{c}SC_M(\alpha),$$

respectively. The dynamic pressure is given by $q = \frac{1}{2}\rho V^2$. The norm of the velocity is denoted by V, S the surface area of the wings, and ρ is the atmospheric density. The coefficients of lift ($C_L(\alpha)$), drag ($C_D(\alpha)$) and the moment coefficient ($C_M(\alpha)$) are determined from a combination of wind tunnel and flight testing and are described in more detail in [**?**], along with the values of the other parameters.

RMM: Add figures? Parameter values?

Real-Time Trajectory Generation for the Ducted Fan. In this section we describe the implementation of the trajectory generation algorithms by using NTG to generate minimum time trajectories in real time. An LQR-based regulator is used to stabilize the system. We focus in this section on aggressive, forward flight trajectories. The next section extends the controller to use a receding horizon controller, but on a simpler class of trajectories.

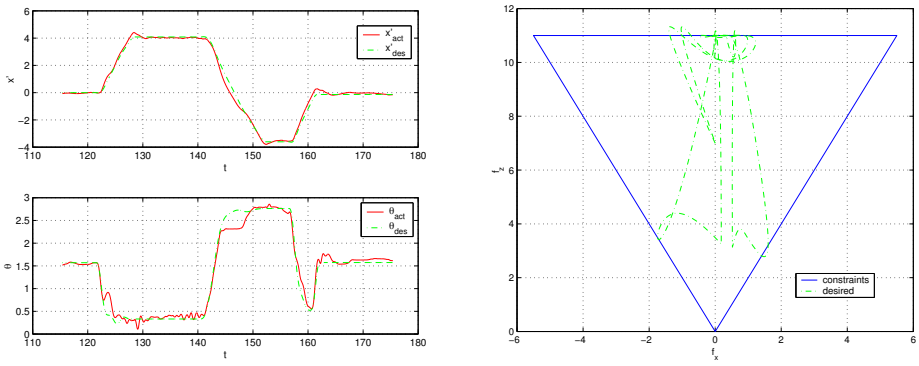
Stabilization Around Reference Trajectory The results in this section rely on the traditional two degree of freedom design paradigm described in Chapter **??**. In this approach, a local control law (inner loop) is used to stabilize the system around the trajectory computed based on a nominal model. This compensates for uncertainties in the model, which are predominantly due to aerodynamics and friction. Elements such as the ducted fan flying through its own wake, ground effects and velocity- and angle-of-attack† dependent thrust contribute to the aerodynamic uncertainty. Actuation models are not used when generating the reference trajectory, resulting in another source of uncertainty.

RMM: punctuation?

Since only the position of the fan is measured, we must estimate the velocities. We use an extended Kalman filter (described in later chapters) with the optimal gain matrix is gain scheduled on the (estimated) forward velocity.

The stabilizing LQR controllers were gain scheduled on pitch angle, θ, and the forward velocity, $\dot{x}$. The pitch angle was allowed to vary from $-\pi/2$ to $\pi/2$ and the velocity ranged from 0 to 6 m/s. The weights were chosen differently for the hover-to-hover and forward flight modes. For the forward flight mode, a smaller weight was placed on the horizontal (x) position of the fan compared to the hover-to-hover mode. Furthermore, the z weight was scheduled as a function of forward velocity in the forward flight mode. There was no scheduling on the weights for hover-to-hover. The elements of the gain matrices for each of the controller and observer are linearly interpolated over 51 operating points.

Nonlinear Trajectory Generation Parameters We solve a minimum time optimal control problem to generate a feasible trajectory for the system. The system is modeled using the nonlinear equations described above and computed



(a) System state (b) Input forces

Figure 14.28: Forward flight test case: (a) θ and $\dot{x}$ desired and actual, (b) desired F_{X_b} and F_{Z_b} with bounds.

the open loop forces and state trajectories for the nominal system. This system is not known to be differentially flat (due to the aerodynamic forces) and hence we cannot completely eliminate the differential constraints.

We choose three outputs, $z_1 = x$, $z_2 = z$, and $z_3 = \theta$, which results in a system with one remaining differential constraint. Each output is parameterized with four, sixth order C^4 piecewise polynomials over the time interval scaled by the minimum time. A fourth output, $z_4 = T$, is used to represent the time horizon to be minimized and is parameterized by a scalar. There are a total of 37 variables in this optimization problem. The trajectory constraints are enforced at 21 equidistant breakpoints over the scaled time interval.

There are many considerations in the choice of the parameterization of the outputs. Clearly there is a trade between the parameters (variables, initial values of the variables, and breakpoints) and measures of performance (convergence, runtime, and conservative constraints). Extensive simulations were run to determine the right combination of parameters to meet the performance goals of our system.

RMM Add a paragraph talking about the input and state constraints

Forward Flight To obtain the forward flight test data, an operator commanded a desired forward velocity and vertical position with joysticks. We set the trajectory update time δ to 2 seconds. By rapidly changing the joysticks, NTG produces high angle of attack maneuvers. Figure 14.28aa depicts the reference trajectories and the actual θ and $\dot{x}$ over 60 s. Figure 14.28b shows the commanded forces for the same time interval. The sequence of maneuvers corresponds to the ducted fan transitioning from near hover to forward flight, then following a command from a large forward velocity to a large negative velocity, and finally returning to hover.

Figure 14.29 is an illustration of the ducted fan altitude and x position for these maneuvers. The air-foil in the figure depicts the pitch angle (θ). It is apparent from

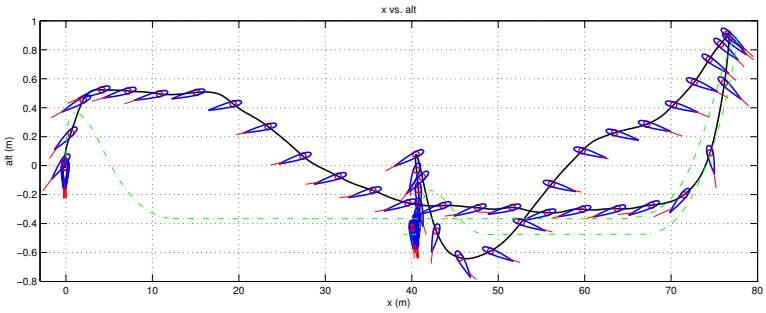


Figure 14.29: Forward flight test case: altitude and x position (actual (solid) and desired (dashed)). Airfoil represents actual pitch angle (θ) of the ducted fan.

this figure that the stabilizing controller is not tracking well in the z direction. This is due to the fact that unmodeled frictional effects are significant in the vertical direction. This could be corrected with an integrator in the stabilizing controller.

An analysis of the run times was performed for 30 trajectories; the average computation time was less than one second. Each of the 30 trajectories converged to an optimal solution and was approximately between 4 and 12 seconds in length. A random initial guess was used for the first NTG trajectory computation. Subsequent NTG computations used the previous solution as an initial guess. Much improvement can be made in determining a "good" initial guess. Improvement in the initial guess will improve not only convergence but also computation times.

RMM: Rewrite to talk about initial conditions

Receding Horizon Control. The results of the previous section demonstrate the ability to compute optimal trajectories in real time, although the computation time was not sufficiently fast for closing the loop around the optimization. In this section, we make use of a shorter update time δ, a fixed horizon time T with a quadratic integral cost, and a CLF terminal cost to implement the receding horizon controller described in Section 14.5. We also limit the operation of the system to near hover, so that we can use the local linearization to find the terminal CLF.

We have implemented the receding horizon controller on the ducted fan experiment where the control objective is to stabilize the hover equilibrium point. The quadratic cost is given by

$$\begin{aligned} L(x,u) &= \frac{1}{2}\hat{x}^T Q \hat{x} + \frac{1}{2}\hat{u}^T R \hat{u} \\ V(x) &= \gamma \hat{x}^T P \hat{x} \end{aligned} \tag{14.47}$$

where

$$\begin{aligned} \hat{x} &= x - x_{eq} = (x, z, \theta - \pi/2, \dot{x}, \dot{z}, \dot{\theta}) \\ \hat{u} &= u - u_{eq} = (F_{X_b} - mg, F_{Z_b}) \\ Q &= \mathrm{diag}\{4, 15, 4, 1, 3, 0.3\} \\ R &= \mathrm{diag}\{0.5, 0.5\}, \end{aligned}$$

For the terminal cost, we choose $\gamma = 0.075$ and P is the unique stable solution to

the algebraic Riccati equation corresponding to the linearized dynamics of equation (14.46) at hover and the weights Q and R. Note that if $\gamma = 1/2$, then $V(\cdot)$ is the CLF for the system corresponding to the LQR problem. Instead V is a relaxed (in magnitude) CLF, which achieved better performance in the experiment. In either case, V is valid as a CLF only in a neighborhood around hover since it is based on the linearized dynamics. We do not try to compute off-line a region of attraction for this CLF. Experimental tests omitting the terminal cost and/or the input constraints leads to instability. The results in this section show the success of this choice for V for stabilization. An inner-loop PD controller on $\theta, \dot{\theta}$ is implemented to stabilize to the receding horizon states $\theta_T^*, \dot{\theta}_T^*$. The θ dynamics are the fastest for this system and although most receding horizon controllers were found to be nominally stable without this inner-loop controller, small disturbances could lead to instability.

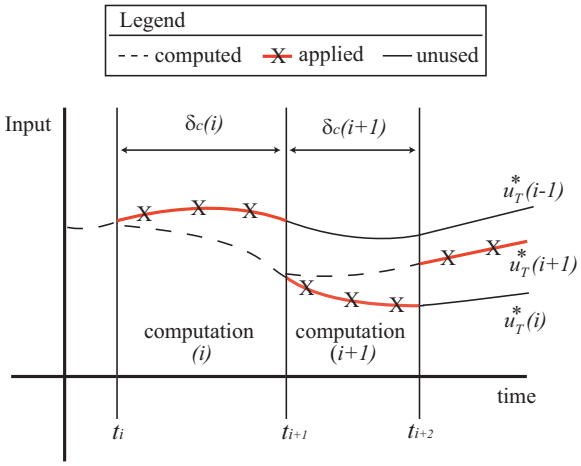
The optimal control problem is set-up in NTG code by parameterizing the three position states (x, z, θ), each with 8 B-spline coefficients. Over the receding horizon time intervals, 11 and 16 breakpoints were used with horizon lengths of 1, 1.5, 2, 3, 4 and 6 seconds. Breakpoints specify the locations in time where the differential equations and any constraints must be satisfied, up to some tolerance. The value of $F_{X_b}^{\max}$ for the input constraints is made conservative to avoid prolonged input saturation on the real hardware. The logic for this is that if the inputs are saturated on the real hardware, no actuation is left for the inner-loop θ controller and the system can go unstable. The value used in the optimization is $F_{X_b}^{\max} = 9$ N.

Computation time is non-negligible and must be considered when implementing the optimal trajectories. The computation time varies with each optimization as the current state of the ducted fan changes. The following notational definitions will facilitate the description of how the timing is set-up:

i	Integer counter of RHC computations
t_i	Value of current time when RHC computation i started
$\delta_c(i)$	Computation time for computation i
$u_T^*(i)(t)$	Optimal output trajectory corresponding to computation i, with time interval $t \in [t_i, t_i + T]$

A natural choice for updating the optimal trajectories for stabilization is to do so as fast as possible. This is achieved here by constantly resolving the optimization. When computation i is done, computation $i+1$ is immediately started, so $t_{i+1} = t_i + \delta_c(i)$. Figure 14.30 gives a graphical picture of the timing set-up as the optimal input trajectories $u_T^*(\cdot)$ are updated. As shown in the figure, any computation i for $u_T^*(i)(\cdot)$ occurs for $t \in [t_i, t_{i+1}]$ and the resulting trajectory is applied for $t \in [t_{i+1}, t_{i+2}]$. At $t = t_{i+1}$ computation $i+1$ is started for trajectory $u_T^*(i+1)(\cdot)$, which is applied as soon as it is available $(t = t_{i+2})$. For the experimental runs detailed in the results, $\delta_c(i)$ is typically in the range of $[0.05, 0.25]$ seconds, meaning 4 to 20 optimal control computations per second. Each optimization i requires the current measured state of the ducted fan and the value of the previous optimal input trajectories $u_T^*(i-1)$ at time $t = t_i$. This corresponds to, respectively, 6 initial

RMM: Perhaps cite Dunbar version (probably below)



RMM: Expand caption

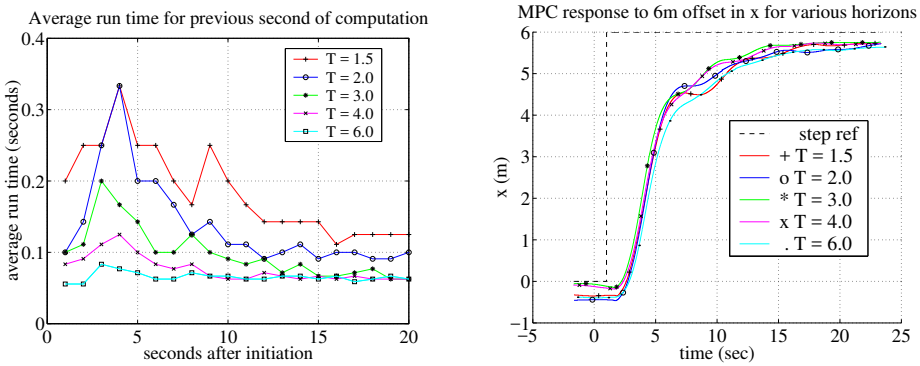
Figure 14.30: Receding horizon input trajectories.

conditions for state vector x and 2 initial constraints on the input vector u. Figure 14.30 shows that the optimal trajectories are advanced by their computation time prior to application to the system. A dashed line corresponds to the initial portion of an optimal trajectory and is not applied since it is not available until that computation is complete. The figure also reveals the possible discontinuity between successive applied optimal input trajectories, with a larger discontinuity more likely for longer computation times. The initial input constraint is an effort to reduce such discontinuities, although some discontinuity is unavoidable by this method. Also note that the same discontinuity is present for the 6 open-loop optimal state trajectories generated, again with a likelihood for greater discontinuity for longer computation times. In this description, initialization is not an issue because we assume the receding horizon computations are already running prior to any test runs. This is true of the experimental runs detailed in the results.

The experimental results show the response of the fan with each controller to a 6 meter horizontal offset, which is effectively engaging a step-response to a change in the initial condition for x. The following details the effects of different receding horizon control parameterizations, namely as the horizon changes, and the responses with the different controllers to the induced offset.

The first comparison is between different receding horizon controllers, where time horizon is varied to be 1.5, 2.0, 3.0, 4.0 or 6.0 seconds. Each controller uses 16 breakpoints. Figure 14.31a shows a comparison of the average computation time as time proceeds. For each second after the offset was initiated, the data correspond to the average run time over the previous second of computation. Note that these computation times are substantially smaller than those reported for real-time trajectory generation, due to the use of the CLF terminal cost versus the terminal constraints in the minimum-time, real-time trajectory generation experiments.

There is a clear trend toward shorter average computation times as the time



(a) Average run time (b) Step responses

Figure 14.31: Receding horizon control: (a) moving one second average of computation time for RHC implementation with varying horizon time, (b) response of RHC controllers to 6 meter offset in x for different horizon lengths.

horizon is made longer. There is also an initial transient increase in average computation time that is greater for shorter horizon times. In fact, the 6 second horizon controller exhibits a relatively constant average computation time. One explanation for this trend is that, for this particular test, a 6 second horizon is closer to what the system can actually do. After 1.5 seconds, the fan is still far from the desired hover position and the terminal cost CLF is large, likely far from its region of attraction. Figure 14.31b shows the measured x response for these different controllers, exhibiting a rise time of 8–9 seconds independent of the controller. So a horizon time closer to the rise time results in a more feasible optimization in this case.

State Estimation and Sensor Fusion

In this chapter we consider the problem of combining the data from different sensors to obtain an estimate of a (common) dynamical system. Unlike the previous chapters, we focus here on discrete-time processes, leaving the continuous-time case to the exercises. We begin with a summary of the input/output properties of discrete-time systems with stochastic inputs, then present the discrete-time Kalman filter, and use that formalism to formulate and present solutions for the sensor fusion problem. Some advanced methods of estimation and fusion are also summarized at the end of the chapter that demonstrate how to move beyond the linear, Gaussian process assumptions.

Sensor Fusion

We now return to the main topic of the chapter: sensor fusion. Consider the situation described in Figure 14.32, where we have an input/output dynamical system with multiple sensors capable of taking measurements. The problem of sensor fusion involves deciding how to best combine the measurements from the individual

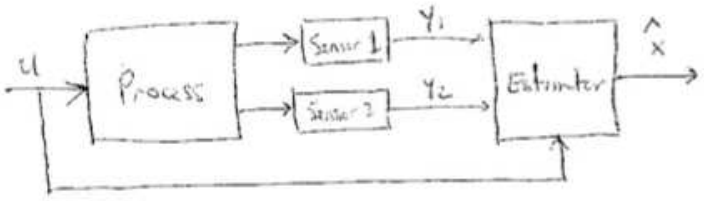


Figure 14.32: Sensor fusion

sensors in order to accurately estimate the process state X. Since different sensors may have different noise characteristics, evidently we should combine the sensors in a way that places more weight on sensors with lower noise. In addition, in some situations we may have different sensors available at different times, so that not all information is available on each measurement update.

To gain more insight into how the sensor data are combined, we investigate the functional form of $L[k]$. Suppose that each sensor takes a measurement of the form

$$Y^i = C^i X + V^i, \qquad i = 1, \ldots, p,$$

where the superscript i corresponds to the specific sensor. Let V^i be a zero mean, white noise process with covariance $\sigma_i^2 = R_{V^i}(0)$. It follows from Lemma **??** that

$$L[k] = P[k|k]C^T R_W^{-1}.$$

First note that if $P[k|k]$ is small, indicating that our estimate of X is close to the actual value (in the MMSE sense), then $L[k]$ will be small due to the leading $P[k|k]$ term. Furthermore, the characteristics of the individual sensors are contained in the different σ_i^2 terms, which only appears in R_W. Expanding the gain matrix, we have

$$L[k] = P[k|k]C^T R_W^{-1}, \qquad R_W^{-1} = \begin{pmatrix} 1/\sigma_1^2 & & \\ & \ddots & \\ & & 1/\sigma_p^2 \end{pmatrix}.$$

We see from the form of R_W^{-1} that each sensor is inversely weighted by its covariance. Thus noisy sensors ($\sigma_i^2 \gg 1$) will have a small weight and require averaging over many iterations before their data can affect the state estimate. Conversely, if $\sigma_i^2 \ll 1$, the data is "trusted" and is used with higher weight in each iteration.

Work through the case with dynamics. **RMM**

Information Filters

An alternative formulation of the Kalman filter is to make use of the inverse of the covariance matrix, called the *information matrix*, to represent the error of the estimate. It turns out that writing the state estimator in this form has several advan-

tages both conceptually and when implementing distributed computations. This form of the Kalman filter is known as the *information filter*.

We begin by defining the information matrix I and the weighted state estimate $\hat{Z}$:

$$I[k|k] = P^{-1}[k|k], \qquad \hat{Z}[k|k] = P^{-1}[k|k]\hat{X}[k|k].$$

RMM: Need to sort out subscripts versus superscripts

We also make use of the following quantities, which appear in the Kalman filter equations:

$$\Omega_i[k|k] = (C^i)^T R_{W^i}^{-1}[k|k]C^i, \qquad \Psi_i[k|k] = (C^i)^T R_{W^i}^{-1}[k|k]C^i\hat{X}[k|k].$$

Using these quantities, we can rewrite the Kalman filter equations as

Prediction	Correction
$I[k\|k-1] = \left(AI^{-1}[k-1\|k-1]A^T + R_W\right)^{-1},$	$I[k\|k] = I[k\|k-1] + \sum_{i=1}^{p} \Omega_i[k\|k],$
$\hat{Z}[k\|k-1] = I[k\|k-1]AI^{-1}[k-1\|k-1]\hat{Z}[k-1\|k-1] + Bu[k-1],$	$\hat{Z}[k\|k] = \hat{Z}[k\|k-1] + \sum_{i=1}^{p} \Psi_i[k\|k].$

RMM: These equations need to be checked and also reformatted

Note that these equations are in a particularly simple form, with the information matrix being updated by each sensor's Ω_i and similarly the state estimate being updated by each sensors Ψ_i.

Remarks:

1. Information form allows simple addition for correction step. Intuition: add information through additional data.
2. Sensor fusion: information content = inverse covariance (for each sensor)
3. Variable rate: incorporate new information whenever it arrives. No data $\Longrightarrow$ no information $\Longrightarrow$ prediction update only.

RMM See EECI notes and NCS book for more details

Additional topics

Converting continuous time stochastic systems to discrete time.

$$\begin{aligned}
\dot{X} &= AX + Bu + Fw \\
x(t+h) &\approx x(t) + h\dot{x}(t) \\
&= x(t) + hAx(t) + hBu(t) + hFW(t) \\
&= (I + hA)X(t) + (hB)u(t) + (hF)W(t) \\
X[k+1] &= \underbrace{(I+hA)}_{\tilde{A}} X[k] + \underbrace{(bB)}_{\tilde{B}} u[k] + \underbrace{(hF)}_{\tilde{F}} W[k].
\end{aligned}$$

Correlated disturbances and noise.

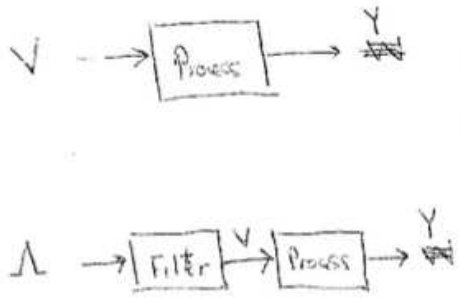


Figure 14.33: Sensor fusion with correlated measurement noise

As in the case of continuous-time Kalman filters, in the discrete time we can include noise or disturbances that are non-white by using a filter to generate noise with the appropriate correlation function.

On practical method to do this is to collect samples $W[1], W[2], \ldots, W[N]$ and then numerically compute the correlation function

$$R_W(l) = E\{W[i]W[i+l]\} = \frac{1}{N-l}\sum_{j=1}^{N-l} W[j]W[j+l].$$

Particle Filters

Discrete-decision making and supervisory control [from US]

Introduction: discrete actions and system modes

Finite transition systems

Temporal logic specifications

Model checking

Correct-by-construction synthesis

Linking Continuous and Discrete Controllers [following US]

Problem description

Tool: simulation relations

Approach: discrete abstraction

Cooperative Control for Autonomous Systems

This paper describes a framework for building cooperative control systems and summarizes recent results in formation stability using graph Laplacians, distributed optimization for cooperative control and protocols for decentralized decision-making. Additional results on networked estimation across packet dropping channels are

also discussed briefly. Applications include multi-vehicle systems performing cooperative tasks and autonomous systems with high-performance, distributed processing.

Introduction

Research on control of multi-vehicle systems performing cooperative tasks dates back to the late 1980s, initially beginning in the field of mobile robotics (see [**?**] for a more detailed history). Aided by the development of inexpensive and reliable wireless communications systems, research in this area increased substantially in the 1990s. California's Partners for Advanced Transit and Highways (PATH) project [**?**] demonstrated multiple automobiles driving together in "platoons" and this was quickly followed by other highway automation projects [**?**, **?**]. In the late 1990s and early 2000s, cooperative control of multiple aircraft, especially unmanned aerial vehicles (UAVs), became a highly active research area in the United States [**?**], spurring further advances. Over the last decade this research area has blossomed, with many new systems being proposed in application areas ranging from military battle systems to mobile sensors networks to commercial highway and air transportation systems.

Today, increases in fast and inexpensive computing and communications have enabled a new generation information-rich control systems that rely on multi-threaded networked execution, distributed optimization, adaptation and learning, and contingency management in increasingly sophisticated ways. The applications of the systems have extended beyond cooperative control of robotic vehicles and include applications such as load balancing in computer networks, distributed electric power generation and manufacturing systems and supply chains. In this paper we focus primarily on applications in robotics and autonomy; for a broader overview of applications see [**?**] and the references therein.

As an example of the type of system that we would like to study, we consider the *RoboFlag* game developed at Cornell [**?**], which is loosely based on "Capture the Flag" and "Paintball". Two teams play the game, the red team and the blue team, as depicted in Figure 14.34. The red team's objective is to infiltrate the blue team's territory, grab the blue flag, and bring it back to the red home zone. At the same time, the blue team's objective is to infiltrate the red team's territory, grab the red flag, and bring it back to the blue home zone. The game is thus a mix of offense and defense: capture the opponent's flag while at the same time preventing the opponent from capturing your flag. Sensing and communications are both limited to provide a more realistic distributed computing environment. The game is meant to provide an example of multi-vehicle, semi-autonomous systems operating in dynamic, uncertain, and adversarial environments. Human operators can also be present in the system and can be used either as high level controllers or as low level (remote) "pilots". A centralized control unit may be used coordinate the vehicles, but it must respect the communication constraints (bandwidth and latency) of the system.

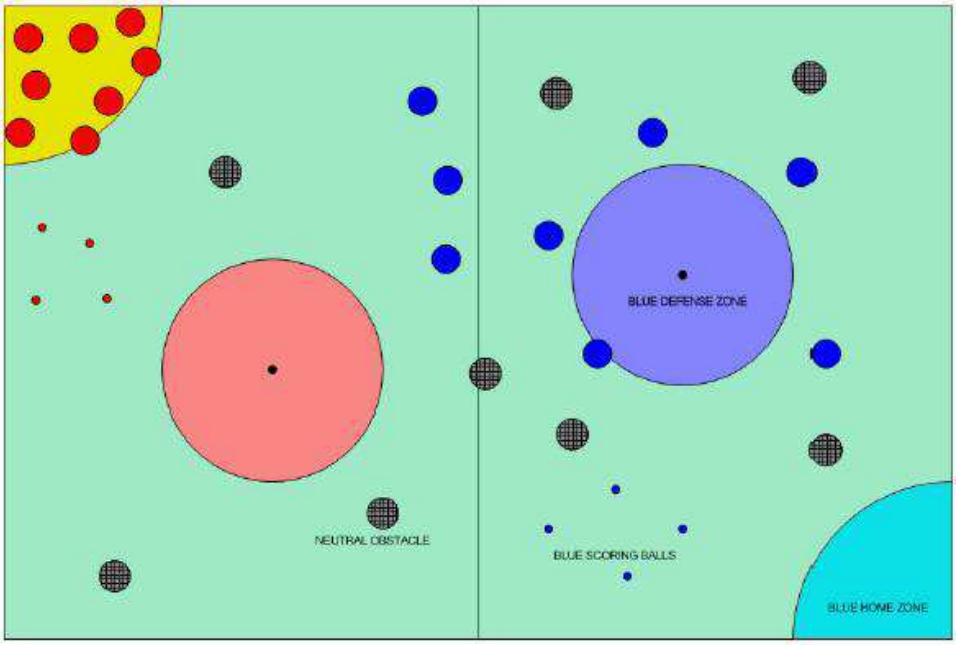


Figure 14.34: The RoboFlag playing field [**?**].

To study this problem and ones like it, we will first describe a mathematical formulation that attempts to capture some of the essential elements of the problem. We then survey some recent contributions to the networked control theory based on the work of the author at Caltech. An earlier version of some of the information in this paper, with more detailed descriptions of applications and work from other groups, is contained in [**?**].

A Mathematical Framework for Cooperative Control

We consider a set of vehicles with the dynamics of the ith vehicle written as

$$\begin{aligned} \dot{x}^i &= f^i(x^i, u^i) \quad x^i \in \mathbb{R}^n, u^i \in \mathbb{R}^m \\ \dot{y}^i &= h^i(x^i) \qquad y^i \in SE(3), \end{aligned} \tag{14.48}$$

where x^i is the state of the ith vehicle, u^i is the control input for the vehicle's and f^i is a smooth vector field representing its dynamics. We assume that the location of the vehicle is given by the output $y^i \in SE(3)$ (position and orientation). We let $x = (x^1, \ldots, x^N)$ represent the complete state for a collection of N vehicles.

We also assume that each vehicle has a discrete state, α^i, which we define as the *role* (or mode) of the vehicle. The role of the vehicle will be represented as an element of a discrete set $\mathscr{A}$. We consider the role variable α^i to represent the portion of the vehicle's overall state that encodes its current actions and its relationship with the overall task being performed. We will assume that the role of

a vehicle can change at any time and we will write a change of role as

$$\alpha' = r(x, \alpha),$$

where α' indicates the new value of α. We let $\alpha = (\alpha^1, \ldots, \alpha^N)$ represent the roles of the collection of N vehicles and write $\alpha^i(t)$ for the role of vehicle i at time t.

We model the communications between vehicles by a graph $\mathcal{G}$ whose nodes of the graph represent the individual vehicles and a directed edge between two nodes represents the ability of a vehicle to receive information from another vehicle. We write $\mathcal{N}^i(\mathcal{G})$ to represent the neighbors of vehicle i. In general, $\mathcal{N}^i$ can depend on the locations and roles of the vehicles, in which case we will write $\mathcal{N}^i(x, \alpha)$. The number of neighbors of the ith vehicle is given by the number of elements of $\mathcal{N}^i$, written $|\mathcal{N}^i|$.

In addition to limits on the agents that can talk to each other, we also allow for the possibility of packet loss along the channels. We model this packet loss using a simple indicator variable $\gamma^{i,j}$ which indicates at any given time whether data is transmitted across the link. The channel transmission characteristics can either be fixed, $\gamma^{i,j}(t)$ is a binary random process independent of the network traffic, or dependent on the environment, $\gamma^{i,j}(t) = \gamma^{i,j}(r(t), y(t))$ where r is the vector of transmission rates across each edge in the communications graph and the y dependence allows the probability of a dropped packet to depend on the physical location of the vehicles.

Given a collection of vehicles with state x and roles α, we will define a *task* in terms of a performance function

$$J = \int_0^T L(x, \alpha, u)\, dt + V(x(T), \alpha(T)),$$

where T is the horizon time over which the task should be accomplished, L represents the incremental cost of the task and V represents the terminal cost of the task. As special cases, we can take $T = \infty$ to represent infinite horizon problems or take $L = 0$ to represent tasks in which we are only interested in the final state. We may also have constraints on the states or inputs, but we omit these here for simplicity.

A *strategy* for a task is an assignment of the inputs u^i for each vehicle and a selection of the roles of the vehicles. We focus on *decentralized strategies* in which the vehicles are only able to use information from their own state and the communicated position and mode from neighbor vehicles. The control law has the form

$$\begin{aligned} u^i(x, \alpha) &= u^i(x^i, \alpha^i, x^{-i}, \alpha^{-i}, \gamma^{-i}) \\ \{g^i_j(x, \alpha) : r^i_j(x, \alpha)\} &= \{g^i_j(x^i, \alpha^i, x^{-i}, \alpha^{-i}) : \\ &\qquad r^i_j(x^i, \alpha^i, x^{-i}, \alpha^{-i})\}, \end{aligned}$$

where we use the shorthand x^{-i}, α^{-i} and γ^{-1} to represent the location, roles and channel status of vehicle i's neighbors (hence $x^{-i} = \{x^{j_1}, \ldots, x^{j_{m_i}}$ where $j_k \in \mathcal{N}^i$ and $m_i = |\mathcal{N}^i|.\}$.

The discrete update laws are written in terms of a guard g^i_j that evaluates to

either true or false and r_j^i is a rule that defines how the role α^i should be updated if the rule evaluates to true. Thus, the role evolves according to the update law

$$\alpha^{i\prime} = \begin{cases} r_j^i(x,\alpha) & g(x,\alpha) = \text{true} \\ \text{unchanged} & \text{otherwise.} \end{cases}$$

This update is allowed to happen asynchronously, although in practice it may be assigned by a central agent in the system, in which case it may evolve in a more regular fashion.

Formation control using graph Laplacians

We consider first the problem of stabilization a collection of agents who share information along a communication graph with fixed topology. For simplicity we assume that the agents' dynamics are linear and governed by

$$\begin{aligned} \dot{x}^i &= Ax^i + Bu^i, \\ y^i &= Cx^i. \end{aligned} \tag{14.49}$$

Fax [**?**] considers a control law in which each system attempts to stabilize itself relative to its neighbors. This is accomplished by constructing an error for each system that is a weighted combination of the relative outputs of the neighbors:

$$e^i = \sum_{j \in \mathcal{N}^i} \alpha_{ij}(y^j - y^i), \tag{14.50}$$

where α_{ij} is the relative weight. For simplicity, we consider uniform weighting here, so that $\alpha_{ij} = 1/|\mathcal{N}^i|$ where $\mathcal{N}^i$ is the number of neighbors of node i. The results are easily extended to the more general case.

Given the error (14.50), we apply a compensator that attempts to stabilize the overall system. For simplicity, we assume here that the stabilizer is given by a constant gain

$$u^i = Ke^i, \tag{14.51}$$

with $K \in \mathbb{R}^{m\times m}$ representing the compensation (gain) matrix. In practice, one can use a dynamic compensator to improve performance, but for analysis purposes we can just assume these dynamics are included in the system dynamics (14.49).

The interconnectedness of the system, represented by the neighbor sets $\mathcal{N}_i$ can be studied using tools from graph theory. In particular, for the the case of uniform weighting of the errors, it turns out that the combined error vector $e \in \mathbb{R}^{N\cdot m}$ can be written as

$$e = (\bar{L} \otimes I)x \tag{14.52}$$

where $\otimes$ represents the Kronecker product and $\bar{L}$ is the *weighted Laplacian* associated with the (directed) graph that models the neighbors of each node. The weighted Laplacian is a standard object in graph theory and can be defined as

$$\bar{L} = D^{-1}(D - A)$$

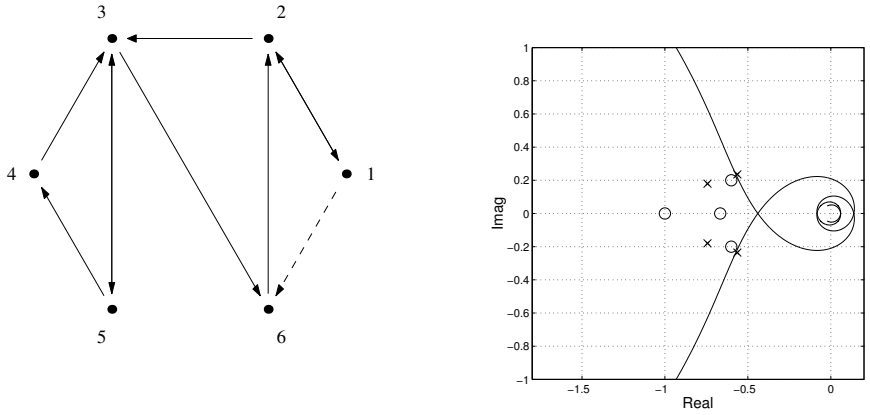


Figure 14.35: Interpretation of Theorem 1. The left figure shows the graph representation of the interconnected system and the right figure shows the corresponding Nyquist test. The addition of the dashed line to the graph moves the negative, inverse eigenvalues of $\bar{L}$ from the positions marked by circles to those marked by crosses.

where D is a diagonal matrix whose entries are the out-degree of each node and A is the adjacency matrix for the graph (see [**?**] for more detail). Using this framework, Fax showed the following:

Theorem 14.2. *A local controller K stabilizes the formation dynamics in equation (14.49) with error (14.52) and gain K if and only if it stabilizes the set of N systems given by*

$$\begin{aligned} \dot{x} &= Ax + B \cdot \lambda_i \cdot (Ky) \\ y &= Cx \end{aligned} \tag{14.53}$$

where λ_i are the eigenvalues of the weighted graph Laplacian $\bar{L}$.

This theorem has a very natural interpretation in in terms of the Nyquist plot of dynamical system. In the standard Nyquist criterion, one checks for stability of a feedback system by plotting the open loop frequency response of the system in the complex plane and checking for net encirclements of the -1 point. The conditions in Theorem 1 correspond to replacing the -1 point with $-1/\lambda_i$ for each eigenvalue λ_i of $\bar{L}$. This interpretation is illustrated in Figure 14.35. The results can easily be extended to consider weightings that are nonuniform.

Theorem 14.2 illustrates how the *dynamics* of the system, as represented by equation (14.49), interacts with the *information flow* of the system, as represented by the graph Laplacian. In particular, we see that it is the eigenvalues of the Laplacian that are critical for determining stability of the overall system. Additional results in this framework allow tuning of the information flow (considered as both sensed and communicated signals) to improve the transient response of the system [**?**]. Extensions in a stochastic setting [**?**, **?**] allow analysis of interconnected

systems whose dynamics are not identical and where the graph topology changes over time.

Distributed optimization

Another way to approach the formation control problem is to formulate it as an optimization problem. If we let $L^i(x^i, x^{-i})$ represent the individual formation error between the ith vehicle and its neighbors, then we can establish a cost function

$$L(x, \alpha, u) = \sum L^i(x^i, x^{-i}) + \|u^i\|_R^2,$$

where the summation over the individual formation errors gives the *cumulative formation error* [**?**] and the final term is a penalty on the inputs (other forms could be used).

This problem can be solved in either a centralized manner or a distributed manner. One distributed approach is the work of Dunbar *et al.* [**?**], who considers cooperative control problems using receding horizon optimal control. For a cost function whose coupling reflects the communication constraints of the vehicles, they generate distributed optimal control problems for each subsystem and establishes that the distributed receding horizon implementation is asymptotically stabilizing. The communication requirements between subsystems with coupling in the cost function are that each subsystem obtain the previous optimal control trajectory of those subsystems at each receding horizon update. The key requirements for stability are that each distributed optimal control not deviate too far from the previous optimal control, and that the receding horizon updates happen sufficiently fast.

The specific optimization problem that is solved by each vehicle has the form

$$\begin{aligned}
&\min_{u^i(\cdot)} \left\{ \int_{t_k}^{t_k+T} L^i(x^i(\tau), x^{-i}(\tau), u^i(\tau)) d\tau + V^i(x^i(t_k+T)) \right\} \\
&\text{subject to} \quad \dot{x}^i = f(x^i, u^i) \\
&\qquad\qquad\quad u^i \in U^i, \quad x^i(t_k+T) \in X_f^i \\
&\qquad\qquad\quad \|x^i(t) - \hat{x}^i(t)\| \leq \delta^2 \kappa,
\end{aligned}$$

where U^f and X_f^i are constraints on the input and final state, $\hat{x}^i$ is the trajectory last sent to vehicle i's neighbors, and δ and κ are parameters that control how much variation in the trajectory is allowed between iterations. Figure 14.36 shows a simulation of Dunbar's results. The vehicles are flying in "fingertip formation", with vehicles 2 and 3 maintaining position relative to vehicle 1 and vehicle 4 maintaining position relative to vehicle 2. The control goal is to maintain formation around the black square, which is flying along a trajectory that is not known to the individual aircraft. The localized optimization for each vehicle uses a previous optimal path for its neighbors while constraining its own path to stay near the previous path that it communicated to others.

A natural question in the context of distributed optimization is that of what tasks can be decomposed into a decentralized strategy. This problem has been

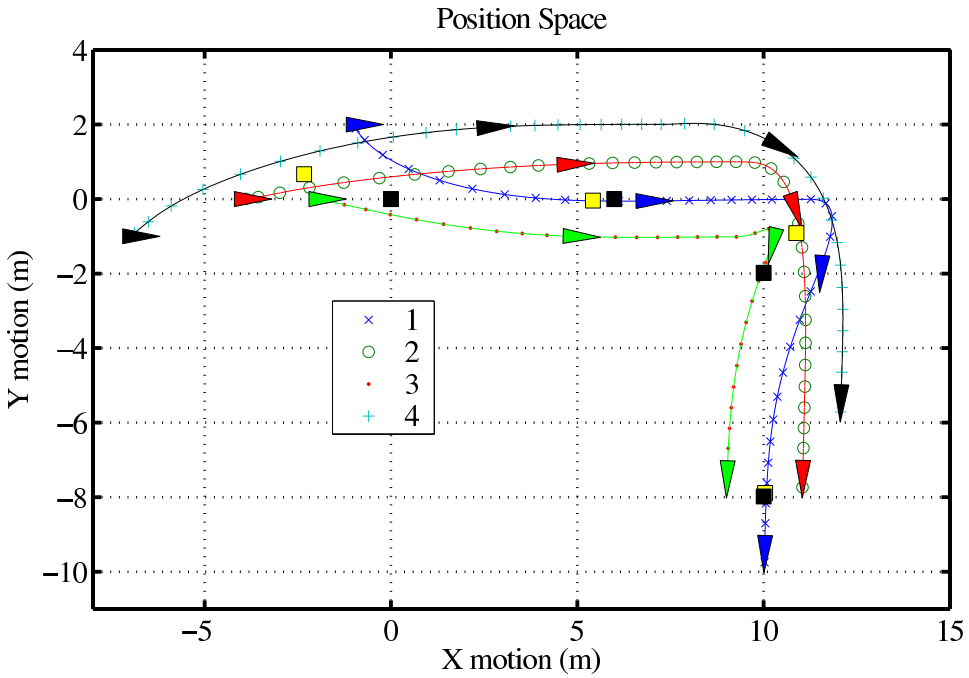


Figure 14.36: Four vehicle formation using distributed receding horizon control [**?**].

recently studied by Spanos [**?**], who presented a formalism for modeling dynamic coordination in networks with local communication capabilities encoded as a set of vectorial subspaces. This representation is capable of modeling many common network interaction patterns including global control, pairwise neighbor interactions, and multi-hop message routing. Within this framework, coordination is modeled as requirements that certain functions either decrease or remain invariant under the dynamics of the network, and that these requirements be locally verifiable in each of the subspaces comprising the network model.

Distributed protocols

The final problem that we present is the incorporation of protocols (or logic) into the solution. We model protocols using the guarded command formalism described in Section 14.5 and presented in more detail in [**?**]. A decentralized protocol is a description of the mode dynamics in the form

$$\alpha^{i\prime} = \begin{cases} r^i_j(x^i, x^{-i}, \alpha^i, \alpha^{-i}) & g(x,\alpha) = \text{true} \\ \text{unchanged} & \text{otherwise}, \end{cases}$$

where $\alpha^{i\prime}$ represents the new value of the mode variables for agent i. Reasoning about such protocols is tricky because we make no assumptions about when individual rules can be tested. Hence the commands can be executed in any order, at

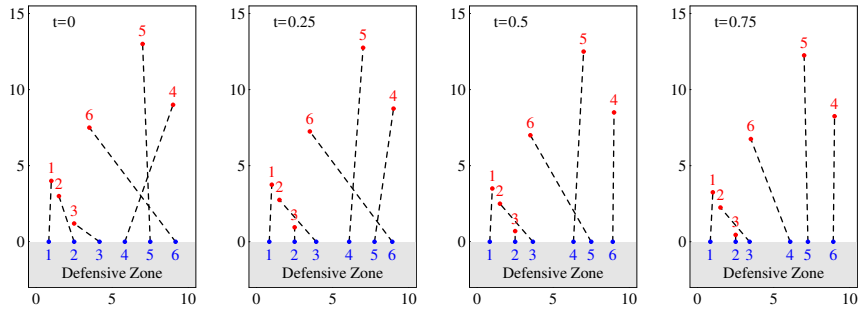


Figure 14.37: The RoboFlag Drill.

any time. We will assume one constraint on the dynamics, namely that all rules are evaluated no more than once before any rule is evaluated twice.

One implementation of this formalism is the computation and control language (CCL) developed by Klavins [**?**]. In this formalism, stability is determined by constructing Lyapunov function $V(x,\alpha)$ that have the property that $V(x,\alpha) \geq 0$ for all x and α, $V(x,\alpha) = 0$ only when the state and mode are at a desired value and for any execution cycle no allowable transitions increase V and at least one transition decreases V. The resulting V has the essential properties of a Lyapunov function and we can use the same intuition as with Lyapunov functions to show that the state (x,α) eventually converges to the desired state.

Figure 14.37 gives an example of how a distributed area denial task can be solved in CCL. In this example, drawn from the RoboFlag game, 6 defensive robots are trying to protect a defense zone for an incoming set of robots, which descend vertically at a fixed speed. The defending robots must move underneath the incoming robots, but are not allowed to run into each other. The defenders are randomly assigned incoming robots and are allowed to talk to their neighbors and switch assignments under a given protocol. A protocol was developed in [**?**] that is able to provably solve this problem, including insuring that no two robots collide and that all defensive robots eventually end up assigned to an incoming robot with now crossing of assignments.

The desired properties for the closed loop systems are described using temporal logic formulas that capture the specifications given above. Let x^i represent the position of the ith defending robot, y^j represent the height of the jth attacking robot, z^j represent the (horizontal) position of jth attacking robot and $\alpha(i)$ is the assignment for defending robot i. We assume that each attacking robot moves down a distance δ at each iteration. The Lyapunov function that is used to verify the stability of the protocol is given by

$$V = \left[\binom{n}{2} + 1 \right] \rho + \beta,$$

where

$$\rho = \sum_{i=1}^{n} r(i,i), \quad r(i,j) = \begin{cases} 1 & \text{if } y^{\alpha(j)} < |z^i - x^{\alpha(j)} - \delta \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\beta = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \gamma(i,j) \text{ where } \gamma(i,j) = \begin{cases} 1 & \text{if } x^{\alpha(i)} > x^{\alpha(j)} \\ 0 & \text{otherwise.} \end{cases}$$

The function ρ encodes the number of defending robots that are too far away to reach the robot they are assigned to defend against and the the function β is the total number of conflicts (assignments requiring robots to collide) in the current state. This Lyapunov function thus reaches a minimum when the set of assignments has a solution and it is possible to show that V always decreases whenever a switch happens. A full proof of stability is given in [**?**].

Extensions to this approach for observability and controllability have also been developed [**?**, **?**].

Additional results

A number of other recent results in networked control systems have become available in the past few years. In this section we briefly summarize some of these results coming out of the author's research group.

Estimation and Control with Information Loss. Through a sequence of results, we have explored the performance of estimation algorithms in the presence of networked channels in which information can be lost. This work builds on the results of Sinopoli *et al.* in which they develop bounds on the performance of a Kalman filter in the presence of packet loss [**?**].

In [**?**], we consider a discrete time state estimation problem over a packet-based network. In each discrete time step, the measurement is sent to a Kalman filter with some probability that it is received or dropped. The previous work of Sinopoli *et al.* on Kalman filtering with intermittent observation losses shows that there exists a certain threshold of the packet dropping rate below which the estimator is stable in the expected sense. In their analysis, they assume that packets are dropped independently between all time steps. We have developed a different point of view that extends this analysis in two ways. First, we do not required that the packets are dropped independently but just that the information gain π_g—defined to be the limit of the ratio of the number of received packets n during N time steps as N goes to infinity—exists. Second, we show that for any given π_g, as long as $\pi_g > 0$, the estimator is stable almost surely, i.e. for any given $\varepsilon > 0$ the error covariance matrix P_k is bounded by a finite matrix M, with probability $1 - \varepsilon$. Given an error tolerance M, π_g can in turn be found. We also give explicit formula for the relationship between M and ε.

Another approach that we have explored is the use of multi-description source coding [**?**, **?**, **?**]. In this work, we split the information that we wish to send across multiple packets, but encode the information so that if any collection of packets are

lost, the resulting information has a known level of distortion. We have considered two cases: when the packet loss over network links occurs in an i.i.d. fashion or in a bursty fashion. Compared with the traditional single description source coding, multi-description (MD) coding scheme can greatly improve the performance of Kalman filtering over a large set of packet loss scenarios in both cases.

We have also considered the problem of (optimal) control in the presence of packet loss [**?**, **?**]. We first prove a separation principle that allows us to solve this problem using a standard LQR state-feedback design, along with an optimal algorithm for propagating and using the information across the unreliable link. Then we present one such optimal algorithm, which consists of a Kalman filter at the sensor side of the link, and a switched linear filter at the controller side. Our design does not assume any statistical model of the packet drop events, and is thus optimal for any arbitrary packet drop pattern. Further, the solution is appealing from a practical point of view because it can be implemented as a small modification of an existing LQG control design.

We have extended this work in three ways to consider more realistic networks, where information may route through multiple nodes before being delivered to its destination.

In the first work, we consider the use of multi-hop protocols for improving the convergence rates of consensus algorithms [**?**]. We propose multi-hop relay protocols based on the current "nearest neighbor rules" consensus protocols. By employing multiple-hop paths in the network, more information is passed around and each agent enlarges its "available" neighborhood. We demonstrate that these relay protocols can increase the algebraic connectivity without physically adding or changing any communication links. Moreover, time delay sensitivity of relay protocols are discussed in detail. We point out that a trade off exists between convergence performance and time delay robustness. Simulation results are also provided to verify the efficiency of relay protocols.

We have also considered the problem of data transmissions over networks, where each node in the network is allowed to perform some amount of computation [**?**]. We consider the problem of determining the optimal processing at each node in the network and provide a strategy that yields the optimal performance at the cost of constant memory and processing at each node. We also provide conditions on the network for the estimate error covariance to be stable under this algorithm. This approach is applicable for networks of sensors that are performing spatio-temporal tasks such as cooperative situational awareness.

Finally, we have also explored the design of control strategies over lossy networks [**?**, **?**]. A network is assumed to exist between the sensor and the controller and between the latter and the actuator. Packets are dropped according to a Bernoulli independent process, with γ and μ being the probabilities of losing an observation packet and a control packet respectively, at time any instant t. A receding horizon control scheme is proposed for the Linear Quadratic Control (LQG) problem. At each instant, N future control inputs are sent in addition to the current one. Under this scheme the separation of estimation and control is shown and sta-

bility conditions, dependent on loss probabilities, are provided. Simulations show how the overall performance, in terms of lower cost, increases with the length of the horizon.

Distributed Sensor Fusion Using Dynamic Consensus.

A complementary way to explore distributed sensor fusion is to make use of previous results on consensus algorithms. A consensus algorithm seeks to get agreement between a set of distributed agents on a common quantity. In [**?**] we examined several dynamical aspects of average consensus in mobile networks. The results allow consensus on general time-varying signals, and allow tracking analysis using standard frequency-domain techniques. Further, the frequency-domain analysis naturally inspires a robust small-gain version of the algorithm, which tolerates arbitrary non-uniform time delays. Finally, we show how to exploit a dynamical conservation property in order to ensure consensus tracking despite splitting and merging of the underlying mobile network.

This work can be extended to to obtain least-squares fused estimates based on spatially distributed measurements [**?**]. This mechanism is very robust to changes in the underlying network topology and performance, making it an interesting candidate for sensor fusion on autonomous mobile networks. Examples have been explored to demonstrate the the dependence of the performance on the structure of the underlying network. A more systematic analysis of the performance as various network quantities such as connection density, topology, and bandwidth are varied has also been carried out [**?**]. Our main contribution is a frequency-domain characterization of the distributed estimator's steady-state performance; this is quantified in terms of a special matrix associated with the connection topology called the graph Laplacian, and also the rate of message exchange between immediate neighbors in the communication network.

These results have led to a more general formulation of the problem of distributed computing [**?**]. The main theoretical contribution of this work is a geometric formalism in which to cast distributed systems. This has numerous advantages and naturally parameterizes a wide class of distributed interaction mechanisms in a uniform way. We make use of this framework to present a model for distributed optimization, and we introduce the distributed gradient as a general design tool for synthesizing dynamics for distributed systems. The distributed optimization model is a useful abstraction in its own right and motivates a definition for a distributed extremum. As one might expect, the distributed gradient is zero at a distributed extremum, and the dynamics of a distributed gradient flow must converge to a distributed extremum. This forms the basis for a wide variety of designs, and we are in fact able to recover a widely studied distributed consensus algorithm as a special case.

Robustness to Process Uncertainty and Node Failure.

In most of the work on sensor coverage and distributed sensor fusion, it is assumed that the participating vehicles stay within communications range of each other and maintain overall connectivity of the network. We have analyzed the fea-

sibility aspects of motion planning for groups of agents connected by a range-constrained wireless network [**?**]. Specifically, we address the difficulties encountered when trajectories are required to preserve the connectedness of the network. The analysis utilizes a quantity called the connectivity robustness of the network, which can be calculated in a distributed fashion, and thus is applicable to distributed motion planning problems arising in control of vehicle networks. Further, these results show that network constraints posed as connectivity robustness constraints have minimal impact on reachability, provided that an appropriate topology control algorithm is implemented. This contrasts with more naive approaches to connectivity maintenance, which can significantly reduce the reachable set.

In [**?**, **?**], we consider a robust network control problem. We consider linear unstable and uncertain discrete time plants with a network between the sensor and controller and the controller and plant. We investigate two defining characteristics of network controlled systems and the impact of uncertainty on the process dynamics (modeled as parametric uncertainty). We compute the minimum data rate and minimum packet arrival rate to ensure stability of the closed loop system.

Another notion of robustness is robustness to failure of individual computational nodes [**?**]. This type of uncertainty has been considered in the area of distributed computation, but has not been considered in many of the networked estimation and control architectures that have been proposed. For a distributed algorithm to be practical, one should be able to guarantee that the task is still satisfactorily executed even when agents fail to communicate with others or to perform their designated actions correctly. We present a concept of robustness which is well-suited for general distributed algorithms for teams of dynamic agents. Our definition extends a similar notion introduced in the distributed computation literature for consensus problems. We illustrate the definition by considering a variety of algorithms and identify possible ways to make an algorithm robust.

Networked Control Systems Architecture. In addition to the specific research accomplishments listed above, work over the past 5 years has led to the development of a networked control systems architecture, illustrated in Figure 14.38, that is serving as the basis for multiple ongoing projects at Caltech. Building on the open source *Spread* group communications protocol, we have developed a modular software architecture that provides inter-computer communications between sets of linked processes. This approach allows the use of significant amounts of distributed computing for sensor processing and optimization-based planning, as well as providing a very flexible backbone for building autonomous systems and fault tolerant computing systems.

This architecture has been implemented and tested as part of Caltech's participation in the 2005 DARPA Grand Challenge [CFG+06, **?**].

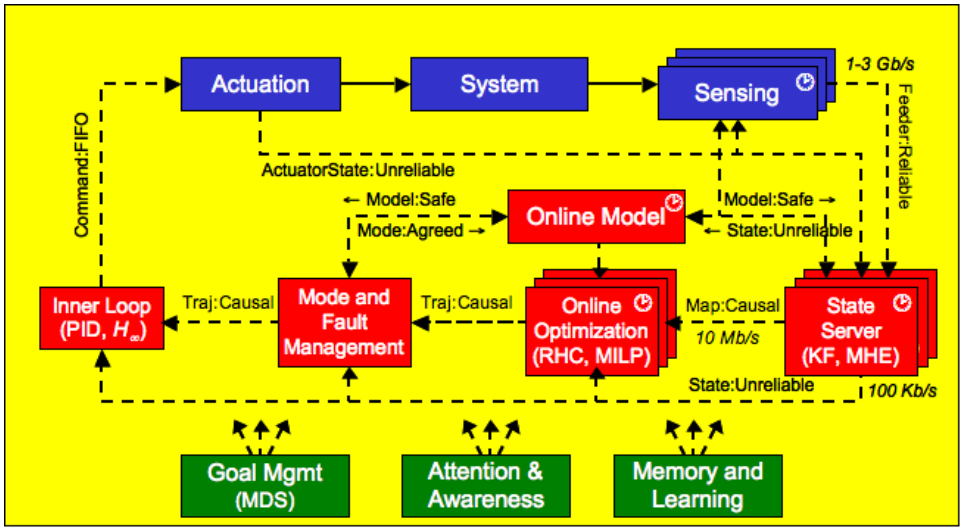


Figure 14.38: Networked Control Systems Architecture.

14.6 Adaptation Learning and Cognition

In this section we will discuss more sophisticased control laws with abilities to adapt, learn and reason. These funtions are key elements for autonomous control. Before proceeding we will consider review the maning of the words. Adapt is to adjust to a specified use or situation, learn is to acquire knowledge or skill by study, instruction or experience, reason is the intellectual process of seeking truth or knowledge by infering from either fact of logic and autonomy is the ability of beeing self-governing. When these words are used in the engineering context it is clear that the abilities are far from what we can accomplish as humans, but the development of autonomous cars and airvehicles are good indicatotrs of progress.

Adaptive Control

Adaptive control is a technique that can be used when there are significant variations in the process and its environment and where neither robust control nor gain scheduling is applicable. Model reference control and the self-tuning controller are two common approaches to adaptive control, see Figure 14.39. Model reference adaptive control (MRAS) is primarily used for command signal following and the self-tuning regulator is used both for intended for reduction of load disturbances. Notice in Figure 14.39 that there are two feedback loops: one conventional feedback loop involving the process P and the controller C and a slower loop to adjust the controller parameters θ.