Feedback Systems

An Introduction for Scientists and Engineers

SECOND EDITION

Karl Johan Åström
Richard M. Murray

Version v3.1.5 (2020-07-24)

This is the electronic edition of *Feedback Systems* and is available from http://fbsbook.org. Hardcover editions may be purchased from Princeton University Press, http://press.princeton.edu/titles/8701.html.

Chapter 15

Architecture and System Design

> *The architect's two most important tools are the eraser in the drafting room and the wrecking bar on the site.*
>
> Frank Lloyd Wright [Jac65].

In this chapter we place the relatively simple feedback loops that have been the focus of the previous chapters in the context of overall system design. We outline a typical design process and discuss the role of architecture and how it can be approached from top-down and bottom-up perspectives. Interaction and adaptation are then reviewed and the chapter ends with a brief overview of control design in some major industrial fields.

15.1 Introduction

So far we have dealt with relatively simple feedback systems. We will now give a glimpse of how they appear as components in real-world systems and how they are designed. All control systems have sensors, actuators, communications, computers, and operator interfaces, but they can have dramatically different sizes and shapes and very different user communities. It is surprising that such a variety of systems can be analyzed and designed using the same engineering framework.

The system to be controlled is often designed before control is considered. There are, however, significant advantages to designing a process and its control system jointly, so-called *co-design*. Care can be taken to ensure that the system is easy to control, for example by avoiding non-minimum phase dynamics. Time delays can be avoided by proper positioning of sensing and actuation. Use of feedback gives an extra degree of freedom to the designer; an extreme example is that a system can be made more maneuverable by making it unstable and then stabilizing it with a controller. The system itself and its physical and operational environment are key elements together with requirements, analysis, and testing.

Architecture, from the Greek word $\alpha\rho\chi\iota\tau\epsilon\kappa\tau\omega\nu$ ($\alpha\rho\chi\iota$ chief and $\tau\epsilon\kappa\tau\omega\nu$ builder, carpenter, mason), is the process of planning, designing, and constructing buildings

and other objects. It is also used to describe the structure of practically anything. In the context of control systems, the elements consist of the process, sensors, actuators, computers, communication devices, human machine interfaces, algorithms, and software. The control system interacts with the operational environment, it observes the process by sensors, and it interacts with the process through actuators and with the users through a range of interfaces. Architecture describes how the system components are connected and how they interact. There is a growing awareness that architecture is important in all engineering fields and today we have software, hardware, and systems architects.

15.2 System and Control Design

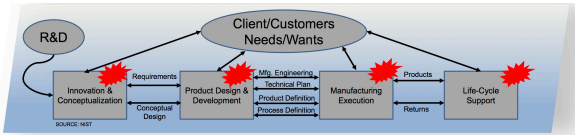
System design starts by developing an understanding of the system and its environment. It includes analysis of static and dynamic properties of the physical system and its sensors and actuators, bounds for safe operation, and characterization of the nature of the disturbances and the users of the system. There are a wide range of problems. Sometimes the process is given *a priori* and the task is to design a controller for a given process. In other cases the process and the controller are designed jointly. Co-design has many advantages because performance could be optimized. Sometimes it is an enabler, as was illustrated by the Wright Flyer, which was discussed in Section 1.5. We quote from the 43rd Wilbur Wright Memorial Lecture by Charles Stark Draper [Dra55]:

> The Wright Brothers rejected the principle that aircraft should be made inherently so stable that the human pilot would only have to steer the vehicle, playing no part in stabilization. Instead they deliberately made their airplane with negative stability and depended on the human pilot to operate the movable surface controls so that the flying system—pilot and machine—would be stable. This resulted in increased maneuverability and controllability.

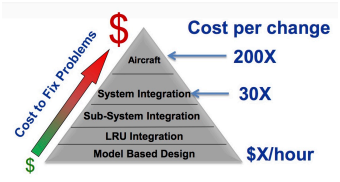
If the stabilization of an unstable airframe is done by an automatic control system, there are very strong requirements on the reliability of the control system. Design of the X-29, which was discussed in Example 14.2, is a similar case. A more recent example, which deals with difficulties caused by insufficient actuator authority, is presented in [EHBM08]. It was attempted to reduce the risk for rotating stall in a jet engine by feedback, but actuators with the required bandwidth were not available. Analysis showed that the problem could instead be alleviated by introducing small asymmetries in the turbine.

Figure 15.1 shows a typical design process and the costs of correcting faults at different stages in the process. Notice the significant value in correcting faults early. Design of complex systems is a major effort where many people and groups are involved. A variety to methods have been developed for efficient design. The so-called *V-model*, dating back to NASA's Apollo program, is a design pattern for both hardware and software [SC92]. It appears in many different forms: one version is part of the official project management methodology of software for the German government [Ano92].

One example of the design V is shown at the top of Figure 15.2. The left



(a) Design Flow



(b) Costs

Figure 15.1: Engineering design process. A typical design cycle is shown in (a) and (b) illustrates the costs of correcting faults or making design changes at different stages in the design process.

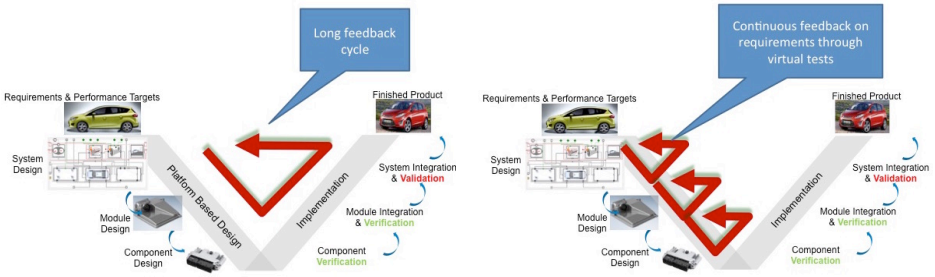
leg of the V illustrates the design process starting with requirements and ending with system, module and component design. The right leg of the V represents the implementation starting with the components and ending with the finished process and its validation. There are many substeps in the design, they include functional requirements, architecture generation and exploration, analysis and optimization. Notice that validation is made only on the finished product.

The cost of faults or changes increase dramatically if they are discovered late in the development process or even worse when systems are in operation, as illustrated in Figure 15.1. Model-based systems engineering can reduce the costs because models allow partial validation using models as virtual hardware at many steps in the development process as illustrated in the bottom part of Figure 15.2. When hardware and subsystems are built they can replace the corresponding models in hardware-in-the-loop simulation.

To perform verification efficiently it is necessary that requirements are expressed mathematically and checked automatically against requirements using models of the system and its environment and a variety of tools for analysis. *Regression analysis* can be used to avoid that changes in one part of a system do not create unexpected errors in other parts of the system. Efficient regression analysis requires robust system-level models and good scripting software that allows analyses to be performed automatically over many operating conditions with little to no human intervention. System-level models are also useful for *root cause analysis* by allowing errors to be reproduced, which is helpful to ensure that the real cause has been found.

There are strong interactions between the models and the analysis tools that are used; therefore, the models must satisfy the requirements of the algorithms for analysis and design. For example, when using Newton's method for solution of nonlinear equations and optimization, the models must be continuous and have continuous first (and sometimes second) derivatives. This property, which is called *smoothness*, is essential for algorithms to work well. Lack of smoothness can be due to: if-then-else statements, an actuator that saturates or by careless modeling of fluid systems with reversing flows. Having tools that check if a given system model has functions with continuous first and second derivatives is valuable.

An alternative to the use of the traditional design V is the *agile development* model, which has been driven by software developers for products with short time to market, where requirements change and close interaction with customers is required.



(a) Classical Design V (b) Design V for Model Based Design

Figure 15.2: The top figure (a) shows a traditional design V. The left side of the V represents the decomposition of requirements, and creation of system specifications. The right side represents the activities in implementation including validation (building the right thing) and verification (building it right). Notice that validation and verification are performed late in the design process when all hardware is available. The bottom figure (b) shows a model-based design process where virtual validation is be made at many stages in the design process, shortening the feedback for validation.

The method is characterized by the *Agile Manifesto* [BBvB+01], which values individuals and interactions over processes and tool, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan. When choosing a design methodology it is also important to keep in mind that products involving hardware are more difficult to change than software.

Control system design is a subpart of system design that includes many activities, starting with requirements and system modeling and ending with implementation, testing, commissioning, operation, and upgrading. In between are the important steps of detailed modeling, architecture selection, analysis, design, and simulation. The V-model used in an iterative fashion is well suited to control design, particular if it is supported by a tool chain that admits a combination of modeling, control design, and simulation. Testing is done iteratively at every step of the design using models of different granularity as virtual systems. Hardware in the loop simulations are also used when they are available. A scripting language is helpful to execute the design.

Control system specifications are typically given by large and small signal behavior of the closed loop system. Large signal behavior is characterized by limits in actuation power and its rate, small signal behavior is typically caused by measurement noise, friction, and resolution of A/D and D/A converters. Requirements for control systems include the ability to deal with disturbances, robustness to process variations and uncertainty, and the ability to follow reference signals.

Many control system specifications can be captured by linear models and they can be expressed in terms of properties of the Gangs of Four and Six, discussed in Section 12.1. Referring to the block diagram in Figure 15.3, load disturbance attenuation can be characterized by the transfer function G_{yv} from load disturbance v to process output y. Measurement noise w generates undesired control actions, the effect of which can be captured by transfer function G_{uw} from measurement noise w

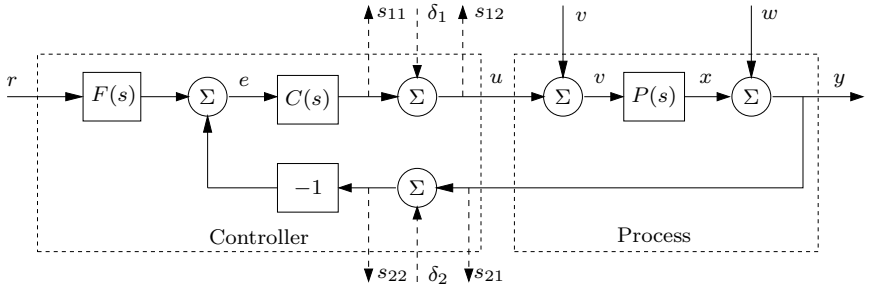


Figure 15.3: Specifications can be tested by injecting signals at test points δ_k and measuring responses at s_{ij}. Compare with Figure 12.1

.

to control action u. Robustness to parameter variations and process uncertainty can be captured by the sensitivity functions S and T. Reference signal response can be shaped independently of response to disturbances and robustness for systems with two degrees of freedom. It is characterized by the transfer functions TF and CSF. Systems with error feedback are more restricted because the response to reference signals is characterized by the complementary transfer function T and a compromise must be made between command signal response and the other requirements.

Since many specifications are expressed in terms of properties of the transfer functions in the Gang of Six, it is important to measure these transfer functions on simulated models and on real hardware. To do this the system must be provided with test points for injecting and measure signals, as indicated by the dashed arrows in Figure 15.3. The transfer function G_{yv}, which characterizes response to load disturbances, can be found by injecting a signal at δ_1 and measuring the output s_{21}. Chirp signals are convenient for measuring frequency responses.

Models of the process and its environment can be obtained from physics, from experiments, or from a combination. Experiments are typically done by changing the control signal and measuring the response. The signals can range from simple step tests to signals that are designed to give optimal information with limited process perturbations. System identification methods and software provide useful tools. The models used at different stages typically have different fidelity, cruder in the beginning and more accurate as the design progresses.

A few standard design methods have been discussed in Chapters 7, 8, 11, and 12, but there are many more methods in the literature [Fri04, GGS01]. Many design methods are based on linear models, however, when environmental conditions change significantly it it necessary to use gain scheduling, nonlinear control, or adaptation. Receding horizon control (also called model predictive control) is another common approach, especially useful when there are constraints on the inputs or states.

Today most control systems are implemented using computer control. Implementation then involves selection of hardware for signal conversion, communication, and computing. A block diagram of a system with computer control is shown in Figure 15.4. The overall system consists of sensors, actuators, analog-to-digital and digital-to-analog converters, and computing elements. The filter before the A/D converter is necessary to ensure that high-frequency disturbances do not appear as

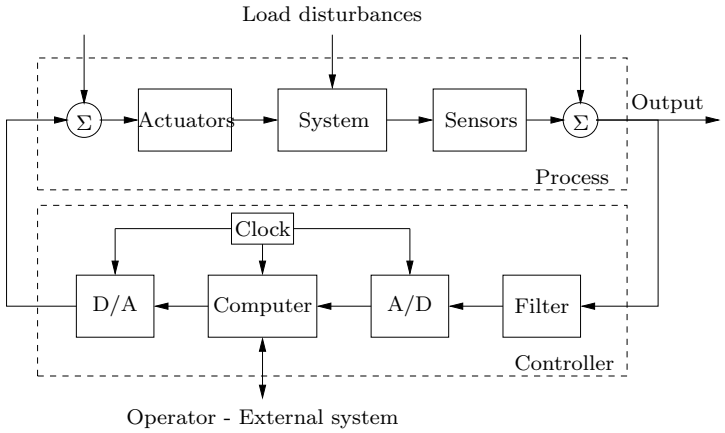


Figure 15.4: Schematic diagram of a control system with sensors, actuators, communications, computer, and interfaces.

low-frequency disturbances after sampling because of aliasing. The operations of the system are synchronized by a clock.

Real-time operating systems that coordinate sensing, actuation, and computing have to be selected, and algorithms that implement the control laws must be generated. The sampling period and the anti-alias filter must be chosen carefully. Since a computer can only do basic arithmetic, the control algorithms have to be represented as difference equations. They can be obtained by approximating differential equations, as was illustrated in Section 8.5, but there are also design methods that automatically give controllers in the form of difference equations. Code can be generated automatically. It must also be ensured that computational delays and synchronization of algorithms do not create problems.

When the design is implemented and tested the system must be commissioned. This step may involve adjustment of controller parameters, and automatic tuning (discussed in Section 11.3) can be very beneficial at this stage. During operation it is important to monitor the behavior of the system to ensure that specifications are still satisfied. It may be necessary to upgrade the system when it has been operating. Specifications may also be modified due to operational experiences.

It is highly desirable to have a suite of test programs that can be used throughout the design and operation stages to ensure that requirements are satisfied.

15.3 Top-Down Architectures

When system design is approached systematically using the design V as described in Section 15.2, it is natural to use a top-down procedure for control design starting with the desired properties of the system and decomposing the overall control design problem into an interlinked set of control problems at different layers of abstraction. At each layer of abstraction, we make assumptions about the interactions with the higher and lower layers in order to simplify the control design problem. In this section we give a brief introduction to some of the organizing principles of top-down architectures for control and its connections to some of the techniques described

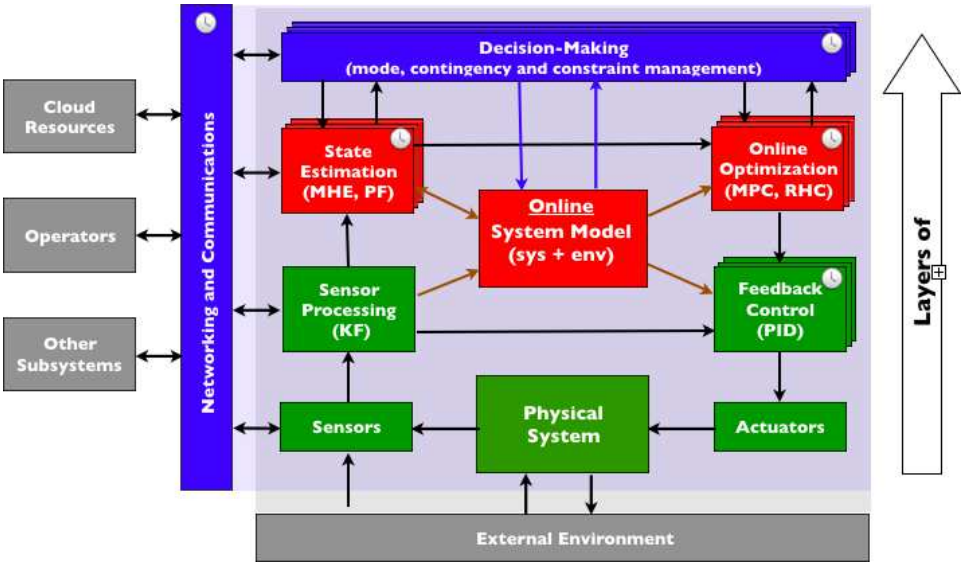


Figure 15.5: Layered decomposition of a control system.

in the text, along with references to the literature for those interested in further details.

There are many other aspects of control architectures that are part of control systems design. These include such topics as cooperative control [Par93, Mur07], diagnostics and health monitoring [DM02b, GQH16], fault recovery and system reconfiguration [HC03, BKLS16], and game-theoretic approaches to control [MS15]. We focus here on a small subset of the problem, with an emphasis on multi-layer approaches to control.

Layered Architectures for Control

For complex control systems, it is often useful to break down the control problem into a hierarchy of control problems, each solved at a different layer of abstraction, as illustrated in Figure 15.5. Different types of specifications are used at each layer to determine the control functionality that will be implemented.

The specific abstraction layers in a control architecture depend on the problem domain. In Figure 15.5, we have used a decomposition that is common in many motion control problems, including robotics, self-driving cars, and flight control. Similar decompositons also appear in application domains such as manufacturing, process control, and computing systems. At the top layer of abstraction, we care about discrete modes of behavior, which could correspond to different phases of operation (takeoff, cruise, landing) or different environment assumptions (highway driving, city streets, parking lot). The next layer of abstraction reasons about trajectories of the system, often using an optimization-based approach. At this layer, we often take into account the constraints on the system operating state and inputs, as well as system-level descriptions of performance. Finally, at the lowest layer of the abstraction hierarchy we have the feedback control design that has been

the main topic in this text thus far, where we may use a linearized model based on the current operating point (along a trajectory).

Note that at each abstraction level we must consider not only the control design but also the way that sensory information is processed and represented. At the lowest levels of abstraction we may use individual sensor signals, perhaps with some filtering. As we move up the abstraction hierarchy, it will be necessary to fuse multiple measurements to obtain a more integrated view of the system state, with the highest level of abstraction often requiring sophisticated methods of reasoning about the state of the environment and the predicted interactions with other entities in that environment.

The architecture in Figure 15.5 is suitable for systems of moderate complexity where the users interact with the system by changing modes and references. More layers are used for complex systems with complicated user interaction. Batch control is a typical example, where a complex manufacturing system is used to control different batches of chemicals and where the material flow through the factory changes. In this case there are two additional layers—procedural control and coordination control—on top of those shown in Figure 15.5. Procedural control executes the sequence of actions necessary to carry out a process oriented task, such as charging a reactor with a specific amount and type of raw material and reporting the result. Coordination control directs, initiates, or modifies the execution of procedural control and the utilization of equipment entities.

In addition to these additional layers, a production facility typically operates in different modes: normal, maintenance, and manual. The maintenance mode has its own control algorithms and safety procedures to ensure that the system does not react in an unsafe manner during maintenance. The manual mode is typically used for equipment maintenance and debugging. Different parts of a manufacturing system can be in different modes. An example of an architecture for distributed control system (DCS), typical for complex manufacturing systems, is shown in Figure 15.6.

An important feature of many control systems architectures is the modularity of the control software, enabling parallel development of components and the ability to upgrade components without having to redesign the entire system. Figure 15.7 shows two types of features that are common in architectures: a “bowtie” pattern and an “hourglass” pattern.

The “bowtie” pattern refers to the use of a common interface within a layer of abstraction that enables many different subsystems to connect together across the interface. As an example, in the context of the sensing system for an autonomous vehicle, a common representation of map data allows many types of sensors to feed information into the map and different layers of controller to extract (fused) information from the map. Through this common interface, new sensors or control functionality can be added on either side of the interface without having to redesign the rest of the system.

The “hourglass” pattern represents a hierarchy of control functions that uses a common interface to enable changes above and below that interface to be changed independently of each other. The Open Systems Interconnection model (OSI model) uses seven standardized layers—applications at the top and the physical layer at the bottom—and has been a key to obtain interoperability in communication systems.

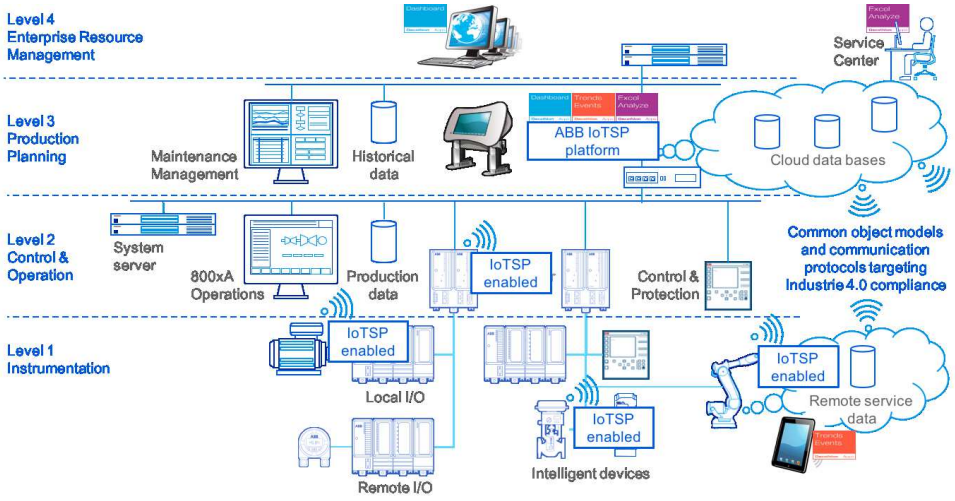


Figure 15.6: Functional architecture of process control system, implemented as a distributed control system (DCS). Figure courtesy of ABB, Inc.

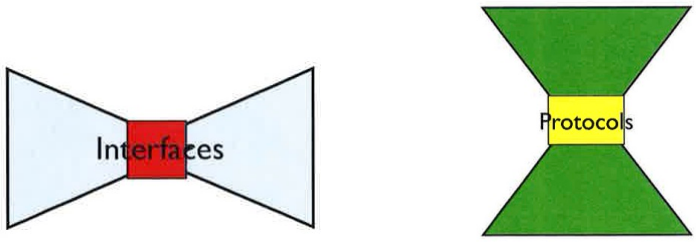
Another example of an hourglass pattern is used for the planning system in an autonomous vehicle. Trajectories are used to connect higher and lower levels of the navigation system. Any high-level function that eventually leads to a trajectory is completely compatible with the lower level controllers that will track that trajectory, allowing the higher levels of decision making to be changed without having to modify the trajectory tracking code. Similarly, the lower level controllers can be changed without having to redesign the high level decision-making, as long as they properly perform the function of tracking a given trajectory.

The bowtie and hourglass patterns shown here can be appear multiple times in a given architecture, so that we obtain appropriate "stacks" of sensing and control functionality. The following example, given already in the introduction, illustrates some of these concepts.

Example 15.1 Autonomous driving

As an example of a top-down architecture for control, we consider a control system for an autonomous vehicle, shown in Figure 15.8. This control system is designed for driving in urban environments. The feedback system fuses data from road and traffic sensors (cameras, laser range finders, and radar) to create a multi-layer "map" of the environment around the vehicle. This map is used to make decisions about actions that the vehicle should take (drive, stop, change lanes) and plan a specific path for the vehicle to follow. An optimization-based planner is used to compute the trajectory for the vehicle to follow, which is passed to a trajectory tracking module. A supervisory control module performs higher-level tasks such as mission planning and contingency management (if a sensor or actuator fails).

We see that this architecture has the basic features shown in Figure 15.5. The control layers are shown in the navigation block, with the mission planner and traffic planner representing two levels of discrete decision-making logic, the path



(a) Bowtie pattern (b) Hourglass pattern

Figure 15.7: Architectural patterns that support modularity. The bowtie pattern is used to connect subsystems at the same level. Proper interface design supports independence of the subsystems. The hourglass pattern is used to connect subsystem at different levels using protocols. Proper protocols supports independence of the subsystems at different levels.

planner representing a trajectory optimization function and then the lower layers of control. Similarly, there are multiple layers of sensing, with low level information, such as vehicle speed and position in the lane, being sent to the trajectory tracking controller, while higher level information about other vehicles on the road and their predicted motions is sent to the trajectory, traffic, and mission planners. ∇

Online Optimization

The use of real-time trajectory generation techniques enables a much more sophisticated approach to the design of control systems, especially those in which constraints must be taken into account. The fact that such trajectories can be computed quickly enables us to use a *receding horizon control* technique: a (optimal) feasible trajectory is computed from the current state to the desired state 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 $t+T+\delta$, as shown in Figure 15.9. Development and application of receding horizon control (also called model predictive control, or MPC) originated in process control industries where

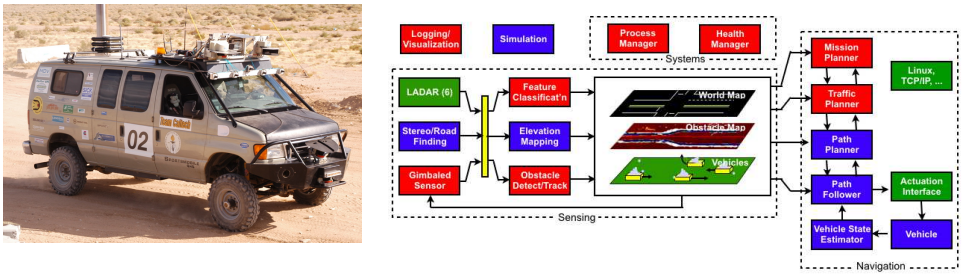


Figure 15.8: DARPA Grand Challenge. "Alice," Team Caltech's entry in the 2005 and 2007 competitions and its networked control architecture [CFG+06].

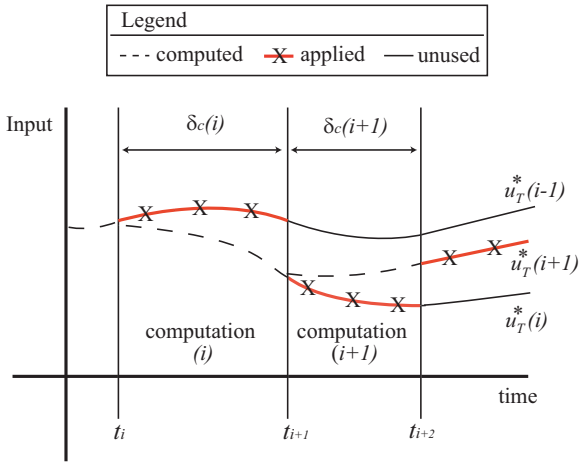


Figure 15.9: The idea of receding horizon control.

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 [QB97] and a survey of the state of stability theory of MPC is given in [MRRS00].

Figure 15.10 shows a typical setup for a receding horizon control problem. In this formulation, the trajectory generation block solves the following constrained trajectory generation problem at each time step:

$$\min_{(x,u)} = \int_t^{t+T} L(x,u)\,d\tau + V(x(t+T)) \quad \text{subject to} \quad \begin{cases} x(t) = \text{current state} \\ \dot{x} = f(x) + g(x)u, \\ lb \le c(x,u) \le ub. \end{cases} \tag{15.1}$$

One of the challenges of properly implementing receding horizon control is that instabilities can result if the problem is not specified correctly. In particular, because we optimize the system dynamics over a finite horizon T, it can happen that choosing the optimal short term behavior can lead us *away* from the long term solution (see [MR96] for an example). To address this problem, the terminal cost $V(x(T))$ must have certain properties to ensure stability (see [MRRS00] for details).

One of the chief challenges in implementing receding horizon control is the need for fast computation of feasible trajectories. One class of systems for which this is easier are *differentially flat* systems, defined briefly in Section 8.5.

Discrete-decision making and supervisory control

Design of control systems involves the analysis and synthesis of feedback controllers at multiple levels of abstraction, from fast feedback loops around actuators and subsystems, to higher level decision-making logic in supervisory controllers and autonomous systems. One of the major challenges in design of complex networked control systems—such as those arising in aerospace, computing, robotics, and critical infrastructure—is insuring that the combination of dynamical behavior and

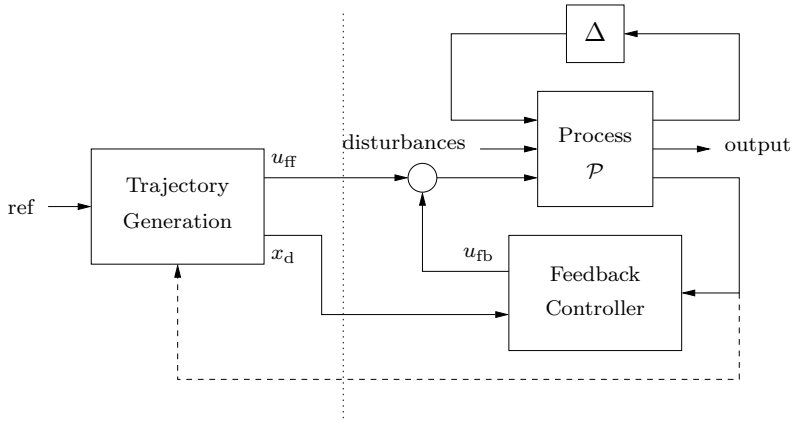


Figure 15.10: Two degree-of-freedom controller design for a process P with uncertainty Δ. The controller consists of a trajectory generator and feedback controller. The trajectory generation subsystem computes a feedforward command $u_{\rm d}$ along with the desired state $x_{\rm d}$. The state feedback controller uses the measured (or estimated) state and desired state to compute a corrective input $u_{\rm fb}$. Uncertainty is represented by the block Δ, representing unmodeled dynamics, as well as disturbances and noise.

logical decision-making satisfies safety and performance specifications. In many of these areas, verification and validation are now dominant drivers of schedule and cost, and the tools available for design of such systems are falling behind the needs of systems and control engineers, particularly in the area of systematic design of the mixed continuous and discrete control laws for networked systems.

We consider systems consisting of subsystems/agents whose dynamics are described by ordinary differential equations of the form

$$\dot{x}^i = f^i(x^i, \alpha^i, u^i), \qquad y^i = h(x^i, \alpha^i),$$

where $x^i \in \mathbb{R}^{n_i}$ is the continuous state of the ith subsystem, $\alpha \in \mathcal{A}$ is the discrete state, $u^i \in \mathbb{R}^{m_i}$ is a control input and $y^i \in \mathbb{R}^{p_i}$ is the measured output of subsystem. The discrete state evolves according to a set of "guarded commands," in which the discrete state α is updated to a new value only if a guard $g^p(x, \alpha)$ is true:

$$g_j^p(x, \alpha) \implies \alpha' = r_j^p(x, \alpha).$$

This specification allows the discrete state to evolve in an asynchronous way (e.g. for modeling failures) or to depend on the system state (e.g. to model nonlinearities or changes in connectivity). A model of this type is called a *discrete transition system*. The overall system, consisting of both continuous dynamics and discrete (state) dynamics, is called a *hybrid system*.

A controller for the system is a combination of a continuous control law and a discrete control *protocol*:

$$u = k(x, \alpha), \qquad g_j^c(x, \alpha) \implies \alpha' = r_j^c(x, \alpha). \tag{15.2}$$

The control protocol (sometimes called a supervisory controller) is in the form of discrete transition system, which controls some subset of the discrete states. The

discrete state is assumed to be updated by a periodically controlled process that examines the guards and updates appropriate rules. This model for the control allows for the possibility of distributed computation in which different systems (or subsystems) execute on loosely regulated clocks.

The system specification for a hybrid system is often composed of both a continuous performance specification and a discrete performance specification. For the continuous portion of the specification, a typical form is to use a cost function J that is written as a finite horizon cost

$$J = \int_0^T L(x, \alpha, u)dt + V(x(T)). \tag{15.3}$$

This function, a variant of which we have just seen in the context of receding horizon control, uses an integral cost over a fixed horizon T along with a terminal cost $V(x(T))$, where V is an appropriate positive function.

For the discrete performance specification, we make use of *temporal logic* formulas. For a discrete transition system, a temporal logic formula describes conditions on the sequence of events. One mathematical language that is widely used is *linear temporal logic* (LTL), which makes use of two temporal operators: always ($\Box$) and eventually ($\Diamond$). Given a logical formula $\varphi(\alpha)$ that evaluates to true or false for a given (discrete) state α, we can define a temporal logic formula $\Box\varphi$, which is interpreted as meaning that $\varphi(\alpha)$ should be true at all times in the future. Similarly, the formula $\Diamond\varphi$ represents the temporal logic statement that the logical formula φ is true at some future state. By combining these temporal operators with standard logical operators, we can obtain more complex formulas. For example, the formula

$$\Box(\varphi \implies \Diamond\psi)$$

can be interpreted as saying that at all times, if the formula φ is satisfied (evaluates to true), then eventually (at some time in the future) the formula ψ is true. This formula is a typical form for specifying that a system should respond to a certain event (captured by φ becoming true) by taking a certain action (captured by ψ becoming true). The always ($\Box$) operator at the outer level of the formula describes the specification that this condition should be satisfied at all times, which means that the system should respond over and over again if the event condition occurs repeatedly.

A typical LTL formula for a supervisory control system has the form

$$\varphi_{\text{init}} \wedge \Box\varphi_{\text{env}} \implies \Box\varphi_{\text{safe}} \wedge \Diamond\varphi_{\text{goal}}, \tag{15.4}$$

where φ_{init} represents a proposition describing the initial state of the system, φ_{env} describes the possible actions of the environment, φ_{safe} is a safety requirement, and φ_{goal} is a progress requirement. The environmental description φ_{env}, safety requirement φ_{safe}, and progress requirement φ_{goal} are typically described using LTL or other temporal logics (including computational tree logic, CTL, or one of its variants, TCTL, pCTL, etc.). These languages allow various specifications on sequences of actions, such as requiring that a certain condition hold until another condition is satisfied, or requiring that a certain condition occurs on a regular basis.

The control design problem for a supervisory system consists of finding a control law of the form in equation (15.2) that satisfies the system specification (15.4) while

minimizing the cost function (15.3). Traditional approaches to this problem involve synthesis of the continuous control law using optimization-based approaches, as described earlier in this section, and manual design of the discrete control protocols. The system is then checked against the specification by running (many) repeated simulations and checking that in each case the system specification is satisfied.

Linking Continuous and Discrete Controllers

In Section 15.3 we saw techniques for finding (optimal) controllers for continuous systems and in the previous section we have talked about techniques for design of discrete controllers. Going back to our initial top-down architecture in Figure 15.5, we now consider the problem of how to link these two different layers of the control system.

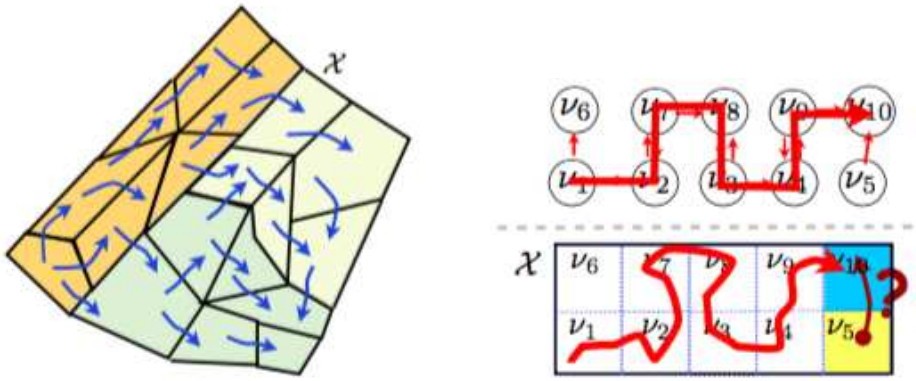
One approach to linking the two layers is to develop control techniques that can handle both the continuous and discrete dynamics in a common framework. One such approach is to make use of the notion of *mixed logical dynamical* (MLD) systems, introduced by Bemporad and Morari [BM99]. In this formulation, we extend the trajectory generation and receding horizon frameworks to handle discrete variables in the underlying constrained optimization problem. This requires the use of so-called *mixed integer solvers*, which allow optimization for problems with both continuous and discrete (integer valued) variables. As computers become increasingly powerful, the size and complexity of problems that we can handle with these tools have increased and these techniques are more and more commonly used.

An alternative is to solve the supervisory control problem and the trajectory generation problem separately, with an appropriate simplified representation of the dynamics of the layers above and below the one that for which we are doing the design.

For example, a common approach in doing trajectory generation is to assume that the discrete state (which might represent an operational mode or an environmental context) transitions from one state to another and then remains fixed for a sufficiently long period of time. Under this assumption, we can focus our attention on the problem of trajectory generation with an initial condition that may represent the state of the system just prior to transition to the new mode and the assume that the mode stays constant for the duration of our planning horizon. We are then required to make sure that our supervisory controller imposes this restriction on the time between mode switches as part of its specification.

This type of linkage between two layers in our abstraction is called a *vertical contract* and can often be written in *assume-guarantee* form. The supervisory layer will assume that the trajectory generation layer maintains the system specification in a given mode given enough "settling" time, and then guarantees that it does not switch the system mode more quickly than the settling time. Similarly, the trajectory generation layer will assume that the switching is sufficiently long, but must then guarantee that it satisfies the system specification within the prescribed settling time.

Similar to the simplified model of the supervisory controller used at the trajectory generation layer (the mode is essentially constant), the supervisory controller must have an appropriate representation of the trajectory generation dynamics. Since the supervisory controller design is done using discrete transition system



(a) Discretization (b) Simulation relation

Figure 15.11: Representation of continuous dynamics as a discrete transition system

. .

models, our representation of the dynamics of the lower layers of abstraction must be in terms of a discrete transition system model. One simple approach to such a representation is to break the continuous state space into a collection of discrete regions, such that each continuous state is contained in one discrete region. The dynamics of the trajectory generation layer can then be represented as a set of transitions between adjacent regions in the discretized state space, as illustrated in Figure 15.11a. In setting up the discrete representation, it is important that any trajectory in the discretized state space correspond to a feasible trajectory in the original continuous state space, and *vice versa*. This is required so that when the supervisory controller layer commands the system to move from one region to another then the trajectory generation layer is able to do so, and conversely when the continuous system executes a command and moves from one (continuous) state to another, it corresponds to a valid transition between the regions.

Model Checking and Program Synthesis

Given a system model, a controller design and performance specifications, we must verify that the controller satisfies the specifications. This is typically done by running many simulations. A problem in using simulation to try to check whether the design satisfies the specification is that it can be prohibitively time-consuming to simulate every possible sequence of events. For purely discrete state systems, in which the dynamics are completely specified by a set of discrete variables α, it turns out that there are more efficient techniques for verifying that a system is specified by making use of the structure of the logical formula and the discrete dynamics of the system. These verification techniques are referred to as *model checking*.

A block diagram that describes how model checking is performed is shown in Figure 15.12. The main idea of model checking is to make use of a model of the (discrete) system dynamics and the temporal logic specification to create a discrete transition system, known as the *product transition system*, where finding a

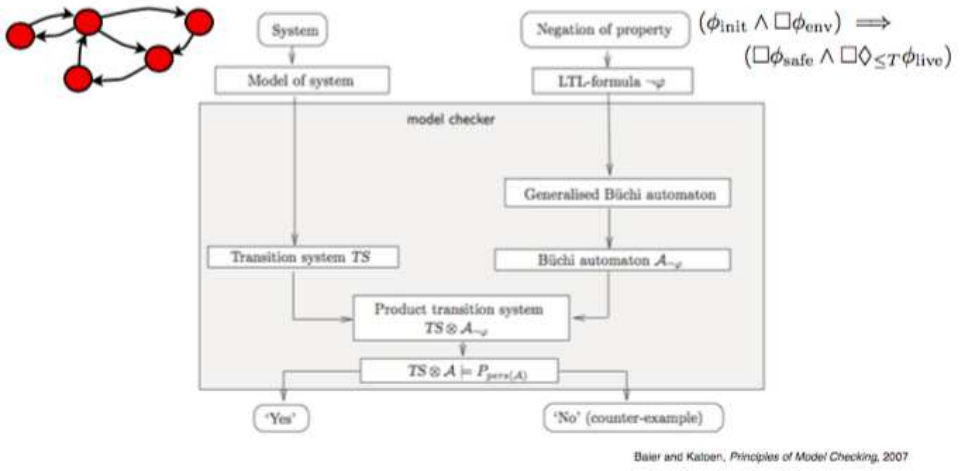


Figure 15.12: Model checking.

path through this transition system represents a possible system execution (set of allowable state transitions) that *violates* the system specification. If no such path can be found, then the system specification is satisfied. But if a path is found, it represents a counter-example that can be used to update the controller design.

Model checking tools, such as nuSMV [CCG+02], PRISM [KNP11], SPIN [Hol03], and TLC [Lam03] are capable of handling relatively large discrete state systems with quite general classes of specifications (beyond simple LTL formulas). They are now widely used in industry and are increasingly being applied in mission and safety critical applications, such as planetary exploration and aviation.

Despite their increasing power and applicability, a limitation in the use of model checking is that it does not necessarily provide any insight into how to redesign the system if the specification is not met. Rather, it provides a counterexample indicating what can go wrong, and leaves it to the designer to understand why the controller is not correct and then redesign the system. The updated design can be re-verified using the model checker, and this process is iterated until a correct design is obtained.

An alternative to iterative manual design is to make use of results in *program synthesis* or *correct-by-construction* synthesis [MW80]. The basic idea in synthesis is to create an algorithm that takes a model of the discrete system dynamics along with a temporal logic specification and then *synthesize* a control protocol such that the resulting closed loop system satisfies the specification. This approach is conceptually similar to the LQR design technique outlined in Section 7.5: given a system specification (cost function to minimize) and system model (A, B, C matrices), create a controller ($u = -Kx$) that by construction stabilizes the system and optimizes the performance specification.

Model checking and program synthesis make use of a common set of information (system model and system specification), but solve different problems, as illustrated in Figure 15.13. The advantage of using synthesis is that it provides a control protocol that is guaranteed to satisfy the specification. That is, no matter what

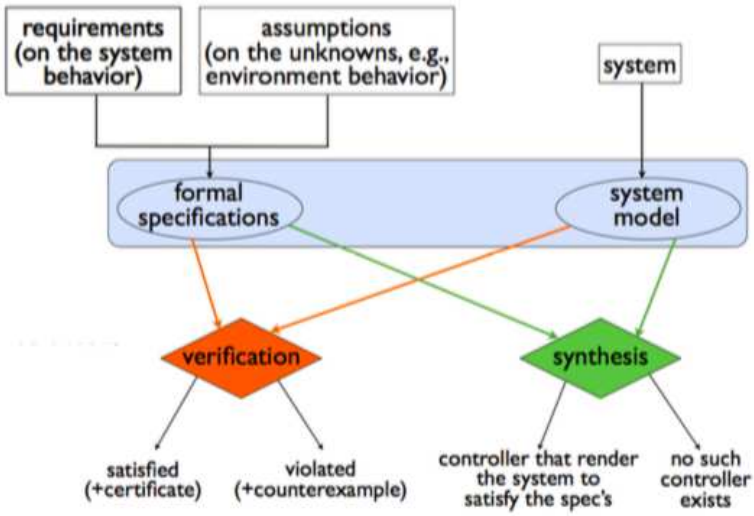


Figure 15.13: Verification versus synthesis.

sequence of environmental events occur, the system will always respond in a manner that satisfies the specification.

Of course, correct-by-construction synthesis is not a panacea. It can only be used for certain types of specifications and it too can be overwhelmed by complexity. But it is an increasingly important tool in the systematic synthesis of control protocols, especially safety-critical systems. A practical way to obtain a safe system is to design the system in layers with algorithms and guards. The algorithms perform their ordinary tasks and the guards determine if the system is functioning properly. The innermost layer, which for flight control is called *flying home mode*,, consists of a simple robust controller that provides the basic properties. If the loop is simple enough it can be designed to be correct-by-construction. The algorithms at the higher layers deliver better performance, and they have guards that move the system to a lower, safer layer if the system does not perform properly. The safety of the guards must of course also be guaranteed. Systems of this type have been designed by Sha [Sha01], and examples of guards for adaptive control are found in [THÅ00].

15.4 Bottom-Up Architectures

An alternative to the approaches described in the previous section is to design controllers by interconnecting low-level control systems to create more sophisticated capabilities. This approach is referred to as "bottom up" design. The idea of building complex systems from standard parts has emerged in many branches of engineering. In design of mechanical devices it was very efficient to standardize nuts and bolts. In electronics, standards emerged for components, circuits, boards, and patterns for VLSI design.

To use bottom-up design of control systems, we must find the appropriate com-

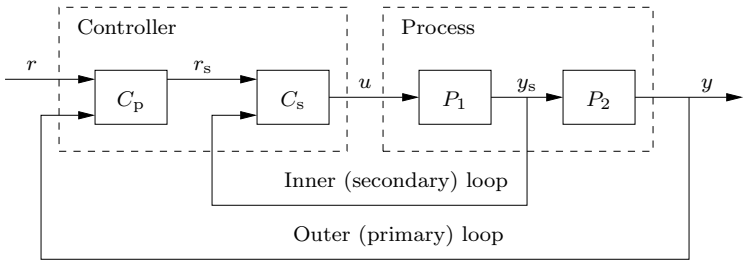


Figure 15.14: Block diagram of a system with cascade control. The dashed controller block has two controllers C_p and C_s in series. Both controllers have two degrees of freedom with two inputs: the reference (top) and the measured variable (bottom). The process has two blocks P_1 and P_2 in cascade. There are two loops: an *inner or secondary loop* and an *outer or primary loop*.

ponents or building blocks and the rules for interconnecting them. The building blocks are controllers (often PID), nonlinear functions, filters, logic, and finite state machines. They can either be separate pieces of hardware or function blocks implemented in software. The systems are built loop by loop by using structures or control principles such as feedback, feedforward, and gain scheduling, which have been discussed extensively in previous chapters. In this section we will introduce other architectural structures: cascade control, mid-range control, selector control, internal model control (IMC), Smith predictors, extremum seeking, and complementary filtering.

Bottom-up architectures can deal with systems having many inputs, many outputs, and constraints. An advantage is that the system can be designed, commissioned, and tuned loop by loop. The disadvantages are that it is not easy to judge the consequences of adding loops and that there may be difficulties when loops are interacting. Bottom-up architectures are easy to use for simple systems, but for complicated systems it may be better to use a top-down approach.

Cascade Control – Several Sensors

Cascade control is a scheme for using one actuator and two or more sensors. Versions of it were previously encountered in Figure 1.13 and in Example 12.9, where it was called inner-outer loop design. A block diagram of a closed loop system with cascade control is shown in Figure 15.14. The dashed process block has one control variable u and two measured signals: the primary output y and the secondary output y_s. The process is modeled by the blocks with transfer functions P_1 and P_2, which capture how the measured signals are related to the control variable. The dashed controller block in the figure has two controllers C_p and C_s, which are connected in cascade. It has three inputs: the reference r and the measured signals y and y_s. The *primary controller* C_p attempts to keep the output y close to the reference r by manipulating the reference input r_s of the *secondary controller* C_s. The secondary controller C_s attempts to keep the secondary output y_s close to its reference r_s manipulating the control variable u.

The controllers C_p and C_s can be of any type, but PID controllers are most

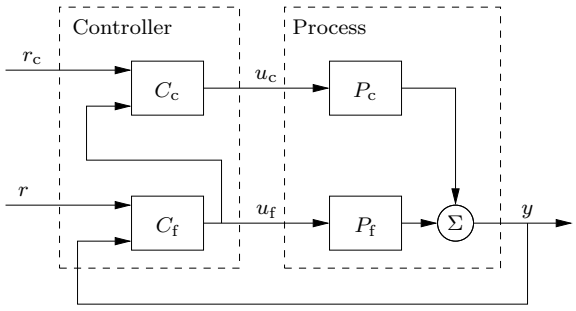


Figure 15.15: Block diagram of a closed loop system with mid-range control. The process has one output y and two inputs u_f and u_c. The input u_f provides fine control with limited range, the input u_c provides coarse control with wide range. The controller C_f attempts to keep the output y close to its reference r. The controller C_c controls u_c so that the variable u_f is in the middle of this range.

common. Design is done loop by loop starting with the inner loop. If integral action is used, it is necessary to have a scheme to avoid integral windup. Anti-windup for the secondary controller can be done in the conventional way, since the controller drives the actuator directly. To provide anti-windup for the primary controller it must be told when the secondary controller goes into anti-windup mode.

Cascade control is a convenient way to use extra sensors to improve control performance. It can be used with more than two sensors: the ultimate case is state feedback when all states are measured. Cascade control is particularly useful when there is significant time delay or dynamics between the input u and the primary output y, but significantly less dynamics between u and the secondary output y_s. A tight feedback in the inner loop reduces effects of disturbances and uncertainties in the block P_1, and simplifies the design of the outer loop.

Mid-Range Control – Many Actuators

Midranging is a control scheme that can be used when several control signals influence the same measured output. A typical example is a CD player with a fast actuator having a small actuation range and a slower actuator with a wide actuation range. The block diagram in Figure 15.15 shows an example. The process has two control signals u_f for fine control and u_c for coarse control. They influence the output y through dynamics described by the transfer functions P_f and P_c.

The controller C_f drives u_f and is the primary controller that attempts to keep the output y close to its reference r. The second controller C_c drives the subsystem P_c, which has wide actuation range. The measured signal for C_c is the output u_f of the controller C_f, and the reference is the middle range of u_f. The controller C_c manipulates the control variable u_c so that u_f is in the middle of its operating range and can handle moderately large disturbances. Both controller will act for large disturbances.

The controllers are tuned loop by loop, starting with the C_f. Anti-windup is handled in the standard manner if controllers have integral action.

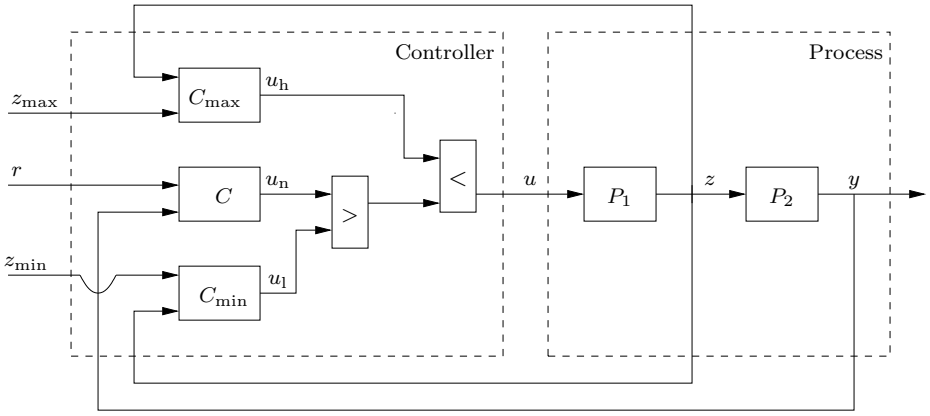


Figure 15.16: Block diagram of a system with selector control. The primary controller C is designed to keep y close to its reference r. The controllers $C_{\max}$ and $C_{\min}$ are controllers that ensures that the intermediary variable z is in its permissible range $z_{\min} < z < z_{\max}$. The block marked $<$ is a minimum selector, whose output equals the smallest input. The block marked $>$ is a maximum selector, whose output equals the largest input.

Selector Control – Equipment Protection

Selector control is used to control a primary variable while keeping auxiliary variables within given constraints for safety or for equipment protection. A selector is a function with many inputs and one output. The output of a *maximum* selector is the largest of the inputs and the output of a *minimum* is the smallest of the inputs.

Selector control is illustrated in Figure 15.16. The primary controlled variable is the process output y. The primary controller C, with output u_{n}, attempts to keep y close to its reference r. To guarantee safe operation the auxiliary variable z should be kept in the range $z_{\min} < z < z_{\max}$. This is accomplished by the secondary controllers $C_{\max}$ and $C_{\min}$, which have reference signals $z_{\max}$ and $z_{\min}$.

The control signal u is generated by sending u_{n}, u_{h} and u_{l} through maximum and minimum selectors as shown in the figure. Under normal conditions the auxiliary variable z is larger than $z_{\min}$ and smaller than $z_{\max}$. The system then acts as if the maximum and minimum controllers were not present and the input to the control system is $u = u_{\mathrm{n}}$. If the variable z goes above its upper limit $z_{\max}$ the error of the controller $C_{\max}$ becomes negative and the minimum selector chooses u_{h} instead of u_{n}. Control is then executed by the controller $C_{\max}$, which attempts to drive z towards $z_{\max}$. A similar situation occurs if the variable z becomes smaller than $z_{\min}$. The switches between the controllers are determined by the limits z_{min} and $z_{\max}$ and the gains of the secondary controllers, which often are proportional controllers.

Selector control is commonly used to provide safety, for example to maintain temperature while ensuring that a pressure does not exceed certain limits or to avoid stall in compressor control. We have only discussed maximum and minimum selectors, but there are also *median selectors* and *two-out-of-three selectors* that are used for high integrity systems with multiple sensors. Selector control is related

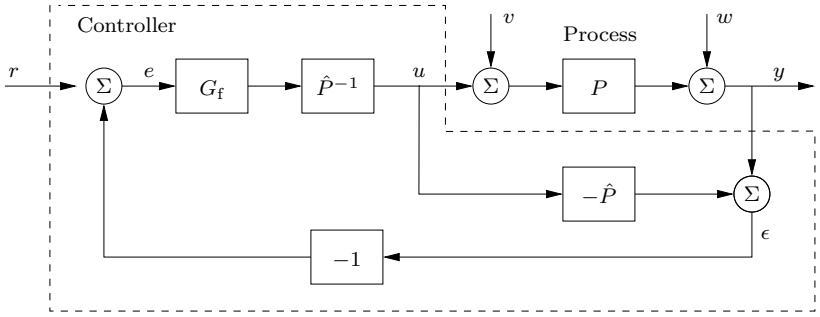


Figure 15.17: Block diagram of a closed loop system with a controller based on the internal model control structure.

to manual control, described briefly in Section 11.4, where the control variable is manipulated directly.

Design of the controllers can be made loop-by-loop since only one of the controllers is in operation at each time. There may, however, be complications when switching between the controllers. With controllers having integral action, it is necessary to track the integral states of those controllers that are not in operation. Windup protections therefore requires care. Selector control will be complicated when there are many constraints. It is then safer to use multivariable architectures and design methods such as model predictive control.

Internal Model Control – Disturbance Observer

Figure 15.17 shows a controller architecture that is called *internal model control* (IMC) or inferential control. The basic idea is to create an estimate of the effect of the disturbances v and w on the output of the system.

If $\hat{P} = P$ it follows from the block diagram that the control signal u has no effect on the signal ϵ, hence $\epsilon = w + Pv$. The signal ϵ is thus the net effect of the disturbances reduced to the process output, which explains the name disturbance observer. If $G_f = 1$ the block $\hat{P}^{-1}$ generates a control signal that eliminates the disturbance. In reality, ideal disturbance rejection cannot be accomplished because the inverse P^{-1} is normally not realizable. Good disturbance attenuation can, however, be achieved by using an approximate inverse and a proper design of G_f.

To investigate the response to reference signals, we neglect the disturbances v and w. If $\hat{P} = P$ we have $\epsilon = 0$ and the transfer function from reference to output becomes $G_{yr} = P\,\hat{P}^{-1}G_f = G_f$. In reality the inverse has to be substituted with an approximate inverse $P^\dagger$ because P normally cannot be inverted. The response to reference signals can be shaped by the transfer function $P^\dagger G_f$.

The block diagram in Figure 15.17 can be also be represented as a standard feedback loop with a process P and a controller C where

$$C = \frac{P^{-1}G_f}{1 - G_f}. \tag{15.5}$$

If $\hat{P} = P$, then the controller C cancels all process poles and zeros, which implies that it cannot be used for processes with unstable poles or zeros. The same

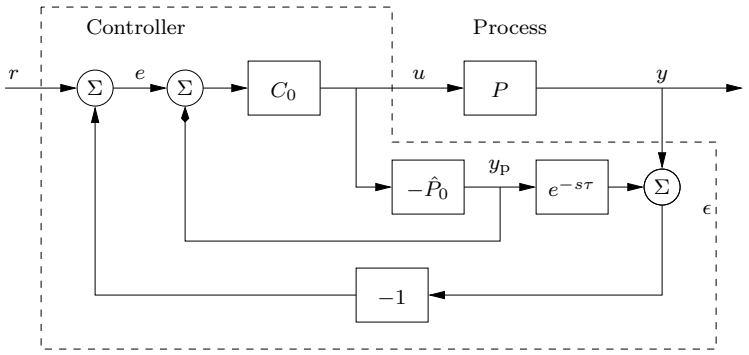


Figure 15.18: Block diagram of a closed loop system with a Smith predictor.

observation can be made from the fact that the system has a parallel paths with two identical systems, which is a prototype for a system lacking observability and reachability.

The Smith Predictor – Phase Advance

The Smith predictor is a special controller architecture for systems with time delays. A block diagram of the controller is shown in Figure 15.18. The controller is provided with a model $\hat{P} = \hat{P}_0\, e^{-s\tau}$, in parallel with the process $P = P_0\, e^{-s\tau}$. The parallel model provides the signal y_p, which is a proxy for the undelayed process output. Notice the similarity with the internal model controller architecture in Figure 15.17. Assuming that $\hat{P}_0 = P_0$ then the signal ϵ is zero for all u. Applying block diagram algebra then gives the following transfer function for the closed loop system:

$$G_{yr}(s) = \frac{P_0(s)\, C_0(s)}{1 + P_0(s)\, C_0(s)} e^{-s\tau}. \tag{15.6}$$

To obtain a desired response to reference signals we can design a controller C_0 for a process with the delay-free dynamics P_0. Notice that the architecture has two parallel paths with identical dynamics, which is particularly serious if the transfer function P_0 has unstable poles. The Smith predictor cannot be used for processes with integrators without modifications [ÅHL94].

To get additional insight into the properties of the Smith predictor, we observe that if $\hat{P} = P = P_0\, e^{-s\tau}$ the block diagram Figure 15.18 can be redrawn as a conventional feedback loop with the controller

$$C = \frac{C_0}{1 + C_0\, P_0(1 - e^{-s\tau})} = C_0\, C_\mathrm{pred}, \qquad C_\mathrm{pred} = \frac{1}{1 + C_0\, P_0(1 - e^{-s\tau})}. \tag{15.7}$$

The controller can thus be viewed as a cascade connection of the conventional controller C_0 with the predictor C_pred. Notice that near the gain crossover frequency for $P_0\, C_0$ we have $P_0\, C_0 \approx -1$ and $C_\mathrm{pred} \approx e^{sL}$, indicating that the transfer function C_pred has a significant phase advance.

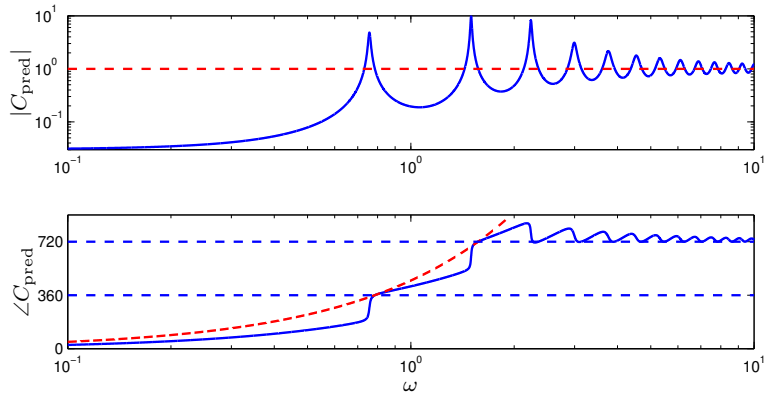


Figure 15.19: Bode plots of the predictor transfer function C_{pred} (solid curve) given by equation (15.7) and the ideal predictor $e^{s\tau}$ (dashed curves) for the transfer functions given in equation (15.8).

An example is shown in Figure 15.19 for the case where

$$P_0 = \frac{1}{s+1}, \qquad C_0 = \left(1 + \frac{1}{0.45s}\right), \qquad L = 8. \tag{15.8}$$

The phase curve of C_{pred} shows that the phase lead is very close to that of an ideal predictor $e^{s\tau}$ for the frequencies $\omega = 0.76$ and $\omega = 1.3$, where the phase lead is 360° and 720°. Also notice that the gain curve of the Bode plot has resonances at $\omega = 0.76$ and $\omega = 1.3$ and that the phase increases approximately 180° at those frequencies. This implies that the transfer function C_{pred} has two complex pole pairs in the right half-plane.

The Smith predictor gives closed loop systems with good responses to reference signals but the response to load disturbance responses are not much better than with PI control because the gain crossover frequency is limited by the time delay (around 0.1 for the system in Figure 15.19). This gain crossover frequency can also be obtained using a PI controller. The predictor C_{pred} is, however, a useful transfer function to provide large phase advance.

Complementary Filtering – Sensor Fusion

Complementary filtering is a technique that can be used to combine the information from different sensors, typically one sensor that is slow but accurate and another that is fast but drifting. One example is to fuse signals from a GPS with signals from gyroscopes and accelerometers to provide good estimates of position, velocity, and acceleration.

Consider the situation when we want to give a good estimate of the variable x and we have two sensors available that give the signals y_1 and y_2 where

$$y_1 = x + w_1, \qquad y_2 = x + w_2.$$

The disturbance w_1 has zero mean but the disturbance w_2 may drift. Using expo-

nential signals the complementary filter for recovering the signal x is given by

$$y_f = \frac{1}{s+1}\, y_1 + \frac{s}{s+1}\, y_2 = G_1(s)\, y_1 + G_2(s)\, y_2. \tag{15.9}$$

Notice that $G_1(s) + G_2(s) = 1$, which explains the name complementary filtering.

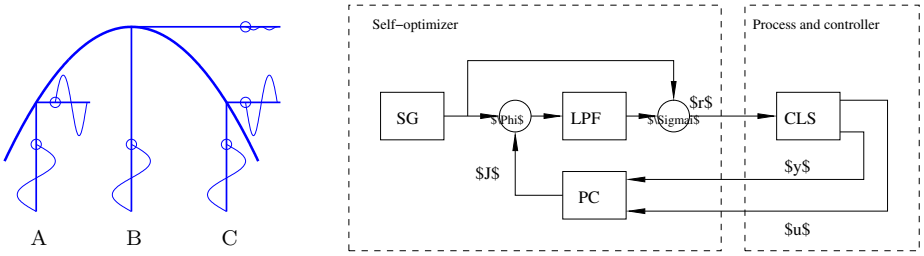
Both complementary filtering and Kalman filtering can provide improved estimates by fusing information from several sensors, and they can also be optimized if information about the noise characteristics are available. Complementary filtering requires only a model of the sensor system but the Kalman filter requires a model of the complete system dynamics. The Kalman filter can, however, also exploit the control actions. Both methods are widely used both in simple and advanced systems.

Extremum Seeking or Self-Optimization

Another set of useful control structures are *extremum seeking* or *self-optimizing controllers*. Instead of keeping the process output close to a constant reference value, self-optimizing controllers attempt to change the reference so that an objective function is minimized. The idea of extremum seeking is to adjust the reference of a closed loop system so that a performance criterion is optimized. This concept is illustrated in Figure 15.20a. The reference value r of the controller is changed sinusoidally and the performance j is observed. The performance changes very little close to the optimum, see B in the figure. They are in phase with the changes of the reference if the reference is to the left of the maximum (A in the figure), and they are out of phase if the reference is to the right of the maximum (C in the figure). The phase difference can be used to find the how the reference should be changed to optimize the performance. A block diagram of an extremal seeker is shown in Figure 15.20b. The self-optimizer has a block PC that calculates the perfjormance J from the process input u and output y. The signal generator SG generates a sinusoidal signal which is sent to the reference value of the closed loop system. The signal is correlated with the output J from the performance calculator and a low-pass filtered version is sent to the reference value. The perturbation signal should be chosen sufficiently slow so that process dynamics can be neglected. There are many other more sophisticated schemes based on optimization and estimation [Krs03]. They differ in how probing, analysis, and action are performed.

15.5 Interaction

A drawback with building a system loop by loop is that there may be unintended interactions. It is therefore important to investigate when interactions arise and what can be done to reduce potential drawbacks. We will start by introducing *the relative gain array (RGA)*, which is a simple measure of interaction. We will then discuss parallel systems, which is a special form of interaction that occurs when several subsystems produce the same effect. A typical example is an electric car with motors on each wheel or a power system with many generators that are synchronized for frequency control.



(a) Performance response (b) Block diagram

Figure 15.20: Self-optimizing control or extremum seeking. The steady state response of the performance J is shown as a function of the reference r in (a), together with the effects of sinusoidal variations in r. A block diagram of the system is shown in (b).

The Relative Gain Array

To explore the effects of interactions we will investigate control of a system loop by loop. The first problem, which is a prototype for a system that lacks reachability and observability, is to decide if y_i should be controlled by u_j or by some other control signal. This is called the *pairing problem (relative gain array)*. The second problem is to investigate if there will be interactions between the loops. It turns out that an understanding of the second problem also solves the first problem.

Consider a system with p inputs and p outputs. Let the transfer function matrix of the system be P. The open loop transfer function from input j to output i is (i, j)th element of the transfer function matrix, which we denote P_{ij}. This transfer function will change when the other loops are controlled. The change depends on the controllers in the other loops in a complicated way. A simple situation is when all other loops are *perfectly controlled* in the sense that all outputs y_k are zero for $k \neq i$. To find the transfer function from u_j to u_i in this case we use exponential signals, which gives

$$u_j = (P^{-1})_{ji}\, y_i.$$

The closed loop transfer function from u_j to y_j is thus $1/(P^{-1})_{ji}$. The relative gain for the loop ij is defined as the ratio of the transfer functions from u_j to y_i under open loop and ideal closed loop control, hence

$$\lambda_{ij} = P_{ij}(P^{-1})_{ji} = P_{ij}(P^{-T})_{ij},$$

where P^{-T} denotes the transpose of P^{-1}. The transfer functions λ_{ij} can be combined into the matrix

$$\Lambda = P \circ P^{-T}. \tag{15.10}$$

where $\circ$ denotes element by element multiplication of the matrices (the Hadamard product denoted $.*$ in MATLAB). The matrix Λ is called *the relative gain array* (RGA) or *Bristol's RGA* after its inventor [Bri66]. It was originally derived for the steady-state case ($s = 0$), which explains the name relative gain and it was later extended to dynamics. We illustrate with an example.

Example 15.2 Relative gain array for 2×2 systems
Consider a static system with two inputs and two outputs. The transfer function and its inverse are

$$P = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix}, \qquad P^{-1} = \frac{1}{p_{11}p_{22} - p_{12}p_{21}} \begin{pmatrix} p_{22} & -p_{12} \\ -p_{21} & p_{11} \end{pmatrix},$$

and the relative gain array (15.10) then becomes

$$\Lambda = \begin{pmatrix} 1-\lambda & \lambda \\ \lambda & 1-\lambda \end{pmatrix}, \qquad \lambda = \frac{p_{11}p_{22}}{p_{11}p_{22} - p_{12}p_{21}}.$$

In this case the interaction can thus be characterized by a single number λ. Notice that the relative gain array Λ has a special structure: the diagonal elements are equal to $1 - \lambda$ and all row and column sums are one. ∇

The matrix Λ given by equation (15.10) has special properties. Since P^{-1} is the inverse of P we have

$$\sum_{k=1}^{n} P_{ik}(P^{-1})_{kj} = \delta_{ij},$$

and hence

$$\sum_{i=1}^{n} P_{ij}(P^{-1})_{ji} = \delta_{ii} = 1.$$

The row and column sums of Λ are thus all equal to one, which implies that the interactions can be characterized by $(n-1)^2$ elements: one element for $m = 2$ as in Example 15.2 and four elements for $m = 3$.

The relative gain has a good physical interpretation in the static case. If $\lambda_{ij} = 1$ there is no interaction because the open and closed loop gains are the same. The interaction increases the gain if $\lambda_{ij} < 1$ and decreases the gain if $\lambda_{ij} > 0$. The interaction changes the sign of the gain if λ_{ij} is negative. The relative gain can also be used for pairing inputs and outputs as illustrated by the example.

Example 15.3 RGA and pairing
Consider a system where the static gain matrix and the RGA are

$$P = \begin{pmatrix} 2.662 & 8.351 & 8.351 \\ 0.382 & -0.559 & -0.559 \\ 0 & 11.896 & -0.351 \end{pmatrix}, \qquad \Lambda = \begin{pmatrix} 0.32 & 0.02 & \mathbf{0.66} \\ \mathbf{0.68} & 0.01 & 0.31 \\ 0 & \mathbf{0.97} & 0.03 \end{pmatrix},$$

where the bold entries are the maximum entries in each row. In this case the pairing is straightforward because there is one largest relative gain in each row and each column, which gives the pairing $y_1 : u_3$, $y_2 : u_1$, and $y_3 : u_2$. The relative gains are also reasonably large. ∇

The relative gain array is a simple measure of interaction, it is dimension free, easy to compute and it gives insight into interactions and pairing of variables. It was originally derived for static systems but analysis of the frequency response $\Lambda(i\omega)$ gives insight into the frequency dependence of the interactions. The RGA also gives information about the variables that should be grouped for multivariable control. Since it was derived under the special assumption of perfect control, it does not capture all aspects of interaction.

Parallel Systems

There are situations when several subsystems are used to control the same variable. Typical examples include temperature control using several cooling or heating devices and control of an electric car with one motor on each wheel. An extreme example is control of a power grid, which may have hundreds of energy sources that all contribute to maintain frequency and voltage of the net. Designing a system loop by loop requires care as is illustrated by the following example.

Example 15.4 Cruise control for electric car
Consider speed control of an electrical car with motors on each wheel. For simplicity we will consider linear motion with only two motors, and we will use the simple model (4.1) in Section 4.1. Neglecting all disturbance forces except the force due to gravity, $F_{\mathrm{d}} = mg\theta$, the model (4.3) becomes

$$m\frac{dv}{dt} = F_1 + F_2 - mg\theta, \tag{15.11}$$

where v is the speed of the car, θ is the slope of the road, and F_1 and F_2 are the forces generated by each tire.

For simplicity we will neglect the dynamics of motors and wheels, so that the forces are simply the output of the wheel controllers. When both wheels have proportional controllers we have

$$F_1 = k_{\mathrm{p1}}(r - v), \qquad F_2 = k_{\mathrm{p2}}(r - v), \tag{15.12}$$

where r is the speed reference. Combining equations (15.11) and (15.12) gives the following equation for the closed loop system:

$$m\frac{dv}{dt} = (k_{\mathrm{p1}} + k_{\mathrm{p2}})(r - v) - mg\theta = k_{\mathrm{p}}(r - v) - mg\theta,$$

where $k_{\mathrm{p}} = k_{\mathrm{p1}} + k_{\mathrm{p2}}$. If the slope θ is constant there will be a steady-state error $e_{\mathrm{ss}} = r - v_{\mathrm{ss}} = mg\theta/k_{\mathrm{p}}$ and the steady-state forces are then

$$F_{1,\mathrm{ss}} = \frac{k_{\mathrm{p1}}}{k_{\mathrm{p1}} + k_{\mathrm{p2}}} mg\theta, \qquad F_{2,\mathrm{ss}} = \frac{k_{\mathrm{p2}}}{k_{\mathrm{p1}} + k_{\mathrm{p2}}} mg\theta.$$

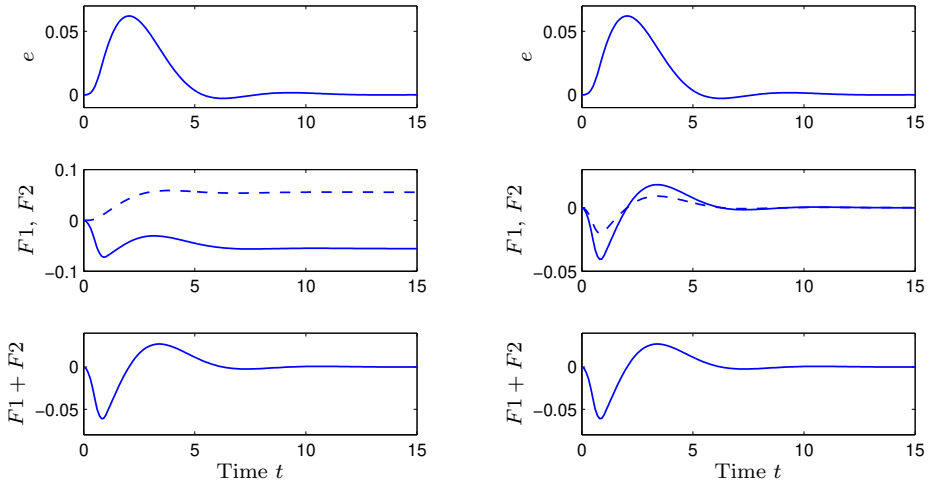
The proportional gains k_{p1} and k_{p2} of the controllers thus determine how the *compensation for the disturbance is distributed among the motors.*

Next we will consider the case when each motor drive is provided with a PI controller. The closed loop system is then described by the equations

$$\begin{aligned} m\frac{dv}{dt} &= (k_{\mathrm{p1}} + k_{\mathrm{p2}})(r - v) + k_{\mathrm{i1}} I_1 + k_{\mathrm{i2}} I_2 - mg\theta, \\ \frac{dI_1}{dt} &= r - v, \qquad \frac{dI_2}{dt} = r - v. \end{aligned} \tag{15.13}$$

Assuming that r and θ are constant, the equilibrium point is given by $v = r$ and constant I_1 and I_2. The variables I_1 and I_2 are not unique, however, and they can have any values as long as

$$k_{\mathrm{p}} I_1 + k_{\mathrm{i2}} I_2 = mg\theta.$$



(a) Drive system with two PI controllers (b) Drive system with one PI controller

Figure 15.21: Transient behavior of car with two controlled wheels. The system in (a) has a PI controller for each wheel drive. The system in (b) has one PI controller whose output is distributed to the wheel drives. The top plots show the velocity error $e = r - v$, the middle plots show the forces F_1 and F_2 generated by the wheels and the bottom plots show the total force $F_1 + F_2$.

The fact that the closed loop system has infinitely many equilibrium points is an indication that there may be difficulties. To explore this we will analyze the closed loop system. The transfer functions from r and θ to v are

$$G_{vr} = \frac{k_\mathrm{p} s + k_\mathrm{i}}{m s^2 + k_\mathrm{p} s + k_\mathrm{i}}, \qquad G_{v\theta} = \frac{mgs}{m s^2 + k_\mathrm{p} s + k_\mathrm{i}}, \tag{15.14}$$

where $k_\mathrm{i} = k_{\mathrm{i}1} + k_{\mathrm{i}2}$. The velocity v follows reference signals without steady-state error since $G_{vr}(0) = 1$ and there will be no steady-state error when encountering a slope since $G_{v\theta}(0) = 0$.

The system (15.13) is of third order but the transfer functions (15.14) are of second order, which means that there is a pole/zero cancellation. The canceled mode is governed by the equation $d(I_1 - I_2)/dt = 0$, which has an eigenvalue at the origin. The system has a Kalman decomposition (Figure 8.9a) with a first-order subsystem $\Sigma_{\overline{\mathrm{ro}}}$ with integrator dynamics that is neither reachable from the forces F_1 and F_2 nor observable from v. Recall that a system with two parallel systems having the same mode is a prototype for a system that is neither reachable nor observable.

The fact that the closed loop system has an unstable mode that is neither reachable nor observable is seen in the simulation in Figure 15.21a. The simulation shows what happens when a pulse like perturbation is introduced at the input of the PI controller for wheel 1. The velocity error $e = r - v$ is quite well behaved and the error settles in about 10 time units. The force F_1 (solid curves) reacts quickly to reduce the effect of the disturbance but the force F_2 (dashed curves) goes in the opposite direction. In steady state the force F_1 settles to a constant value as does

F_2. The steady-state forces have different signs, however, which means that one wheel is driving and the other is braking: clearly not a satisfactory behavior.

The irregular behavior is caused by an unreachable and unobservable mode in the closed loop system, which caused by two PI controllers in parallel. Having understood what happens, it is straightforward to find a remedy: use one PI controller and distribute its output to the two wheels. The controller is then

$$F_1 = \alpha\big(k_{\mathrm{p}}(r-v) + k_{\mathrm{i}} I\big), \quad F_2 = (1-\alpha)\big(k_{\mathrm{p}}(r-v) + k_{\mathrm{i}} I\big), \quad \frac{dI}{dt} = r - v, \tag{15.15}$$

where the parameter $0 < \alpha < 1$ tells how the forces are distributed between the wheels. The closed loop system is described by

$$\frac{dv}{dt} = k_{\mathrm{p}}(r-v) + k_{\mathrm{i}} I - mg\theta, \qquad \frac{dI}{dt} = r - v, \tag{15.16}$$

and the forces are given by equation (15.15). This system is of second order and the transfer functions G_{vr} and $G_{v\theta}$ from reference r and slope θ to velocity v are given by equation (15.14).

A simulation of the system is shown in Figure 15.21b. The behavior of the forces are now much more reasonable because they both collaborate to reduce the disturbance. The behavior of the error e and the total force F are, however, the same as for the system with two PI controllers. ∇

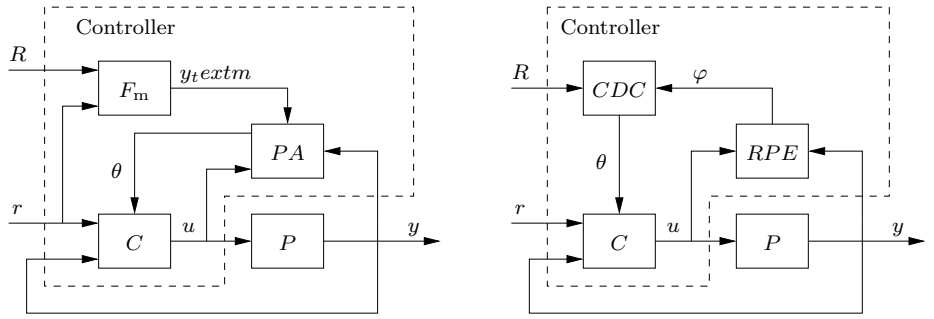
The conclusion from the example can be generalized. If parallel systems are controlled by proportional controllers, then the controller gains determine how disturbance attenuation is divided among the subsystems. However, integral control cannot be used in the individual subsystems. Instead we can use a central integrator and distribute its output to the separate controllers for the subsystems.

15.6 Adaptation and Learning

In this section we will briefly discuss control systems with abilities to adapt, learn, and reason. Adaptation is used to adjust to a specified use or situation, learning is used to acquire knowledge or skill by study, instruction, or experience, and reasoning is the intellectual process of seeking truth or knowledge by inferring from either fact or logic. Cars and air vehicles with autonomy are areas where adaptation, learning, and reasoning are essential, but it should be recognized that the abilities of humans are still far ahead of what is achievable using human-engineered systems.

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 regulator are two common adaptive systems: their block diagrams are shown in Figure 15.22. Notice that in both cases there are two feedback loops: one conventional feedback loop involving the process P and the controller C and a slower loop that adjusts the parameters θ of the controller.



(a) Model reference adaptive control (b) Self-tuning regulator

Figure 15.22: Block diagrams of systems with (a) a model reference adaptive controller (MRAC) and (b) a self-tuning regulator (STR). The block P is the process, C is a controller with adjustable parameters. For the model reference system the requirements R are given in terms of the model F_m, which gives the ideal response to reference signals. The controller parameters θ are adjusted by the parameter adjustment mechanism PA. In the self-tuning regulator the controller parameters θ are adjusted indirectly based on a control design calculation CDC, where the process model is obtained by a recursive parameter estimator RPE.

Model reference adaptive control (MRAC) is primarily used for reference signal tracking. A block diagram of the controller is shown in Figure 15.22a. The controller consists of three blocks F_m, C, and PA. The desired response to command signals is given by the transfer function F_m, which is shaped to satisfy the requirements R. The controller C has adjustable parameters θ. The parameter adjustment mechanism PA, receives the process input u, the process output y, and the desired response y_m, and it generates the process parameters θ. A simple parameter adjustment mechanism is given by the "MIT rule" [ÅW08b, Section 5.2]:

$$\frac{d\theta}{dt} = -\gamma\, e\, \frac{\partial e}{\partial \theta}, \tag{15.17}$$

where γ is a parameter, $e = y_\mathrm{m} - y$, and $\partial e/\partial\theta$ is a sensitivity derivative. The MIT rule is a very simple way to adjust the parameters. There are many other rules, some of them are derived from Lyapunov theory [ÅW08b].

The self-tuning regulator (STR) is used both for reference signal tracking and for regulation. The controller is based on the idea of developing a process model automatically and applying some design method to find a suitable controller. A block diagram of a system is shown in Figure 15.22b. The controller has three blocks: a controller C with adjustable parameters θ, a recursive parameter estimator RPE and a controller design calculation CDC. The parameter estimator RPE estimates the process parameters θ recursively from the process input u and output y. The controller design block CDC determines the controller parameters from the process parameters using some design method. In this calculation it is common to treat the estimates as the true parameters, a principle from decision making under uncertainty called the *certainty equivalence principle* [Sim56]. Uncertainties in the estimates can be taken into account because many estimation schemes provide estimates of parameter uncertainty. The self-tuning regulator is very flexible because

many different methods can be used for parameter estimation and control design.

Recursive least squares is a common method for estimating parameters in the model

$$\begin{aligned} y_{t+1} &= -a_1 y_t - a_2 y_{t-1} + \cdots + b_1 u_t + \cdots + e_{t+1} = \varphi_t \theta + e_{t+1}, \\ \varphi_t &= [-y_t \ -y_{t-1} \ \cdots \ u_t \ u_{t-1} \ \cdots], \qquad \theta = [a_1 \ a_2 \ \cdots \ b_1 \ b_2 \ \cdots]^T \end{aligned} \tag{15.18}$$

by minimizing the mean square error $\sum e_k^2$. The estimates are given by

$$\begin{aligned} \hat{\theta}_t &= \hat{\theta}_{t-1} + K_t(y_t - \varphi_t \hat{\theta}_{t-1}^T), \qquad K_t = P_t \varphi_t^T, \\ P_t &= P_{t-1}\varphi_t^T(\lambda + \varphi_t P_{t-1} \varphi_t^T)^{-1}, \end{aligned} \tag{15.19}$$

which is a special case of the Kalman filter: see equation (8.8) in Section 8.2. The parameter λ controls how quickly old data is forgotten. There are many variations of the parameter estimator: directional forgetting and square root algorithms, where the square root of P is updated instead of P itself [ÅW08b], are of particular interest.

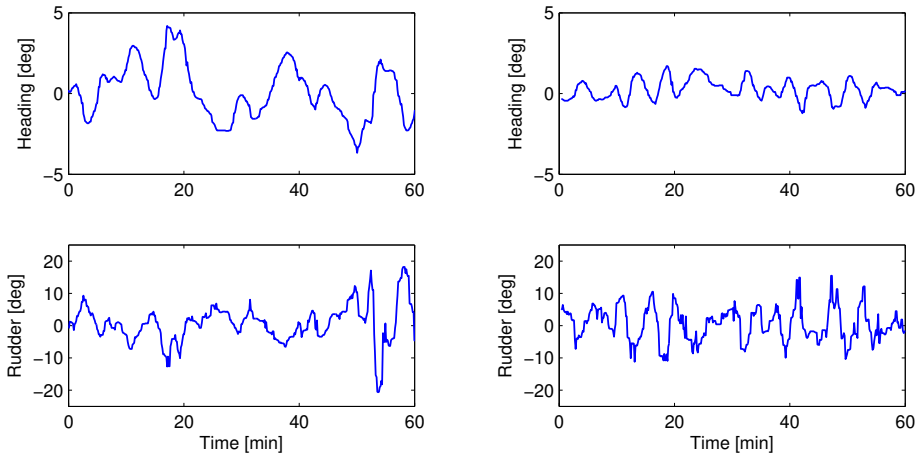
Applications of adaptive control are found in flight control, process control, and wheel-slip control and other automotive applications. We illustrate by a ship steering application.

Example 15.5 Adaptive ship steering
A conventional autopilot for ship steering is typically based on PID control. The major disturbances are due to wind and waves, which can change significantly. An adaptive controller can model wave generated forces and counteract them efficiently. Ship dynamics and wave forces can be captured by a model of the form (15.18) with 4 a-parameters, constrained to contain an integrator, and 2 b-parameters. Extensive sea trials have shown that the adaptive autopilot has better performance than the conventional autopilot in normal whether conditions and substantially better performance in bad weather conditions. Figure 15.23 shows results from evaluation of the Steermaster autopilot developed by Kockums and now marketed by Northrop Grumman. In the experiments, the conventional and adaptive autopilots were run repeatedly for about an hour each during normal operation. The figure shows that the adaptive autopilot has significantly smaller variations in heading than the conventional autopilot. The difference corresponds to about 3% less fuel consumption. ∇

A difficulty with adaptive control is that parameter estimation is performed when the system is in closed loop. It is then important where the excitation signals occur. Consider for example the standard feedback loop in Figure 15.3. If the only perturbation on the system is the injected signal δ_1 and we assume $r = 0$, $v = 0$ and $w = 0$, we get

$$y = \frac{P(s)}{1 + P(s)C(s)}\delta_1, \qquad u = -\frac{C(s)P(s)}{1 + P(s)C(s)}\delta_1.$$

It then follows that $y = -\frac{1}{C(s)}\, u$ and any attempt to find a model relating u and y will thus result in the negative inverse of the controller transfer function. However,



(a) Conventional autopilot (b) Self-tuning autopilot

Figure 15.23: Heading deviation (top) and rudder motion (bottom) for ship steering for (a) a conventional autopilot and (b) an adaptive autopilot. The experiments were performed on a 225000 ton tanker Seascape, wind velocity around 10 knots, from [KÅT+79].

if we inject disturbances through the reference r and set all other inputs to zero we have instead

$$y = \frac{P(s)C(s)}{1 + P(s)C(s)}F(s)r, \qquad u = \frac{C(s)}{1 + P(s)C(s)}F(s)r.$$

Hence, $y = P(s)u$ and the process model can indeed be estimated.

To obtain reliable estimates of the process parameters, there must be sufficient variations in the control signal. This can be captured by the notion of *persistent excitation*. A signal $u(t)$ is persistently exciting of order n if

$$U = \lim_{t\to\infty} \frac{1}{T} \int_0^T (A(p)u(k))^2 > 0$$

for all nonzero polynomials A of the differential operator $p = d/dt$ with degree less than or equal to $n - 1$. Persistency of excitation determines the number of parameters that can be estimated reliably. A constant is persistently exciting of order 1 and permits estimation of one parameter. A sinusoid is persistently exciting of order two.

To have a reliable parameter estimation it is important to be aware of where disturbances enter and to monitor the excitation. Load disturbances of the process are particularly harmful. A scheme for detecting harmful load disturbances is presented in [THÅ00]. To obtain reliable estimates it is necessary to monitor the excitation of the process and only update parameters when there is sufficient excitation of the process.

An interesting approach to control of uncertain systems was proposed by Feldbaum [Fel65], who emphasized that control should be *investigating* as well as *directing*, and he coined the term dual control for this property. Feldbaum used optimal

stochastic control to obtain a controller that was actively introducing perturbations in the process when the process was not properly excited by natural disturbances. The *hyper state* of a dual controller is the conditional probability distribution of the regular states of the process and the parameters. The computations of a dual controller can only be performed in simple cases because the state of the system is a conditional probability distribution over states and parameters [ÅH86]. Many heuristic schemes to monitor excitation and to introduce perturbations when needed have therefore been developed. In the field of machine learning this approach is called reinforcement learning.

Learning

A nonlinear function with a learning mechanism is a simple example of a learning system. Learning can be done in two different ways. In supervised learning, the function is created automatically by providing it with a large number of arguments and corresponding function values. In reinforcement learning a criterion for good fit is provided and learning is executed by selecting random arguments and changing the function until a good fit is provided. Representation of the learning mechanism is a central issue. A simple way to represent a function of several variables is to quantize the variables, which we illustrate by an example of unsupervised learning.

Example 15.6 Michie's BOXES method
Michie and Chambers developed a simple learning program called BOXES [MC68] for game playing. An early connection between learning and control was established when the program was applied to the classical control problem of stabilizing an inverted pendulum. Consider a cart–pendulum system such as that in Figure 7.2b on page 7-5, where cart position and velocity and pendulum angle and angular velocity are measured. The control signal is a function: $f : \mathbb{R}^4 \to \mathbb{R}$. BOXES was used to learn an approximation of this function. To implement the system the states were quantized crudely: 5 levels for position and angle, 3 levels for velocities, and 2 levels left (L) or right (R) for the control signal. The control law is then represented by a table with $5 \times 5 \times 3 \times 3 = 225$ entries. Each entry in the table has five numbers: LL (left life), RL (right life), LU (left usage), RU (right usage), which capture features of past experiments, and T (target), which is related to the current mean time for stabilization. The life (L) is a weighted average of the number of times the entry was used before failure. The usage (U) is a weighted number of times the entry was used before failure. The control actions taken upon entering a box is a heuristic function of the values of LL, RL, LU, RU, and T. The system is initialized by introducing random numbers in the table. Experiments are run and the table is updated. In a typical experiment the system was able to stabilize the pendulum in 25 minutes after a 60 hour training period. ∇

Pendulum stabilization is perhaps not the best case to demonstrate learning, since a student could design a stabilizing controller in less than an hour. Control performance will also be better because a conventional design can avoid the crude quantization used in BOXES. There have, however, been significant developments in machine learning since the late 1960s when BOXES was developed.

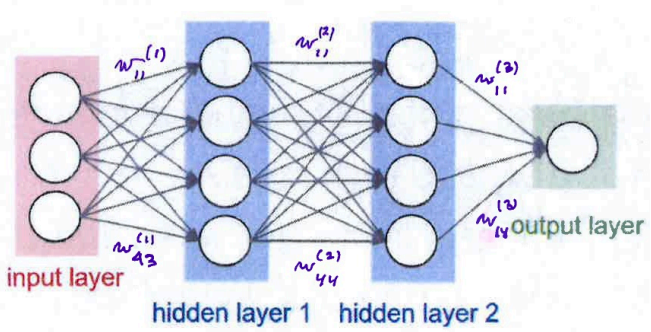


Figure 15.24: Schematic diagram of a feedforward neural network with an input layer, a hidden layer, and an output layer.

Neural Networks

A severe drawback of schemes like BOXES is that the nonlinear function is represented by gridding the state variables, which requires very large tables. To have efficient learning schemes it is necessary to find more efficient ways to represent nonlinear functions and to find efficient learning mechanisms. An artificial neural network is such a representation.

Artificial neural networks were inspired by neuroscience although their current implementation is far from their biological origin. A neuron has many synapses which receives inhibitory or excitatory signals from other neurons. The neuron emits a pulse to other neurons if the net excitation over a short time interval is above a certain level. An *artificial neuron* mimics a real neuron but it operates continuously. A very simple model is

$$y = g\Big(\sum_{k=1}^{n} w_i u_i\Big), \tag{15.20}$$

where the parameters w_i are weights. The function g was originally a sigmoid shaped function, for example $g(x) = \tanh x$ or $g(x) = (1+e^{-x})^{-1}$, but many other functions are currently used, such as $\max(0, x)$, $e^{-x^T Q x}$, and $(1 + x^T Q x)^{-1}$.

An *artifical neural network (ANN)* is a combination of neurons in a layered network as shown in Figure 15.24, which represents the function $f : \mathbb{R}^3 \to \mathbb{R}^2$ as

$$f(u) = g\Big(W^{(3)} g\big(W^{(2)} g(W^{(1)} u)\big)\Big), \tag{15.21}$$

where $W^{(k)}$ is a matrix of the weights to the layer k, and g is a monotone function of the type discussed above (when x is a vector, $g(x)$ is a vector obtained when the function is applied to each element of the vector). The neural network may appear very restricted, but Kolmogorov proved that any continuous function on a finite cube can be represented by a neural network with only one hidden layer [Kol57]. The strong advantage of having many layers was clarified by Håstad [Hås87].

Neural networks have weights $w_{ij}^{(k)}$, which are determined experimentally by matching a large number of arguments and corresponding function values. A useful

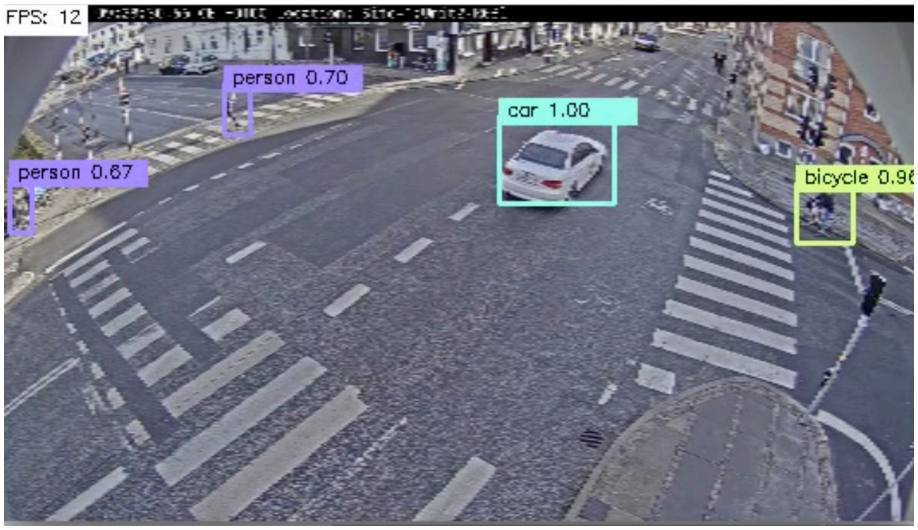


Figure 15.25: Illustration of feature detection based on deep learning. Objects are detected and classified as car, bicycle, or person, their position and spatial extent are also determined. The images typically have low-resolution to prevent identification of license plates and humans. The particular network which generated this figure has 13 layers and more than 25 million parameters [AJÅ+17].

feature is that both the function and its inverse can be generated from data. The weights in a simple neuron can be determined by the gradient algorithm

$$w_i(k+1) = w_i(k) + \gamma u_i^0(k)\big(y^0(k) - y(k)\big), \tag{15.22}$$

where u_i^0 and y^0 are training data and y is computed from equation (15.20) with $u = u^0$. This rule can be interpreted as an approximation of a gradient scheme for minimizing the mean square error. It is similar to the MIT rule (15.17) for model reference adaptive control. In neurophysiology the algorithm (15.22) is known as Hebb's rule [Heb49]. Parameters of multi-layer neural networks can be updated by gradient descent, where the gradient is computed by back propagation..

Deep Learning

Neural networks with many layers, so-called *deep learning*, have proven very useful in many fields. Preprocessing of data from information-rich sensors, such as spectrometers and cameras, is particularly useful for control. For example, in autonomous driving it is useful to recognize objects such as houses, road markings, traffic lights, cars, bicycles, and pedestrians, and to track these objects in real time.

A typical scene is illustrated in Figure 15.25, where object recognition is made by a *convolutional neural network (CNN)*. An image is represented as an $m \times n$ array and an additional index k is used to represent images of different colors (red, green, and blue) or different features (lines, corners, wheels, cars, bicycles, etc.). Artificial neural networks have only two operations—a weighted summation and a monotone function of a scalar variable—while the convolutional neural networks have more operations.

The *convolution* (C) function acts on a three-dimensional array and generates a three-dimensional array. The function can be written $y = C(x)$, where

$$y(i,j,k) = w_o(k) + \sum_u \sum_v \sum_l x(i-u, j-v, l) w(u,v,l,k).$$

The function depends on the bias term w_o and the weights w. Convolution is used to detect features, such as edges, corners, wheels, cars, bicycles, etc.

The *rectified linear unit* (ReLU) R operates on an array and generates an array of the same size. The function can be written $y = R(x)$, where

$$y(i,j,k) = \max\big(x(i,j,k), 0\big).$$

It operates element-wise on the image. The function is analogous to the monotone function used in conventional neural networks.

The *max-pooling* function (P) operates on an array x of size $m \times n \times k$ and generates an array y of size $m/2 \times n/2 \times k$. The function can be written $y = P(x)$, where

$$y(i,j,k) = \max\big(x(2i,2j,k), x(2i-1,2j,k), x(2i,2j-1,k), x(2i-1,2j-1,k)\big).$$

Max-pooling is also called down sampling or subsampling, reduces the dimension of an array but retains the most important features of the image. There are also other pooling operators such as minimum and average pooling.

The function *Softmax* (S) operates on an array and generates an array of the same size. The function can be written $y = S(x)$, where

$$y(i,j,k) = \frac{e^{x(i,j,k)}}{\sum_l e^{x(i,j,l)}}, \qquad \sum_k y(i,j,k) = 1.$$

The entries of the output are in the range $[0,1]$, they indicate the subjective probability of finding the object k in position ij. The function is typically used in the final layer to assign a subjective probability to a detected feature.

A convolutional neural network can be large with many parameters, but parameters appear only in the convolution layer. The parameters are determined by optimization based on large training sets.

Determination of the sizes of the arrays, the number of layers and the order of the operators is an art that requires experience. In computer vision it is inspired by earlier efforts based on detection of lines, objects, corners, and blobs.

Figure 15.26 shows a simplified representation of the network used to generate Figure 15.25. The input to the network is a box with 3 RGB planes with 300×300 pixels and the output is a box of three images of size 20×20 pixels each representing a recognized object. The network has 25 million parameters which have to be estimated. The network can be described by the function

$$Q = S \circ C \circ R \circ C \circ (P \circ R \circ C)^3,$$

where $f \circ g$ denotes function composition, i.e. $f \circ g(x) = f(g(x))$. The function is a composition of 13 functions. The first operations creates a wide range of patterns using a combination of convolution (C), ReLU (R), and max-pooling (P).

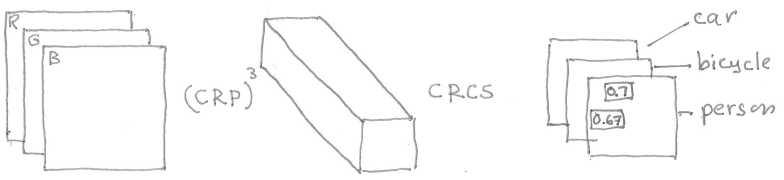


Figure 15.26: A simplified version of the convolutional neural network used for the object detection in Figure 15.25. The input is a color image. The output is three layers of images representing the subjective probability of finding car, bicycle, or person at a given position in the image.

Recognition of the final objects car, bicycle, and pedestian is done by $S \circ C \circ R \circ C$. The final result three images with numbers that indicate the subjective probability of finding an object at a particular location.

The arrays are illustrated by boxes in the figure. The first layer consists of three arrays of red, blue, and green pixels of the object. The convolution operates sequentially on part of the picture, the kernel is typically a fraction of the image size. Different convolution kernels are applied to detection edges, corners, and other features. The result of for each kernel is stored in a separate layer. Convolution is followed by a rectified linear operator. The size of the image is then reduced using max-pooling (P). The operations C, R, and P are repeated several times to create more features. The final part consists of three layers generated by C, R, and C. The final result, a classification as car, bicycle, pedestrian, or no object, is obtained by applying the softmax operator. The final result can be mapped on the original image as shown in Figure 15.26. There are several ways to improve the position of the objects and their sizes.

A camera with a convolutional neural network can be regarded as a trainable sensor that will detect, classify, and position objects such as cars, bicycles, and people in real time. It is clearly a useful component for autonomous driving and for other applications in which vision-based sensors are used. From the data it is easy to generate warnings, zones where a vehicle can safely enter, and other useful information. The computations are fast, since they only require evaluation of simple functions; computations can also be parallellized. The network used to generate Figure 15.25 has more than 12 million parameters and it estimates 11 objects. The final trained network can be executed in 60 Hz on a standard PC with an NVIDIA Titan X graphics card.

15.7 Control Design in Common Application Fields

Control is sometimes called the *hidden technology* because it is successfully used practically everywhere without being noticed [Åst99]. In this section we will present the role of control in four different applications fields, which provides a flavor of the commercial landscape of controls, the systems, the controllers, and the users.

Aerospace – High Performance Systems and Highly Skilled Users

The aerospace industry was an early user of control. The Wright brothers flew in 1905 because they had a good insight into dynamics and control. The first autopilot was designed by Sperry in 1914 and autonomous flight was demonstrated in 1947. Today aerospace is a flourishing application area for control, dominated by large companies for civil and military markets. The industry produces airplanes, helicopters, drones, satellites, rockets, missiles, and quadrocopters as well as infrastructure for flight control, which includes air traffic control and automatic landing systems. An indication of the size of the business is that about 2500 aircraft were produced in 2015, generating revenues over 20 billion dollars, and that is only one part of the industry. Aerospace companies typically have large central groups for systems and control.

Typical aerospace systems are operated by highly selected, skilled pilots and astronauts, who are well-trained in simulators and interact with the system by direct manipulation of actuators and reference signals or by changing operating modes. A consequence is that the systems are designed so that the user can directly influence the system in many different ways. Sometimes users have taken over operation of the system and saved the mission, as was done in Apollo 13 [Min08].

The aerospace industry has been a technology driver with new hardware and control techniques emerging from the industry. There are extreme requirements on safety, which has led to the practice of redundant systems and components, since adopted by the automotive industry. The industry pioneered the use of simulation and model-based systems engineering, development of high precision accelerometers and gyroscopes, and anti-lock braking (used on aircraft as early as 1929). Extremal control was first applied to control of aircraft engines already in 1951. Wide variations of operating conditions stimulated the development of gain scheduling and adaptive control. Optimal control and Kalman filtering were used in the early space efforts. Nonlinear control was used extensively in control of satellites. Unmanned air vehicles were used operationally already in 1970.

Automotive – Complex Systems Used by Ordinary People

The automotive industry is a multi-trillion dollar business. It is dominated by six large companies and many subcontractors. Control is used extensively both in the cars themselves and in the manufacturing of cars. Automobiles are used by ordinary people who interact with the system by changing modes and setpoints and by direct actuation. Control is executed using electronic control units (ECUs), microprocessors with input/output interfaces. Modern cars have more than 100 electronic control units.

Servo-assisted power braking was used in racing cars in 1914 and became commonly used in the 1920s. Computer control was introduced in the late 1970s to cope with the stringent emission requirements. Once computers were introduced they were applied to more functions: suspension control, anti-lock braking systems (ABS), electronic braking systems (EBS) and electronic stability control (ESC). These systems used accelerometers and gyroscopes to control the brakes individually to improve stability and steering. Adaptive cruise control, based on radar

sensors, maintains a constant distance to the car in front. The excellent experience with these systems inspired car manufacturers to introduce more sophisticated systems such as collision avoidance and parking assist. Autonomous driving is well on its way. Control is a key element both on its own but also in combination with computer vision.

Model-based systems engineering is used extensively to improve the efficiency of engineering. The design of the Toyota Prius is an example where modeling and simulation replaced much of the traditional testing using hardware prototypes. Another example is design of climate control systems. The major European car manufacturers and their component suppliers have created an infrastructure for model-based design where the suppliers deliver components with validated dynamical models enabling the car manufacturers to simulate complete systems (e.g. [LBSP05]).

A worldwide development partnership of vehicle manufacturers, suppliers, and software companies called AUTOSAR (AUTomotive Open System Architecture) was formed in 2003 (see http://www.autosar.org). Standards that enable modularity, scalability, transferability, and reusability of functions have been created, providing a standardized platform for automotive software systems. The standard enables system-wide configuration and optimization to meet run-time requirements of automotive devices.

The large size of the automotive industry provides a mass market for a wide range of industries to develop components and subsystems. The industry stimulated the development of inexpensive emission sensors, accelerometers, and gyroscopes, and even more importantly the microcontroller and the programmable logic controller (PLC).

Early manufacturing systems were automation systems controlled by relays for logic and sequencing. General Motors challenged the electronics industry with requirements for a standard machine controller that could replace the relays resulting in the PLC. The system architecture is based on round robin schedulers with different cycle rates. PLCs were originally programmed in a graphical language called ladder diagrams (LD), which emulated the ladder logic used to describe relay circuits. Several different programming styles were later standardized: function block diagrams (FBD), sequential function charts (SFC), and structured text (ST). PLCs developed rapidly and became a standard tool for automation in many industries.

Process Industry – Complex Systems with Many Different Users

Process control provides automation for a variety of industries such as chemicals, oil refining, pulp and paper, pharmaceuticals power plants and many others. A characteristic feature of the industry is that control and automation is typically delivered by special companies. This began with instrument companies that developed sensors, recorders, and controllers, including Taylor Instruments, founded in 1851, and Foxboro, founded in 1908. By the mid 1930s there were many companies who supplied sensors, actuators, and controllers to the process industry.

It is normally difficult to find out how much of the turnover of a business is related to control. In process control this data is available, since automation is done by special companies. Control and automation is a 100 billion dollar industry. Distributed control systems account for about 20% of the market, the rest is sensors,

actuators, software, and other components. Five dominating suppliers have more than 50% of the market.

Functions of control and logic and sequencing are essential for process operation. Early process control systems had cabinets with analog controllers for regulation and cabinets with relays for logic and sequencing governing startup, shut down, and equipment protection. As technology developed the relays were replaced by programmable logic controllers (PLCs), originating in the automotive industry, and the analog controllers were replaced by distributed control systems (DCS).

DCS is now the standard tool to provide control in the process industry, as illustrated in Figure 15.6. It has facilities for connecting sensors, actuators, and algorithms and can be viewed as a toolbox for implementing control systems. It is interesting to note that ExxonMobil has recently contracted Lockheed Martin to specify the next generation of distributed control system for process control. The system will be open, secure, and based on standards, leveraging experiences from the Future Airborne Capability Environment (FACE) consortium in the aerospace industry [Ope14].

Process control systems typically have thousands of sensors and actuators, and the systems are also widely distributed geographically. Sensors and actuators are connected to the DCS system by standardized networks (IEC 61784).

Valves are commonly used for actuation in process control. It is customary to have cascade control with inner analog loops with valve positioners to reduce effects of friction and nonlinearities at the lowest level of the hierarchy, and feedback loops for control of pressure and temperature and quality variables at the higher levels.

There are several different types of users of the DCS: the plant managers who set production schedules and directs equipment maintenance, the process engineers who select, configure, and modify the system, the instrument engineers who tune controllers and maintain sensors and actuators and the operators who supervise the operation of the system (see Figure 15.6).

Distributed control systems have many control algorithms that easily can be configured using graphical interfaces. The control algorithms are implemented by process and instrument engineers both by company personnel and by consultants. Controllers are tuned during operation and the system is occasionally reconfigured. Algorithms and languages are standardized by international committees. A wide range of standards for control and automation are set by the International Society of Automation (ISA) and the International Electrical Commission (IEC). There are also some standards for communication organized by special groups.

Although PID control was used in many fields, the major development of the controller and its tuning procedure occurred in the process industries. Most of the controllers (typically 97% [DM02b]) are PID controllers, only a small fraction of them using derivative action. A recent investigation of 100 boiler-turbine units in the Guangdong Province in China showed 94.4% PI, 3.7% PID, and 1.9% advanced controllers [SLL16] .

Control paradigms such as cascade, selector, and midrange control are common, as are gain scheduling, automatic tuning, and model predictive control. Model predictive control emerged from efforts at the Shell oil company to develop effective techniques for control of multi-variable processes. It was originally called dynamic matrix control [CR80, JBRM99].

Telecommunication – Billions of Systems

Black's invention of the negative feedback amplifier was inspired by the needs to make phone calls over long distances. Intellectual giants like Bode, Nyquist, and Shannon developed theoretical foundations of control and communications.

Today the global telecommunication system is said to be the world's largest man-made artifact, with the total number of mobile subscriptions in the beginning of 2016 at around 7.4 billion. In many countries the number of mobile subscriptions exceeds the population. The *Internet of Things (IoT)* is of particular interest because it enables simple ways of using feedback, and combined with the cloud it offers many interesting opportunities for novel control applications. It is expected that the number of IoT devices will surpass mobile subscriptions by 2018. Telecommunications is a high pace industry where the consumer preferences change quickly, making it hard to predict what products will be like 2–3 years from now and the rules of the game change continuously.

The development of camera modules, GPS modules, accelerometers, and gyros for the mobile phone industry has decreased the cost for such sensors by several orders of magnitude because of the large volumes involved. Reduction in size and improvements in power efficiency have been required to fit sensors into hand held devices with reasonable battery life times. The inexpensive components have then found several uses in other fields, for example virtual reality.

Cost efficiency is vital for production in large volumes, making it economical to put large engineering efforts into cost optimization even for minute details. The requirements from the communication field lead the development of smaller and more energy efficient solutions.

There is extensive standardization in the telecommunication markets, forced by the fact that the technologies share the common radio frequency spectrum and a carefully controlled use of this limited resource has been needed. It is also highly beneficial for the consumer if devices from different manufacturers can function together. The development of technology is coordinated in different groups. The 3GPP consortium develops the standards for mobile communication using GSM (2G), WCDMA (3G), LTE (4G) as well as future communication standards such as 5G and beyond. Similarly, IEEE working groups develop standards, for example 802.11 for wireless routers. Competing operators, vendors, and mobile equipment manufacturers meet in standardization meetings to discuss and decide on new functionality and performance specifications for future devices. New inventions and intellectual property rights are very important for competitive reasons which leads to both portfolio agreements and patent battles.

Control enters on many levels, from analog electronics where it is used to improve performance such as linearity and power efficiency of power amplifiers, to higher functional levels where control is used to continuously choose suitable system parameters such as transmission power and coding schemes depending on existing communication conditions. PID control is often used together with and gain scheduling and simple adaptive schemes.

15.8 Further Reading

A comprehensive treatment of architecture and design of complex systems is given in [CCS15]. Much of the development of PID controllers occurred in the power and process industries, where also many of the associated bottom-up control architectures like cascade, selector, and midranging control appeared. A detailed treatment is given in [ÅH06]. Complementary filtering is well described in []. The internal model control architecture is described in [BT78]. The internal model principle which says that a good controller should contain a model of the process is formulated in [FW76]. The Smith predictor [Smi57a, Hun07] is closely related to internal model control because it also uses a parallel model. It was invented by Otto J. M. Smith, a legendary professor from University of California, Berkeley who also invented posicast control [Smi57b], a scheme for controlling highly oscillatory systems.

Decoupling has long been used in many areas of control, the architecture with inverted decoupling is introduced in [Wad97], a state space version is given in [HK83]. The relative gain array, a simple way to quickly estimate the interaction, was introduced in [Bri66]. A detailed treatment is found in [McA83].

Gain scheduling is widely used in practice, overviews are presented in [LWE00, SA92, Rug91]. Adaptive control emerged from the desire to avoid gain scheduling in flight control and the dream of having a controller that can automatically adjust itself to good performance in the process industry. Early pioneering work is found in [Gre59]. Current knowledge is found in [ÅW08a, CS08, LW13]. System identification is a key element of adaptive control; the book [Lju99a] is an excellent reference. In spite of advances in adaptive control, gain scheduling is still the dominant control scheme for flight control [Ste80], with one reason being the significant developments in air-data sensors. The process engineers dream of having a simple universal controller that self-adjusts to provide good robust performance; the relay auto-tuner for PID control is a partial answer.

Rosenblatt's perceptron [Ros62] was the first neural network, which was used to separate hyperplanes in pictures. An analog version, the Adaline [WS85], invented by Widrow, was used in simple adaptive systems and for noise cancellation. A severe and partially unfair criticism was given by Minsky and Papert [MP69] who did not realize the advantage with many layers. There was a revival when backpropagation was introduced to find parameters in networks with many layers [RHW86]. Backpropagation is closely related to dynamic programming [Bel57]. The disadvantage of networks with few layers was clarified in [Hås87]. An early application to picture classification is given in [LBD$^+$89], using deep structures and convolutional networks. There have lately been significant advances in object recognition, driven by improved algorithms and good test bases. The ImageNet Large Scale Visual Recognition Challenge is a benchmark with millions of images and hundreds of objects. Competitions have been run since 2010. The classification error was brought down from around 30% in 2010 to 3% in 2016 [KSH12, SZ14, SHM$^+$16a]. The usefulness of deep convolutional networks has been proven in many domains: character recognition, computer vision, and game-playing. AlphaGo's algorithm uses a Monte Carlo tree search to find its moves based on knowledge previously obtained by machine learning based on extensive training, both from playing with humans and computers [SHM$^+$16b].