Optimization-Based Control

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

With contributions from:
Karl Åström, William Dunbar, Ryan Franz, John Hauser,
Ali Jadbabaie, Mark Milam, Nicolas Petit, and ChatGPT

Version v2.3h (12 March 2023)

© California Institute of Technology
All rights reserved.

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

Preface

These notes serve as a supplement to *Feedback Systems* by Åström and Murray [ÅM21] and expand on some of the topics introduced there. They are motivated by the increasing role of online optimization in feedback systems. This is a change from the traditional use of optimization in control theory for *offline* design of control laws and state estimators. Fueled by Moore's law and improvements in real-time algorithms, it is now possible to perform estimation and control design algorithms online, allowing the system to better account for nonlinearities and to adapt to changes in the underlying dynamics of the controlled process. This changes the way that we think about estimation and control since it allows much greater flexibility in the design process and more modularity and flexibility in the overall system.

Our goal in this supplement is to introduce the essential formalisms and tools required to design optimization-based controllers. Key topics include real-time trajectory generation using differential flatness, the maximum principle, receding horizon optimal control, stochastic processes, Kalman filtering, moving horizon estimation, and (distributed) sensor fusion. While these topics might normally constitute separate textbooks, in this set of notes we attempt to present them in a compact way that allows them to be used in engineering design. We also briefly survey additional advanced topics through the text, with pointers to further information for interested readers.

The various mathematical techniques that are described have been implemented in the Python Control Systems Library (python-control), which is available online at `http://python-control.org`. Most chapters have a section describing the relevant python-control module that implements the topics of the chapter, and exercises are included to allow students to explore various concepts using python-control to carry out numerical calculations. Files associated with these exercises can be downloaded from the relevant portion of the *Feedback Systems* website:

`https://fbswiki.org/OBC`

This supplement has been used in a second quarter controls course at Caltech, taken by a mixture of advanced undergraduates and beginning graduate students with interest in a variety of application areas. The first half of the 10 week course focuses on trajectory generation and optimal control, ending with receding horizon control. In the second half of the course, we introduce stochastic processes and derive the Kalman filter and its various extensions, including the information filter and sensor fusion. The prerequisites for the course are based on the material covered in *Feedback Systems*, including basic knowledge in Lyapunov stability theory and observers. If needed, these topics can be inserted at the appropriate point in covering the material in this supplement, though they can generally be omitted for a class focused primarily on applications of the concepts.

The notation and conventions in the book follow those used in the main text. Because the notes may not be used in their entirety or in the order presented here, we have attempted to write each chapter as a standalone extension of topics that are briefly introduced in *Feedback Systems*. To this end, each chapter starts with a short description of the prerequisites for the chapter and citations to the relevant literature. Advanced sections, marked by the “dangerous bend” symbol shown in the margin, contain material that requires a slightly more technical background, of the sort that would be expected of graduate students in engineering. Additional information is available on the *Feedback Systems* website:

`https://fbswiki.org`

Contents

1 Introduction **1-1**
- 1.1 System and Control Design 1-1
- 1.2 The Control System "Standard Model" 1-5
- 1.3 Layered Control Systems 1-7
- 1.4 The Python Control Systems Library 1-10

2 Trajectory Generation and Tracking **2-1**
- 2.1 Two Degree of Freedom Design 2-1
- 2.2 Trajectory Tracking and Gain Scheduling 2-3
- 2.3 Trajectory Generation and Differential Flatness 2-7
- 2.4 Implementation in Python 2-16
- 2.5 Other Methods for Generating Trajectories 2-18
- 2.6 Further Reading 2-20

3 Optimal Control **3-1**
- 3.1 Review: Optimization 3-1
- 3.2 Optimal Control of Systems 3-4
- 3.3 Examples 3-8
- 3.4 Implementation in Python 3-10
- 3.5 Linear Quadratic Regulators 3-11
- 3.6 Choosing LQR weights 3-15
- 3.7 Advanced Topics 3-16
- 3.8 Further Reading 3-20

4 Receding Horizon Control **4-1**
- 4.1 Overview 4-1
- 4.2 Receding Horizon Control with Terminal Cost 4-4
- 4.3 Implementation in Python 4-13
- 4.4 Receding Horizon Control Using Differential Flatness 4-15
- 4.5 Choosing Cost Functions 4-16
- 4.6 Implementation on the Caltech Ducted Fan 4-20
- 4.7 Further Reading 4-27

5 Stochastic Systems **5-1**
- 5.1 Brief Review of Random Variables 5-1
- 5.2 Introduction to Random Processes 5-8

5.3 Continuous-Time, Vector-Valued Random Processes 5-11
5.4 Linear Stochastic Systems with Gaussian Noise 5-14
5.5 Random Processes in the Frequency Domain 5-17
5.6 Implementation in Python 5-19
5.7 Further Reading 5-21

6 Kalman Filtering **6-1**
6.1 Linear Quadratic Estimators 6-1
6.2 Extensions of the Kalman Filter 6-4
6.3 LQG Control 6-6
6.4 Implementation in Python 6-7
6.5 Application to a Thrust Vectored Aircraft 6-8
6.6 Further Reading 6-9

7 Sensor Fusion **7-1**
7.1 Discrete-Time Stochastic Systems 7-1
7.2 Kalman Filters in Discrete Time (FBS2e) 7-3
7.3 Predictor-Corrector Form 7-4
7.4 Sensor Fusion 7-6
7.5 Implementation in Python 7-7
7.6 Additional Topics 7-8

Bibliography **B-1**

Index **I-1**

Chapter 1

Introduction

This chapter provides an introduction to the optimization-based framework that is used in throughout this supplement and also introduces the Python Control Systems Library (python-control), which implements all the functionality required for material presented in this supplement.

Prerequisites. Readers should be familiar with standard concepts in control theory, including input/output modeling, feedback interconnections, and the role of feedback in allowing the design of (closed loop) dynamics and providing robustness to uncertainty. We utilize state sapce representations using ordinary differential equations, though detailed knowledge of differential equations is not required.

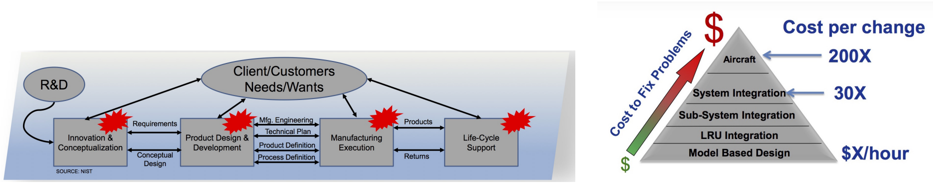
1.1 System and Control Design[1]

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 can be optimized. Sometimes it is an enabler, as was illustrated by the Wright Flyer, which was discussed in FBS2e 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.

---

[1] The material in this section is drawn from FBS2e, Chapter 15 (online version).



(a) Design Flow (b) Costs

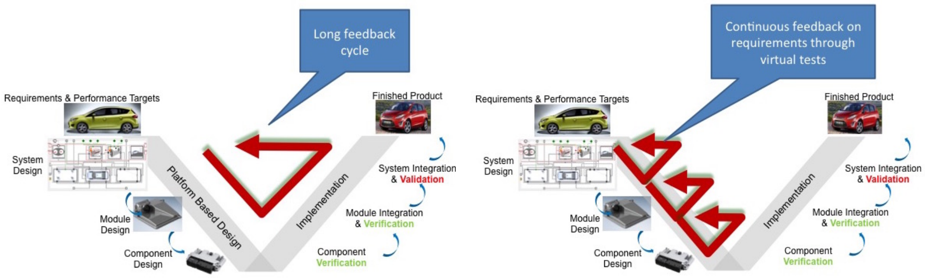
Figure 1.1: Engineering design process. A typical design cycle is shown in (a) and (b) illustrates the costs of correcting faults or making design changes at different stages in the design process.

In design of modern control systems, the engineering workflow is broken down into phases to manage the complexity of overall system. Early phases of the design create a basic architecture for the system, with interaction between subsystems that provide the main functionality of the system. For an aircraft, those subsystems might be the airframe (fuselage and wings), propulsion system (engines), the hydraulic system, the electric power system, the flight control system, and other critical components. For a cell phone, the subsystems might be the chassis, the display (including touch interface), the communications subsystem (5G, wifi), the audio system (speakers and microphone), the power system (batteries and charging), among others. In each case, a high level architecture is required that describes what the subsystems are responsible for and how they will interact. Each subsystem is itself comprised of a variety of components, which also have their own specifications and interfaces. The engineering workflow typically operates by carrying out a succession of refinements of the design from one level of abstraction down to the next, and then assembly the components of the design from the components to the subsystems to the overall system, with validation and testing at each stage.

Figure 1.1a shows a typical design process for a modern engineering system. Design is broken down into phases such as research and development (R&D), conceptualization, development, manufacturing, and life-cycle support. One of the features of engineering complex systems is that it can be very costly to make corrections late in the product development cycle, since a substantial amount of engineering effort has already been carried out and may need to be redone. These costs are illustrated in Figure 1.1b. 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 *design V*, shown in Figure 1.2a, dates back to NASA's Apollo program [SC92] and is a common design pattern for both hardware and software. The left leg of the V illustrates the design process starting with requirements and ending with system, module, and component designs. 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


Long feedback cycle
Requirements & Performance Targets
Finished Product
System Design
Platform Based Design
Implementation
System Integration & Validation
Module Design
Module Integration & Verification
Component Design
Component Verification
Continuous feedback on requirements through virtual tests
Requirements & Performance Targets
Finished Product
System Design
Implementation
System Integration & Validation
Module Design
Module Integration & Verification
Component Design
Component Verification

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

Figure 1.2: Design methodologies for complex systems. (a) The 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. (b) A model-based design process where virtual validation is be made at many stages in the design process, shortening the feedback for validation.

the development process or even worse when systems are in operation, as illustrated in Figure 1.1b. 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 Figure 1.2b. When hardware and subsystems are built they can replace the corresponding models using hardware-in-the-loop simulations.

To perform verification efficiently it is necessary that requirements are expressed mathematically and checked automatically using models of the system and its environment, along with a variety of tools for analysis. *Regression analysis* can be used to ensure 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 many factors: 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 mar-

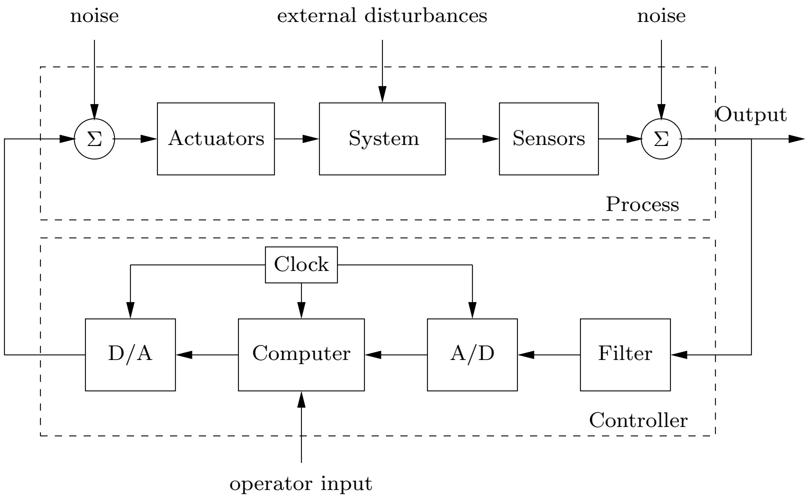


Figure 1.3: Schematic diagram of a control system with sensors, actuators, communications, computer, and interfaces.

ket, where requirements change and close interaction with customers is required. The method is characterized by the *Agile Manifesto* [BBvB$^+$01], which values individuals and interactions over processes and tools; 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.

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 1.3. 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 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 FBS2e 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 FBS2e, 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.

1.2 The Control System "Standard Model"

Feedback control appears across an enormous variety of applications and in various forms. Despite the wide range of applications and implementations, there is a common design pattern for most modern feedback control systems, as illustrated in Figure 1.4. The starting point for the control system is the process that we wish to control, which we model as an input/output system. The inputs to the process consist of both those inputs that the controller specifies as well as inputs that may come from external disturbances and uncertainties in the model. The outputs from the process, possibly corrupted by noise, are processed by an "observer", whose function is to estimate the state of the underlying process from the measured (and sometimes noisy) signals. Finally, the controller is responsible for taking some level of description of the task to be accomplished and generating those inputs that will cause the process to carry out the desired operation.

While this general diagram is likely familiar to anyone with experience in classical control theory (e.g., PID control of a linear dynamical system represented by its transfer function), it is perhaps useful to point out that this basic pattern is present in many systems that use different representations of the dynamics and uncertainty. For example, the process that we are controlling may be an infrastructure management system in which there are requests for resources that must be managed and balanced. Uber is one example of such a "control system", with the process consisting of the dynamics of individually driven Uber vehicles that can be dispatched based on observations of riders requesting transport. The models used

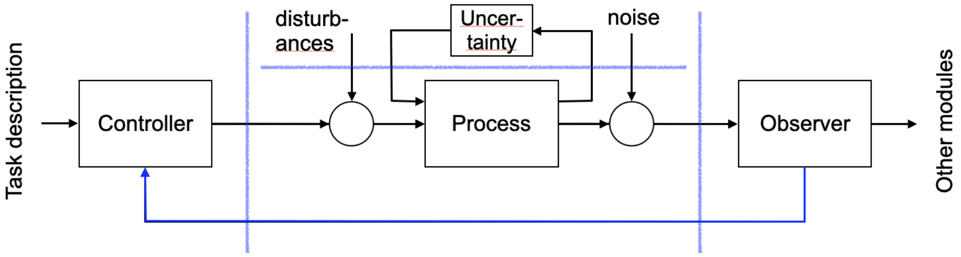


Figure 1.4: Control system standard model.

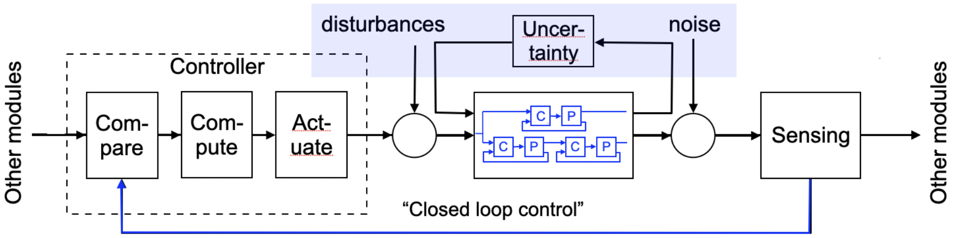


Figure 1.5: Nested control systems.

for this type of process are likely to be based on stochastic queuing system models, but the basic pattern is still there. Other examples of control systems that match this pattern range from aircraft, to the supply chain, to your cell phone.

Another common feature of control systems is that the process itself may be a control system, so that we have a nested set of controllers, as illustrated in Figure 1.5. Note that in this view the inputs and outputs of the overall system are themselves coming from other modules. We have also expanded the view of the controller to include its three key functions: comparing the current and desired state of the process, computing the possible actions that can be taken to bring these closer, and then "actuating" the process being controlled via some appropriate command. This type of nested system could emerge, for example, if Uber vehicles were autonomous vehicles, where there is a control system in place for each car (which is itself a nested set of control systems, as we shall discuss in the next section).

Based on these observations, we define key elements of the "standard model" of a control system as follows:

Process. The process represents that system that we wish to control. The inputs to the system include controller and environmental inputs, and the outputs to the system are the measurable variables.

Task Description. The task description is an input to the controller that describes the "task" to be performed. Depending on the type of system that is being controlled, the task description could be anything from a simple signal that should be tracked to a description of a complex task with cost functions and constraints.

Observer. The observer takes the outputs of the process and performs calculations to estimate the underlying state of the process and/or the environment. In some cases the observer may also make predictions about the future state of the system or the environment.

Controller. The controller is responsible for determining what inputs should be applied to the system in order to carry out the desired task. It takes as inputs the description of the task as well as the output of the observer (often the estimated state and/or the state of the environment).

Disturbances. Disturbances represent exogenous inputs to the process dynamics that are not dependent on the dynamics of system or the controller. In Figure 1.4 the disturbances are modeled as being added to the inputs, but more general disturbances are also possible.

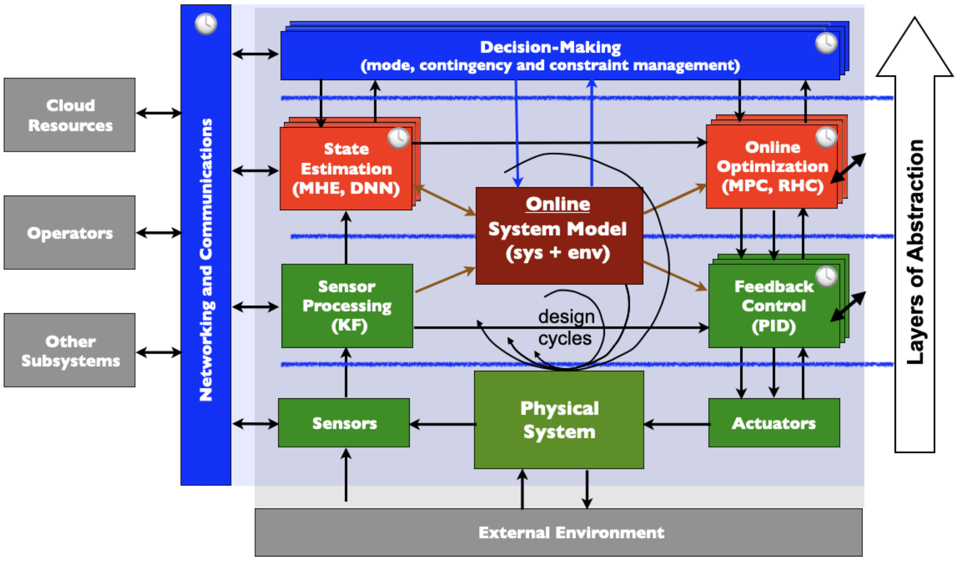


Figure 1.6: Layered control systems. The system consists of four layers: the physical layer (lowest), the feedback regulation layer (green), the trajectory generation layer (red), and the decision-making layer (blue). Networking and communications allows information to be transferred between the layers as well as with outside resources (left). The entire system interacts with the external environment, represented at the bottom of the figure.

Noise. Noise represents exogenous inputs to the observer that corrupt the measurements of the system outputs. In Figure 1.4 the noise is modeled as being added to the inputs, but more general noise signals are also possible.

Uncertainty. This block represents uncertainty in the dynamics of the process or "reactive" uncertainty in the environment in which the process operates. We represent the environment as a feedback interconnection with the process to reflect the fact that the unmodeled dynamics and or environmental dynamics may depend on the state of the system.

Other Modules. Control systems are often connected with other modules of the overall system, in either a distributed, nested, or layered fashion. The type of interconnection can be through the controllers of the other modules, through physical interconnections, or both.

1.3 Layered Control Systems

A related view of a modern control system is as a "layered" control system in which we reason about the system at different layers of abstraction, as shown in Figure 1.6. To a large extent this is just a different view of the "nested" representation of a control system in Figure 1.5, but here we are more explicit about the different representations of the system. In this figure, the control system is described by four layers of abstraction, separated by horizontal lines.

The lowest layer is the physical layer, representing the physical process being controlled as well as the sensors and actuators. This layer is often described in terms of input/output dynamics that model how the system evolves over time. The simplest (and one of the most common) representations is an ordinary differential equation model of the form

$$\frac{dx}{dt} = f(x, u, d), \qquad y = h(x, n),$$

where $x \in \mathbb{R}^n$ represents the state of the system, $u \in \mathbb{R}^m$ represents the inputs that can be commanded by the controller, $d \in \mathcal{D}$ represents disturbance signals that come from the external environment, $y \in \mathbb{R}^p$ represents the measured outputs o the system, and $n \in \mathcal{N}$ represents process or sensor noise. The design of the physical system will normally attempt to make sure that the region of the state space in which the system is able to operate (called the *operating envelope*) satisfies the needs of the user or customer. For an aircraft, for example, this might consist of specifications on the altitude, speed, and maneuverability of the physical system.

The next layer is the *feedback regulation* layer (sometimes also called the "inner loop") in which we use feedback control to track a reference trajectory. This layer commonly represents the abstractions used in classical control theory, where we have a reference input r that we wish to track while at the same time attenuating disturbances d and avoiding amplification of process or sensor noise n. The system and controller at this level might be represented by transfer functions $P(s)$ and $C(s)$ and our specification might be on various input/output transfer functions such as the Gang of Four (see FBS2e, Section 12.1):

$S = \dfrac{1}{1 + PC}$	sensitivity function	$PS = \dfrac{P}{1 + PC}$	load (or input) sensitivity function
$T = \dfrac{PC}{1 + PC}$	complementary sensitivity function	$CS = \dfrac{C}{1 + PC}$	noise (or output) sensitivity function

A typical specification for design at this layer of abstraction might be a weighted sensitivity function, such as

$$\| |W_1 S| + |W_2 T| \|_\infty < 1.$$

The feedback regulation phase of design will also often compensate for the effects of unmodeled dynamics, traditionally done by the specification of gain, phase, and stability margins.

This layer also carries out some level of sensor processing to try to minimize the effects of noise. In classical control design the sensor processing is often integrated into the controller design process (for example by imposing some amount of high frequency rolloff), but many modern control systems will use Kalman filtering to process signals and also perform sensor fusion. Kalman filtering is described in more detail in Chapter 6.

Continuing up our abstraction hierarchy, the next layer of abstraction is the *trajectory generation* layer (sometimes also called the "outer loop"). In this layer we attempt to find trajectories for the system that satisfy a commanded task, such as moving the system from one operating point to another while satisfying constraints on the inputs and states. At this layer, we assume that the effects of noise, disturbances, and unmodeled dynamics have been taken care of at lower levels but nonlinearities and constraints are explicitly accounted for. Thus we might use a model of the form

$$\frac{dx}{dt} = f(x, u), \qquad g(x, u) \leq 0,$$

where $g \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^k$ is a nonlinear function describing constraints on inputs and states. Our control objective might be to optimize according to a cost function of the form

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

where $L(x, u)$ represents the integrated cost along the trajectory and $V(x)$ represents the terminal cost (e.g., it should be small near the final operating point that we seek to reach). We will study this problem and its variants in Chapters 2 and 3.

As in the case of the feedback regulation layer, the trajectory generation layer also has a "observer" function, labeled as "state estimation" in Figure 1.6. The details of this observer depend on the application, but could represent additional sensor processing that is required for trajectory generation or sensing of the environment for the purpose of satisfying specifications relative to that environment. The latter case is particularly common in applications such as autonomous vehicles, where the state estimation often includes perception and prediction tasks that are used to identify other agents in the environment, their type (e.g., pedestrian, bicycle, car, truck), and their predicted trajectory. This information will be used by the trajectory generation algorithm to avoid collisions or to maintain the proper position relative to those other agents. The applications of Kalman filtering and sensor fusion to problems at this layer are considered in Chapters 6 and 7.

The highest layer of abstraction in Figure 1.6 is the *decision-making* layer (sometimes also called the "supervisory" layer). At this layer we often reason over discrete events and logical relations. For example, we may 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). This layer can also be used to reason over discrete (as opposed to continuous) decisions that we must make (stop, go, turn left, turn right). This final layer is not explicitly part of the material covered in this book; a brief discussion of the design problem at this layer can be found in FBS2e, Section 15.3.

In a full system design, the three control layers that we depict here may in fact include additional layers within them, or be divided up slightly differently. Similarly, the physical layer may consist of system that themselves have internal control loops running, potentially at multiple layers of abstraction. And the system may be networked to other agents and information systems that provide information and constraints on system operation. Thus, our system is a combination of nested, layered, distributed systems, all operating together.

Another important element of modern control systems is their distributed and interconnected nature. Much of this is already presented in the layered control structure described above, but there can also be "external" interactions. On the left side of Figure 1.6 are a set of blocks that represent some of the elements that can connected through networked information channels. These can include cloud resources (such as computing or databases), operators (humans or automated), and interactions with other systems and subsystems. The increased capability and capacity of networking and communications is one of the drivers of complexity in modern control systems and has created both new opportunities and new challenges.

Finally, we note the effect of the environment, represented in Figure 1.6 as a block at the bottom of the diagram. This block represents many things, including noise, disturbances, unmodeled dynamics of the process, and the dynamics of other systems with which our system is interacting. It is the uncertainty represented in this catchall block that is driving the need for feedback control, and the impact of these different types of uncertainty appears in each level of our controller design.

1.4 The Python Control Systems Library[2]

The Python Control Systems Library (python-control) is a Python package that implements basic operations for analysis and design of feedback control systems. The package was created in 2009, shortly after the publication of the first edition of *Feedback Systems*. The initial goal of the project was to implement the operations needed to carry out all the examples in the book. A primary motivation for the creation of the python-control library was the need for open-source control design software built on the Python general-purpose programming language. The "scientific stack" of NumPy, SciPy, and Matplotlib provide fast and efficient array operations, linear algebra and other numerical functions, and plotting capabilities to Python users. Python-control has benefited from this foundation, using, e.g., optimization routines from SciPy in its optimal control methods, and Matplotlib for Bode diagrams.

The python-control package provides the functionality required to implement all of the techniques described in this supplement. This section provides a brief overview of the python-control package, with the intent of indicating the calling structure of the code and including a few simple examples. More detailed examples are given in subsequent chapters, and more detailed documentation is available at `http://python-control.org`.

Package Structure and Basic Functionality

The python-control package implements an inheritance hierarchy of dynamical system objects. For the most part, when two systems are combined in some way through a mathematical operation, one will be promoted to the type that is the highest of the two. Arranged in order from most to least general, they are:

- `InputOutputSystem`: Input/output system that may be nonlinear and time-varying
    - `InterconnectedSystem`: Interconnected I/O system consisting of multiple subsystems
    - `NonlinearIOSystem`: Nonlinear I/O system

---

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

  - `LinearICSystem`: Linear interconnected I/O systems
  - `LinearIOSystem`: Linear I/O system

- `LTI`: Linear, time-invariant system
  - `FrequencyResponseData`: Frequency response data systems
  - `StateSpace`: State space systems
  - `TransferFunction`: Transfer functions

Each system type can be either discrete-time, that is, $x(k+1) = f(x(k), u(k))$; $y(k) = g(x(k), u(k))$ or continuous time, that is, $\dot{x} = f(x, u)$; $y = g(x, u)$. A discrete-time system is created by specifying a nonzero 'timebase' dt when the system is constructed:

- $dt = 0$: continuous time system (default)
- $dt > 0$: discrete time system with sampling period dt
- $dt =$ `True`: discrete time with unspecified sampling period
- $dt =$ `None`: no timebase specified

Linear, time-invariant systems can be interconnected using mathematical operations `+`, `-`, `*`, and `/`, as well as the domain-specific functions `feedback`, `parallel` (`+`), and `series` (`*`). Some important functions for `LTI` systems and their descriptions are given in Table 1.1. Other categories of tools that are available include model simplification and reduction tools, matrix computations (Lyapunov and Riccati equations), and a variety of system creation, interconnection and conversion tools. A MATLAB compatibility layer is provided that has functions and calling conventions that are equivalent to their MATLAB counterparts, e.g. `tf, ss, step, impulse, bode, margin, nyquist` and so on. A complete list is available at `http://python-control.org`.

Table 1.1: Sample functions available in the python-control package.

Frequency domain analysis:

`sys(s)`	Evaluate frequency response of an LTI system at complex frequenc(ies) s
`sys.frequency_response()`	Evaluate frequency response of an LTI system at real angular frequenc(ies) omega
`stability_margins()`	Calculate stability margins and associated crossover frequencies
`bode_plot()`	Bode plot for a system
`nyquist_plot()`	Nyquist plot for a system
`gangof4_plot()`	Plot the "Gang of 4" transfer functions for a system
`nichols_plot()`	Nichols plot for a system

Time domain analysis:

`forced_response()`	Simulated response of a linear system to a general input
`impulse_response()`	Compute the impulse response for a linear system
`initial_response()`	Initial condition response of a linear system
`step_response()`	Compute the step response for a linear system
`step_info()`	Compute step response characteristics
`phase_plot()`	Phase plot for 2D dynamical systems

Other analysis functions and methods:

`sys.dcgain()`	Return the zero-frequency (or DC) gain of an LTI system
`sys.pole()`	Compute poles of an LTI system
`sys.zero()`	Compute zeros of an LTI system
`sys.damp()`	Compute natural frequency and damping ratio of LTI system poles
`pzmap()`	Plot a pole/zero map for a linear system
`root_locus()`	Root locus plot
`sisotool()`	Sisotool style collection of plots inspired by MATLAB

Synthesis tools:

`acker()`	Pole placement using the Ackermann method
`h2syn()`	H_2 control synthesis for plant P
`hinfsyn()`	H_∞ control synthesis for plant P
`lqr()`	Linear quadratic regulator design
`lqe()`	Linear quadratic estimator design (Kalman filter) for continuous-time systems
`mixsyn()`	Mixed-sensitivity H-infinity synthesis
`place()`	Place closed-loop poles

Linear Systems Example

To illustrate the use of the package, we present an example of the design of an inner/outer loop control architecture for the planar vertical takeoff and landing (PVTOL) example in FBS2e, Example 12.9. A slightly different version of this example is available in the python-control GitHub repository.

We begin by initializing the Python environment with the packages that we will use in the example:

```
# pvtol-nested.py - inner/outer design for vectored thrust aircraft
# RMM, 5 Sep 2009 (updated 11 May 2021)
#
# This file works through a control design and
# analysis for the planar vertical takeoff and
# landing (PVTOL) aircraft in Astrom and Murray.

import control as ct
import matplotlib.pyplot as plt
import numpy as np
```

We next define the system that we plan to control:

```
# System parameters
m = 4             # mass of aircraft
J = 0.0475        # inertia around pitch axis
r = 0.25          # distance to center of force
g = 9.8           # gravitational constant
c = 0.05          # damping factor (estimated)

# Transfer functions for dynamics
Pi = ct.tf([r], [J, 0, 0]) # inner loop (roll)
Po = ct.tf([1], [m, c, 0]) # outer loop (posn)
```

The control design is performed by using a lead compensator to control the inner loop (roll axis):

```
# Inner loop control design
#
# Controller for the pitch dynamics: the goal is
# to have a fast response so that we can use this
# as a simplified process for the lateral dynamics

# Design a simple lead controller for the system
k_i, a_i, b_i = 200, 2, 50
Ci = k_i * ct.tf([1, a_i], [1, b_i])
Li = Pi * Ci
```

We can now analyze the results by plotting the frequency response as well as the Gang of 4:

```
# Loop transfer function Bode plot, with margins
plt.figure(); ct.bode_plot(Li, margins=True)
plt.savefig('pvtol-inner-ltf.pdf')

# Make sure inner loop specification is met
```

```
plt.figure(); ct.gangof4_plot(Pi, Ci)
plt.savefig('pvtol-gangof4.pdf')
```

Figures 1.7a and b show the outputs from these commands.

The outer loop (lateral position) is designed using a second lead compensator, using the roll angle as the input:

```
# Design lateral control system (lead compensator)
a_o, b_o, k_o = 0.3, 10, 2
Co = -k_o * ct.tf([1, a_o], [1, b_o])
Lo = -m * g * Po * Co

# Compute real outer-loop loop transfer function
L = Co * Hi * Po
```

We can analyze the results using Bode plots, Nyquist plots and time domain simulations:

```
# Compute stability margins
gm, pm, wgc, wpc = ct.margin(L)

# Check to make sure that the specification is met
plt.figure(); ct.gangof4_plot(-m * g * Po, Co)

# Nyquist plot for complete design
plt.figure(); ct.nyquist_plot(L)
plt.savefig('pvtol-nyquist.pdf')

# Step response
t, y = ct.step_response(T, np.linspace(0, 20))
plt.figure(); plt.plot(t, y)
plt.savefig('pvtol-step.pdf')
```

Figures 1.7c and d show the outputs from the `nyquist_plot` and `step_response` commands (note that the `step_response` command only computes the response, unlike MATLAB, which also plots the response).

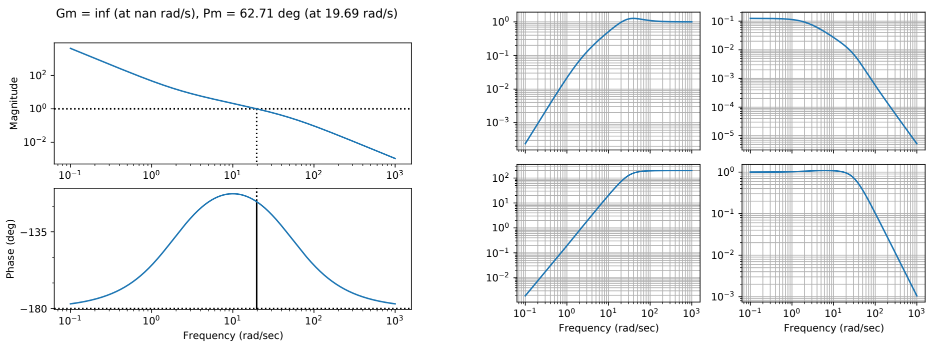
Input/output systems

Python-control supports the notion of an input/output system in a manner that is similar to the MATLAB "S-function" implementation. Input/output systems can be combined using standard block diagram manipulation functions (including overloaded operators), simulated to obtain input/output and initial condition responses, and linearized about an operating point to obtain a new linear system that is both an input/output and an LTI system.

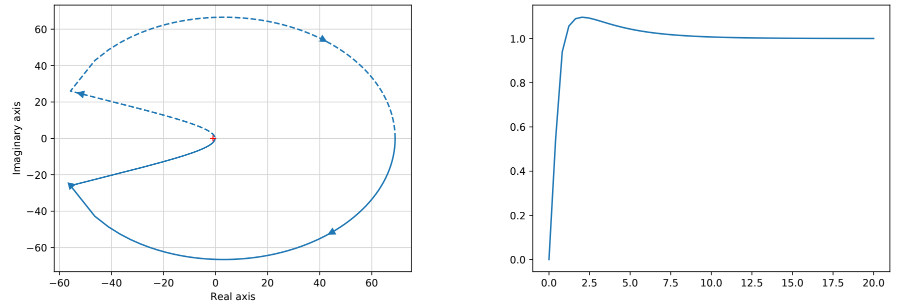
An input/output system is defined as a dynamical system that has a system state as well as inputs and outputs (either inputs or states can be empty). The dynamics of the system can be in continuous or discrete time. To simulate an input/output system, the `input_output_response()` function is used:

```
t, y = input_output_response(io_sys, T, U, X0, params)
```

Here, the variable `T` is an array of times and the variable `U` is the corresponding inputs at those times. The output will be evaluated at those times, though this



(a) Inner loop, with margins (b) Gang of 4 for inner loop



(c) Nyquist plot for full system (d) Step response for full system

Figure 1.7: Sample outputs for PVTOL example.

can be overridden using the `t_eval` keyword, or the NumPy `interp` function can be used to interpolate inputs at a finer timescale, if desired.

An input/output system can be linearized around an equilibrium point to obtain a state space linear system. The `find_eqpt()` function can be used to obtain an equilibrium point and the `linearize()` function to linearize about that equilibrium point:

```
xeq, ueq = find_eqpt(io_sys, X0, U0)
ss_sys = linearize(io_sys, xeq, ueq)
```

The resulting `ss_sys` object is a `LinearIOSystem` object, which is both an I/O system and an LTI system, allowing it to be used for further operations available to either class.

Nonlinear input/output systems can be created using the NonlinearIOSystem class, which requires the definition of an update function (for the right-hand side of the differential or difference equation) and output function (computes the outputs from the state):

```
io_sys = NonlinearIOSystem(
   updfcn, outfcn, inputs=m, outputs=p, states=n)
```

More complex input/output systems can be constructed by making use of the `interconnect()` function, which allows a collection of input/output subsystems to be combined with internal connections between the subsystems and a set of overall system inputs and outputs that link to the subsystems:

```
summation = ct.summing_junction(['u1', '-u2'], 'y', name='summation')

clsys = ct.interconnect(
    [plant, controller, summation], name='system',
    connections=[
        ['summation.u2', 'plant.y'],
        ['controller.e', 'summation.y'],
        ['plant.u', 'controller.u'],
    ],
    inplist=['summation.u1'], inputs='r',
    outlist=['plant.y'], outputs='y')
```

In addition to explicit interconnections, signals can also be interconnected automatically using shared signal names by simply omitting the `connections` parameter.

Interconnected systems can also be created using block diagram manipulations such as the `series()`, `parallel()`, and `feedback()` functions. The `InputOutputSystem` class also supports various algebraic operations such as `*` (series interconnection) and `+` (parallel interconnection).

Exercises

1.1 (Basics of python-control). Consider a second order linear system with dynamics given by the following state space dynamics and transfer function:

$$\frac{d}{dt}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0 & 1\\-k & -b\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}u, \qquad y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}, \qquad P(s) = \frac{1}{s^2 + bs + k}.$$

In this problem you will design a controller for this system in either the time or frequency domain (depending on which you are most comfortable with).

(a) Design either a state space or frequency domain controller for the system that can be used to track a reference signal r (corresponding to a desired state $x_\text{d} = (r, 0)$). Write down the closed loop dynamics for the system and give conditions on the parameters of your controller such that the closed loop system is stable and the steady state error ($e = y - r$) for a step input of magnitude 1 is no more than γ. (The conditions for the gains in your controller should be in terms of inequalities involving the system parameters b and k and performance parameter γ.)

(b) Suppose now that we take $k = 1$ and $b = 0.1$. Pick specific parameters for your controller such that the steady state error is no more than 10% ($\gamma = 0.1$) and the settling time is more more than 5 seconds. Plot the step response for the sytem and compute the rise time, settling time, overshoot, and steady state error for your design in response to a step change in the input r. (You can do the computations either analytically or computationally.)

(c) Using the same parameters for the system and your controller, compute the steady state ratio of the output magnitude to the reference magnitude and the phase offset between the output and the reference for a reference signal $r = \sin(2t)$. (You can do these computations either analyitically or computationally.)

If you carry out the computations for parts 0b and/or 0c numerically, include the MATLAB or Python code used to generate your results, as well as any plots generated by your code and used to determine your answers.

1.2 (Frequency domain analysis using python-control). Consider a control system with

$$P(s) = \frac{b}{(s+a)^2}, \qquad C(s) = \frac{k_\text{p} s + k_\text{i}}{s},$$

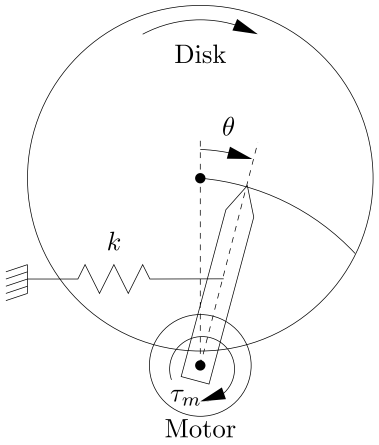
and set $a = b = 1$ and $k_\text{p} = 1$, $k_\text{i} = 0.1$. Using the Python Control Systems Library (python-control), do the following:

(a) Plot the step response of the closed loop system and compute the rise time, settling time, and steady state error.

(b) Plot the frequency response of the open loop system (Bode plot) and compute the gain margin, phase margins, and bandwidth of the system.

(c) Plot the Nyquist plot of the system and compute the stability margin (smallest distance to the -1 point).

1.3 (I/O systems using python-control). Consider a simple mechanism for positioning a mechanical arm and the associated equations of motion:



$$J\ddot{\theta} = -b\dot{\theta} - kr\sin\theta + \tau_\text{m}$$

The system consists of a spring-loaded arm that is driven by a motor. The motor applies a force against the spring and pulls the tip across a rotating platter. The input to the system is the motor torque τ_m. In the diagram above, the force exerted by the spring is a nonlinear function of the head position due to the way it is attached. The output of the system sensors is the offset of the end of the arm from the center of the platter, with a small offset depending on the angular rate:

$$y = l\theta + \epsilon\dot{\theta}$$

Take the system parameters to be

$$k = 1, \quad J = 100, \quad b = 10, \quad r = 1, \quad l = 2, \quad \epsilon = 0.01.$$

Starting with the template Jupyter notebook posted on the FBS2e website, create a Jupyter notebook that documents the following operations:

(a) Compute the linearization of the dynamics about the equilibrium point corresponding to $\theta_\mathrm{e} = 15°$.

(b) Plot the step response of the linearized, open-loop system and compute the rise time and settling time for the output y.

(c) Plot the frequency response of the linearized, open-loop system and compute the bandwidth of the system.

(d) Assuming that the full system state is available, design a state feedback controller for the system that allows the system to track a desired position y_d and sets the closed loop eigenvalues to $\lambda_{1,2} = -10 \pm 10i$. Plot the step response for the closed loop system and compute the rise time, settling time, and steady state error for the output y.

(e) Plot the frequency response (Bode plot) of the closed loop system. Use the frequency response to compute the steady state error for a step input and the bandwidth of the system (frequency at which the magnitude of the output is less than $1/\sqrt{2}$ from its reference value).

Hint: if you are not familiar with frequecy responses of linear time invariant systems, see Section 6.3 (Input/Output Response) of FBS2e.

(f) Design a frequency domain compensator that provides tracking with less than 10% error up to 1 rad/sec and has a phase margin of at least 45°. Demonstrate that your controller meets these requirements by showing Bode, Nyquist, and step response plots, and compute the rise time, settling time, and steady state error for the system using your controller design.

(g) Create simulations of the full nonlinear system with the linear controllers designed in parts 0d and 0f and plot the response of the system from an initial position of 0 m at $t = 0$, to 1 m at $t = 0.5$, to 3 m at $t = 1$, to 2 m at $t = 1.5$.

Chapter 2

Trajectory Generation and Tracking

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

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

2.1 Two Degree of Freedom Design

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

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

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

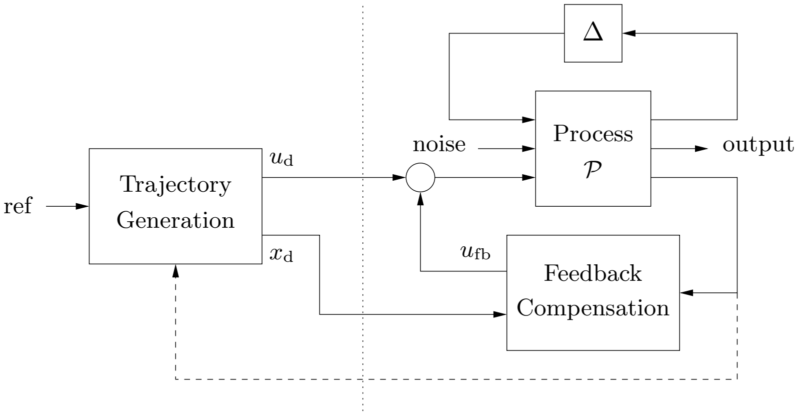


Figure 2.1: Two degree of freedom controller design for a process P with uncertainty Δ. The controller consists of a trajectory generator and feedback controller. The trajectory generation subsystem computes a feedforward command $u_{\rm d}$ along with the desired state $x_{\rm d}$. The state feedback 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. The dashed line represents the use of the current system state for real-time trajectory generation (described in more detail in Chapter 4).

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

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

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

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

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

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

A *feasible trajectory* for the system (2.1) is a pair $(x_{\rm d}(t), u_{\rm d}(t))$ that satisfies the differential equation and generates the desired trajectory:

$$\dot{x}_{\rm d}(t) = f\bigl(x_{\rm d}(t), u_{\rm d}(t)\bigr), \qquad r(t) = h\bigl(x_{\rm d}(t), u_{\rm d}(t)\bigr).$$

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

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

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

2.2 Trajectory Tracking and Gain Scheduling

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

$$\begin{aligned}\dot e &= \dot x - \dot x_\mathrm{d} = f(x,u) - f(x_\mathrm{d}, u_\mathrm{d})\\ &= f(e + x_\mathrm{d}, v + u_\mathrm{d}) - f(x_\mathrm{d}, u_\mathrm{d}) =: F(e, v, x_\mathrm{d}(t), u_\mathrm{d}(t)).\end{aligned}$$

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

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

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

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

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

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

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

$$v = -Ke.$$

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

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

$$v = -K(x_\mathrm{d})e.$$

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

$$u = u_\mathrm{d} - K(x_\mathrm{d})(x - x_\mathrm{d}).$$

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

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

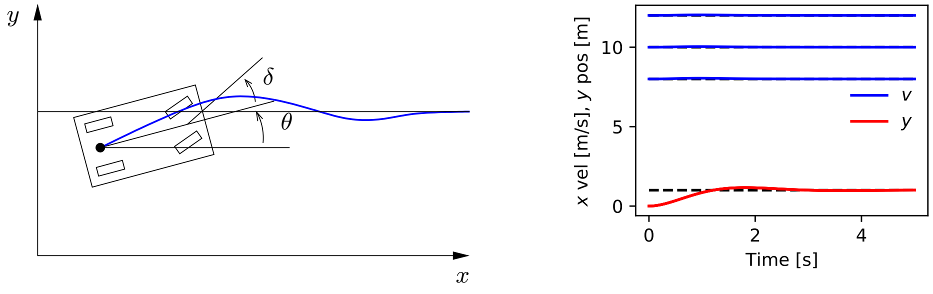
$$u = u_\mathrm{d} - K(x, \mu) \cdot (x - x_\mathrm{d}),$$

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

Example 2.1 Steering control with velocity scheduling

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

$$\dot{x} = \cos\theta\, v, \qquad \dot{y} = \sin\theta\, v, \qquad \dot{\theta} = \frac{v}{l}\tan\delta, \tag{2.2}$$



(a) (b)

Figure 2.2: Vehicle steering using gain scheduling.

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

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

Linearizing the system about the desired trajectory, we obtain

$$\begin{aligned} A_\mathrm{d} &= \left.\frac{\partial f}{\partial x}\right|_{(x_\mathrm{d},u_\mathrm{d})} = \left.\begin{bmatrix} 0 & 0 & -\sin\theta\, v \\ 0 & 0 & \cos\theta\, v \\ 0 & 0 & 0 \end{bmatrix}\right|_{(x_\mathrm{d},u_\mathrm{d})} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v_\mathrm{r} \\ 0 & 0 & 0 \end{bmatrix}, \\ B_\mathrm{d} &= \left.\frac{\partial f}{\partial u}\right|_{(x_\mathrm{d},u_\mathrm{d})} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & v_\mathrm{r}/l \end{bmatrix}. \end{aligned}$$

We form the error dynamics by setting $e = x - x_\mathrm{d}$ and $w = u - u_\mathrm{d}$:

$$\dot e_x = w_1, \qquad \dot e_y = e_\theta, \qquad \dot e_\theta = \frac{v_\mathrm{r}}{l} w_2.$$

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

This can accomplished by setting

$$\begin{aligned} w_1 &= -\lambda_1 e_x \\ w_2 &= -\frac{l}{v_\mathrm{r}}\left(\frac{a_2}{v_\mathrm{r}} e_y + a_1 e_\theta\right). \end{aligned}$$

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

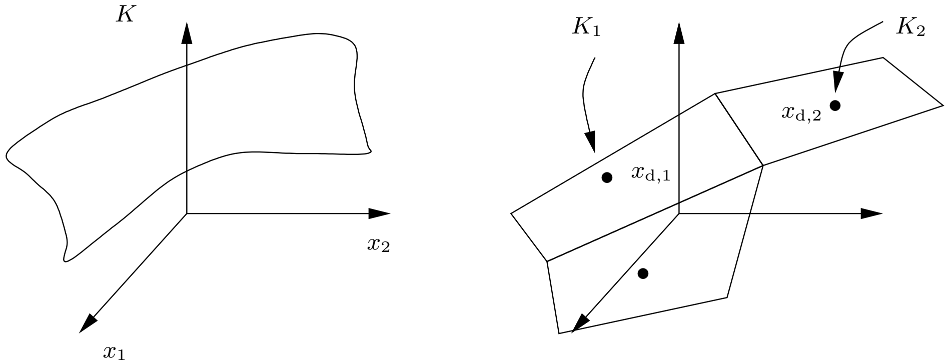


Figure 2.3: Gain scheduling. A general gain scheduling design involves finding a gain K at each desired operating point. This can be thought of as a gain surface, as shown on the left (for the case of a scalar gain). An approximation to this gain can be obtained by computing the gains at a fixed number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

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

$$\begin{bmatrix} v \\ \delta \end{bmatrix} = \underbrace{\begin{bmatrix} v_\mathrm{r} \\ 0 \end{bmatrix}}_{u_\mathrm{d}} - \underbrace{\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \dfrac{a_2 l}{v_\mathrm{r}^2} & \dfrac{a_1 l}{v_\mathrm{r}} \end{bmatrix}}_{K(x_\mathrm{d}, u_\mathrm{d})} \underbrace{\begin{bmatrix} x - v_\mathrm{r} t \\ y - y_\mathrm{r} \\ \theta \end{bmatrix}}_{e}.$$

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

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

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

$$u = u_\mathrm{d} - K(x) e \qquad K(x) = \sum_{j=1}^{N} \rho_j(x) K_j,$$

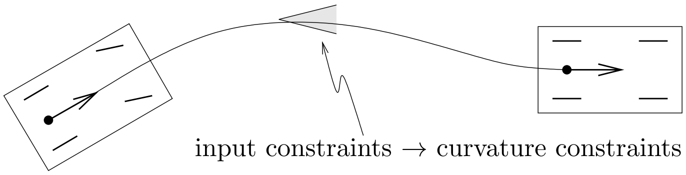


Figure 2.4: Simple model for an automobile. We wish to find a trajectory from an initial state to a final state that satisfies the dynamics of the system and constraints on the curvature (imposed by the limited travel of the front wheels).

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

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

2.3 Trajectory Generation and Differential Flatness

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

$$\dot x_\text{d} = f(x_\text{d}, u_\text{d}), \qquad x_\text{d}(0) = x_0,\, x_\text{d}(T) = x_\text{f}. \tag{2.3}$$

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

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

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

As an example of the type of problem we would like to study, consider the problem of steering a car from an initial condition to a final condition, as shown in Figure 2.4. To solve this problem, we must find a solution to the differential equations (2.2) that satisfies the endpoint conditions. Given the nonlinear nature of the dynamics, it seems unlikely that one could find explicit solutions that satisfy the dynamics except in very special cases (such as driving in a straight line).

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

$$\frac{\dot y}{\dot x} = \frac{\sin\theta}{\cos\theta} \qquad \implies \qquad \theta_\mathrm{d} = \tan^{-1}(\dot y_\mathrm{d}/\dot x_\mathrm{d}).$$

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

$$\dot x = v\cos\theta \qquad \implies \qquad v_\mathrm{d} = \dot x_\mathrm{d}/\cos\theta_\mathrm{d},$$

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

$$\dot\theta = \frac{v}{l}\tan\delta \qquad \implies \qquad \delta_\mathrm{d} = \tan^{-1}(\frac{l\dot\theta_\mathrm{d}}{v_\mathrm{d}}).$$

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

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

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

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

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

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

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

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

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

If the system is differentially flat then

$$\begin{aligned} x(0) &= \beta\big(z(0), \dot z(0), \ldots, z^{(q)}(0)\big) = x_0,\\ x(T) &= \gamma\big(z(T), \dot z(T), \ldots, z^{(q)}(T)\big) = x_\mathrm{f}, \end{aligned} \tag{2.5}$$

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

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

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

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

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

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

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

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

Example 2.2 Nonholonomic integrator

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

$$\dot x_1 = u_1, \qquad \dot x_2 = u_2, \qquad \dot x_3 = x_2 u_1.$$

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

$$x_1 = z_1, \qquad x_2 \quad = \dot x_3 / \dot x_1 = \dot z_2 / \dot z_1, \qquad x_3 = z_2. \tag{2.7}$$

Using simple polynomials as our basis functions,

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

the equations for the feasible (flat) trajectory become

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

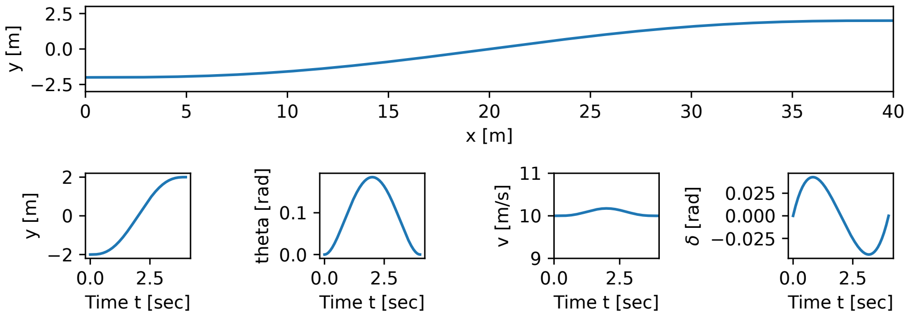
This is a set of 8 linear equations in 8 variables. It can be shown that the matrix M is full rank when $T \neq 0$ and the system can be solved numerically. ∇

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

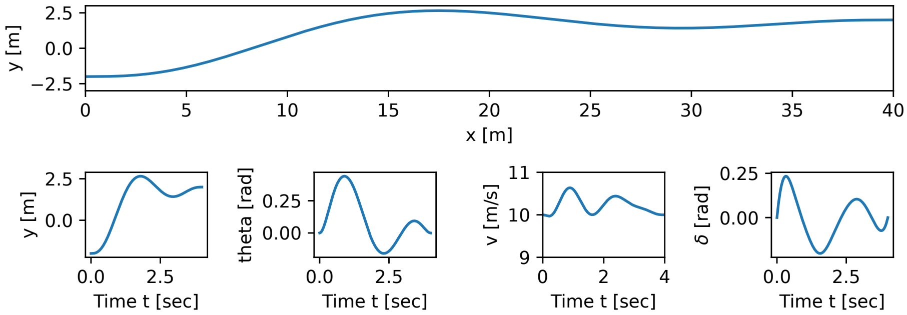
In the example above we had exactly the same number of basis functions as the total number of initial and final conditions, but it is also possible to choose a larger number of basis functions and use the remaining degrees of freedom for other purposes. For example, if there is a performance index for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

Example 2.3 Vehicle steering

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



(a) Least squares solution



(b) Penalize lateral error, input cost

Figure 2.5: Trajectory generation for vehicle steering example.

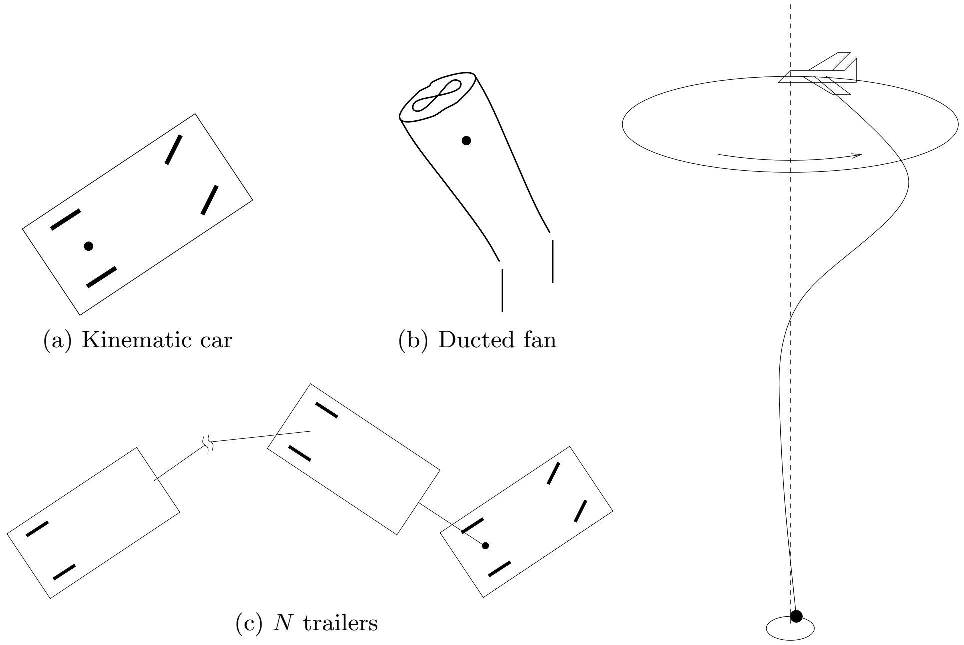
One possible solution for the now underdetermined linear set of equations that we obtain in equation (2.6) is to use the least squares solution of the linear equation $Ma = \bar{z}$, which provides the smallest possible a (coefficient vector) that satisfies the equation. The results of applying this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

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

$$J(x,u) = \int_0^T \big((y(\tau) - y_\text{f})^2 + (v(\tau) - v_\text{f})^2 + 10\delta^2(\tau)\big)\, d\tau,$$

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

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



(a) Kinematic car

(b) Ducted fan

(c) N trailers

(d) Towed cable

Figure 2.6: Examples of flat systems.

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

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

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

Example 2.4 Vectored thrust aircraft

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

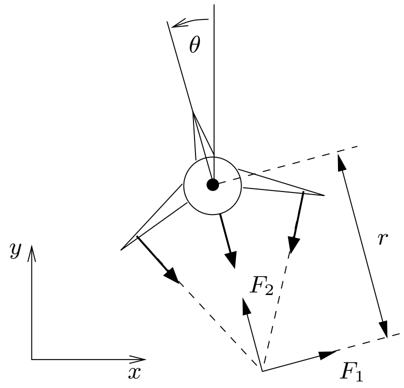


Figure 2.7: Vectored thrust aircraft (from FBS2e). The net thrust on the aircraft can be decomposed into a horizontal force F_1 and a vertical force F_2 acting at a distance r from the center of mass.

The dynamics for the system are

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

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

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

Using the system dynamics, it can be shown that

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

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

Additional remarks on differential flatness

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

Flatness of linear systems and feedback linearizable systems. All single-input reachable linear systems are differentially flat, which can be shown by putting the system into reachable canonical form (FBS2e, equation (7.6)) and choosing the last state as the flat output. Multi-input, reachable linear systems are often differentially flat, though the construction of the flat outputs can be more complicated. Similarly, systems that are feedback linearizable (see FBS2e, Figure 6.15) are commonly differentially flat, as are systems in pure feedback form (2.8) (under appropriate regularity conditions). More details on some of the conditions under which feedback linearizable systems are differentially flat can be found in [vNRM98].

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

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

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

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

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

where $z \in \mathbb{R}^p$, $p \geq m$ represents a set of outputs that parameterize the trajectory and Φ represents a set of remaining differential constraints on the output. The minimum number of possible remaining differential constraints is called the *defect* of the system. Even though such a system is not differentially flat, the problem of trajectory generation may still be considerably simplified since the number of differential equation constraints may be small. In some cases, it even can turn out that even if a system is differentially flat the computations required to find the states based on the flat outputs may be sufficiently complicated that it is beneficial to use a set of outputs that only partially determine the system states.

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

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

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

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

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

The resulting dynamics become

$$M(q)\ddot{q} = M(q)v \qquad \Longrightarrow \qquad \ddot{q} = v,$$

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

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

---

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

2.4 Implementation in Python[2]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

---

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

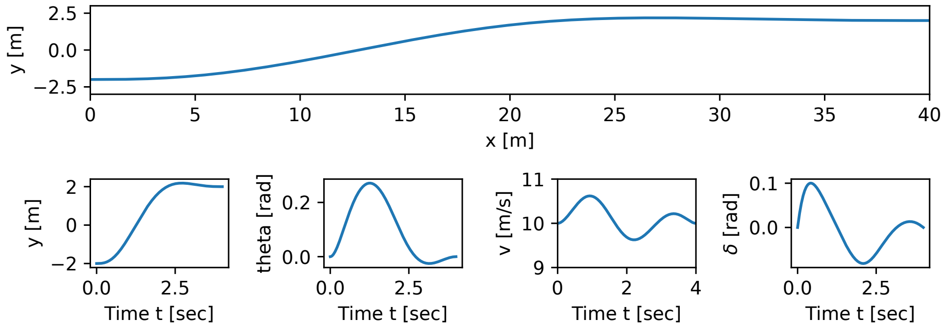


Figure 2.8: Trajectory generation using differential flatness.

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

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

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

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

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

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

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

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

A basic state feedback controller of the form

$$u = u_\mathrm{d} - K(x - x_\mathrm{d})$$

can be created with the python-control command

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

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

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

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

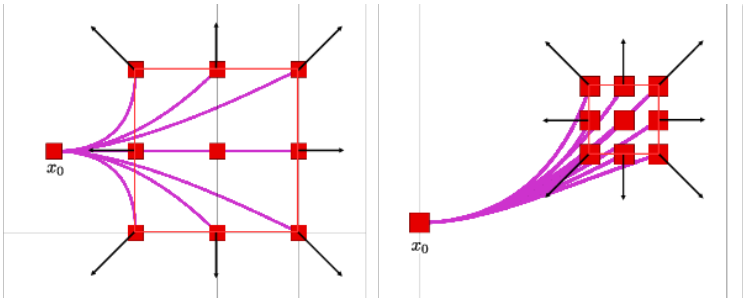


Figure 2.9: Examples of motion primitives for a planar vehicle. Example of 9 planar motion primitives from initial state x_0 for an acceleration-controlled system (left) and a jerk-controlled system (right). The black arrow indicates corresponding control input. The red boundary shows the feasible region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

The list of indices can either be integers indicating the offset into the controller input vector $(x_{\text{d}}, u_{\text{d}}, x)$ or a list of strings matching the names of the input signals. The controller implemented in this case has the form

$$u = u_{\text{d}} - K(\mu)(x - x_{\text{d}})$$

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

2.5 Other Methods for Generating Trajectories

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

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

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

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

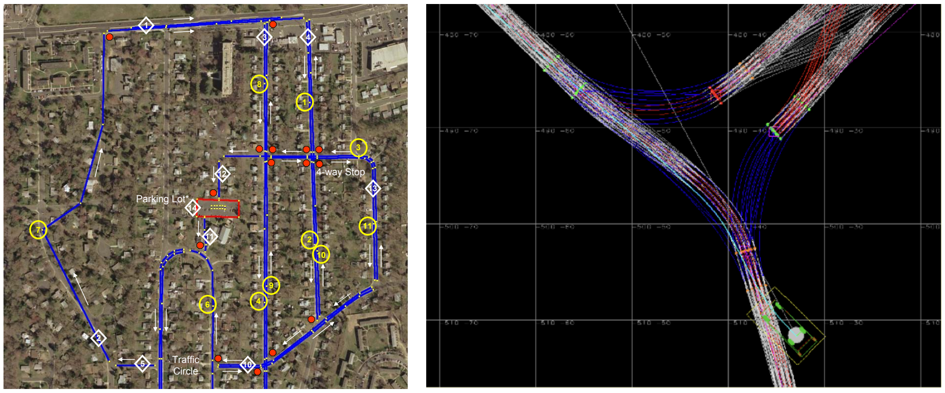


Figure 2.10: Graph-based planning. (a) Road network definition file (RNDF), used for high level route planning. (b) Graph of feasible trajectories at an intersection.

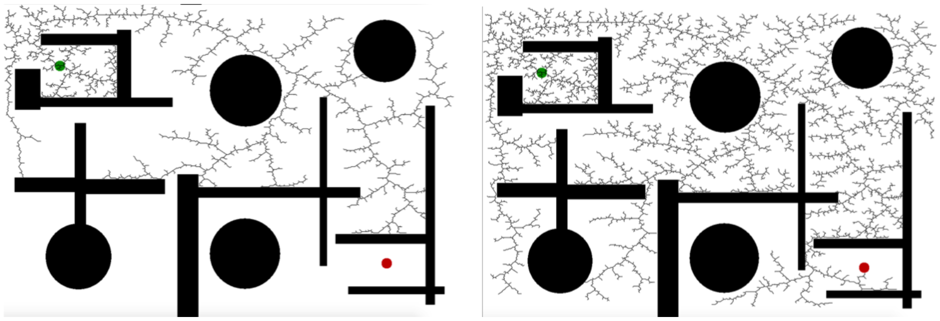


Figure 2.11: Rapidly exploring random tree (RRT) path planning. Random exploration of a 2D search space by randomly sampling points and connecting them to the graph until a feasible path between start and goal is found. Figure and caption courtesy Correll et al. [CHHR22] (CC-BY-ND-NC).

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

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

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

2.6 Further Reading

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

Exercises

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

$$\dot{x} = Ax + Bu, \qquad y = Cx \tag{2.13}$$

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

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

for x_d and u_d.

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

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

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

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

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

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

$$x_\text{d} = \begin{bmatrix} \texttt{square}(2\pi t/20) \\ 0 \end{bmatrix}, \qquad u_\text{d} = 0,$$

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

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

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

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

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

2.3. Consider the nonholonomic integrator in Example 2.2:

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

Using Bezier curves as the basis functions for the flat outputs, find a trajectory between $x_0 = (0, 0, 0)$, $u_0 = (1, 0)$ and $x_\text{f} = (10, 0, 5)$, $u_\text{f} = (1, 1)$ over a time interval $T = 2$. Plot the states and inputs for your trajectory.

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

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

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

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

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

$$\frac{dx}{dt} = \begin{bmatrix} -a_1 & -a_2 & -a_3 & \dots & -a_n \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & & 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u, \qquad (2.14)$$

$$y = \begin{bmatrix} b_1 & b_2 & b_3 & \dots & b_n \end{bmatrix} x + du.$$

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

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

where N is a sufficiently large integer.

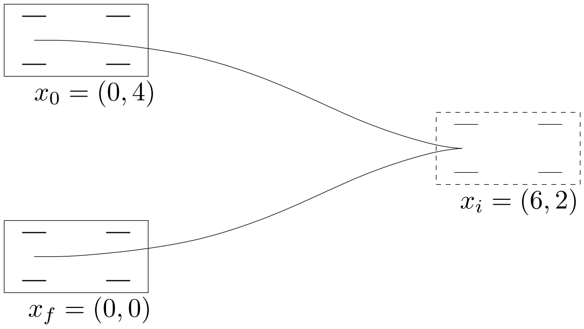
(a) Compute the state space trajectory $x(t)$ and input $u(t)$ corresponding to equation (2.15) and satisfying the differential equation (2.14). Your answer should be an equation similar to equation (2.7) for each state x_i and the input u.

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

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

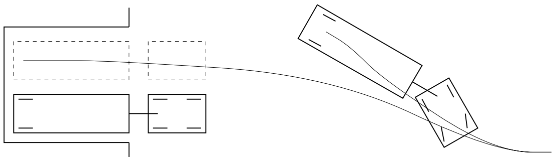
2.6. Show that the servomechanism example from Exercise 1.3 is differentially flat and compute the forward and reverse mappings between the system states and inputs and the flat flag.

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



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

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



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

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

(a) Show that the system is differentially flat using the center of the rear wheels of the trailer as the flat output.

(b) Generate a trajectory for the system that steers the vehicle from an initial condition with the truck and trailer perpendicular to the loading dock into the loading dock.

(c) Write a simulation of the system stabilizes the desired trajectory and demonstrate your two-degree of freedom control system maneuvering from several different initial conditions into the parking space, with either disturbances or modeling errors included in the simulation.

Chapter 3

Optimal Control

This chapter expands on Chapter 7 of FBS2e, which introduces the concepts of reachability and state feedback. We also expand on topics in Section 8.5 of FBS2e in the area of feedforward compensation. Beginning with a review of optimization, we introduce the notion of Lagrange multipliers and provide a summary of the Pontryagin's maximum principle. Using these tools we derive the linear quadratic regulator for linear systems and describe its usage.

Prerequisites. Readers should be familiar with modeling of input/output control systems using differential equations, linearization of a system around an equilibrium point, and state space control of linear systems, including reachability and eigenvalue assignment. Some familiarity with optimization of nonlinear functions is also assumed.

3.1 Review: Optimization

Optimization refers to the problem of choosing a set of parameters that maximize or minimize a given function. In control systems, we are often faced with having to choose a set of parameters for a control law so that the some performance condition is satisfied. In this chapter we will seek to optimize a given specification, choosing the parameters that maximize the performance (or minimize the cost). In this section we review the conditions for optimization of a static function, and then extend this to optimization of trajectories and control laws in the remainder of the chapter. More information on basic techniques in optimization can be found in [Lue97] or the introductory chapter of [LS95].

Consider first the problem of finding the minimum of a smooth function $F\colon \mathbb{R}^n \to \mathbb{R}$. That is, we wish to find a point $x^* \in \mathbb{R}^n$ such that $F(x^*) \leq F(x)$ for all $x \in \mathbb{R}^n$. A necessary condition for x^* to be a minimum is that the gradient of the function be zero at x^*:

$$\frac{\partial F}{\partial x}(x^*) = 0.$$

The function $F(x)$ is often called a cost function and x^* is the optimal value for x. Figure 3.1 gives a graphical interpretation of the necessary condition for a minimum. Note that these are *not* sufficient conditions; the points x_1, x_2, and x^* in the figure

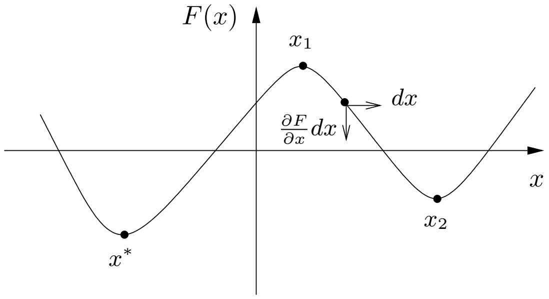
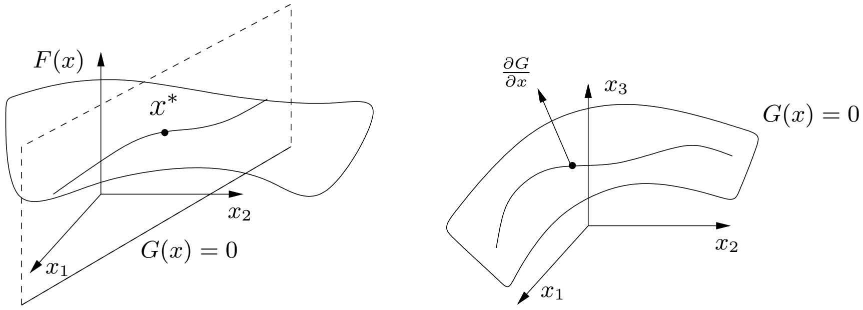


Figure 3.1: Optimization of functions. The minimum of a function occurs at a point where the gradient is zero.



(a) Constrained optimization (b) Constraint normal vectors

Figure 3.2: Optimization with constraints. (a) We seek a point x^* that minimizes $F(x)$ while lying on the surface $G(x) = 0$ (a line in the x_1x_2 plane). (b) We can parameterize the constrained directions by computing the gradient of the constraint G. Note that $x \in \mathbb{R}^2$ in (a), with the third dimension showing $F(x)$, while $x \in \mathbb{R}^3$ in (b).

all satisfy the necessary condition but only one is the (global) minimum.

The situation is more complicated if constraints are present. Let $G_i \colon \mathbb{R}^n \to \mathbb{R}$, $i = 1, \dots, k$ be a set of smooth functions with $G_i(x) = 0$ representing the constraints. Suppose that we wish to find $x^* \in \mathbb{R}^n$ such that $G_i(x^*) = 0$ and $F(x^*) \le F(x)$ for all $x \in \{x \in \mathbb{R}^n \colon G_i(x) = 0, i = 1, \dots, k\}$. This situation can be visualized as constraining the point to a surface (defined by the constraints) and searching for the minimum of the cost function along this surface, as illustrated in Figure 3.2a.

A necessary condition for being at a minimum is that there are no directions tangent to the constraints that also decrease the cost. Given a constraint function $G(x) = (G_1(x), \dots, G_k(x))$, $x \in \mathbb{R}^n$ we can represent the constraint as a $n - k$ dimensional surface in $\mathbb{R}^n$, as shown in Figure 3.2b. The tangent directions to the surface can be computed by considering small perturbations of the constraint that remain on the surface:

$$G_i(x + \delta x) \approx G_i(x) + \frac{\partial G_i}{\partial x}(x)\delta x = 0. \qquad \Longrightarrow \qquad \frac{\partial G_i}{\partial x}(x)\delta x = 0,$$

where $\delta x \in \mathbb{R}^n$ is a vanishingly small perturbation. It follows that the normal directions to the surface are spanned by $\partial G_i/\partial x$, since these are precisely the vectors that annihilate an admissible tangent vector δx.

Using this characterization of the tangent and normal vectors to the constraint, a necessary condition for optimization is that the gradient of F is spanned by vectors that are normal to the constraints, so that the only directions that increase the cost violate the constraints. We thus require that there exist scalars λ_i, $i = 1, \dots, k$ such that

$$\frac{\partial F}{\partial x}(x^*) + \sum_{i=1}^{k} \lambda_i \frac{\partial G_i}{\partial x}(x^*) = 0.$$

If we let $G = \begin{bmatrix} G_1 & G_2 & \dots & G_k \end{bmatrix}^\mathsf{T}$, then we can write this condition as

$$\frac{\partial F}{\partial x} + \lambda^\mathsf{T} \frac{\partial G}{\partial x} = 0 \tag{3.1}$$

the term $\partial F/\partial x$ is the usual (gradient) optimality condition while the term $\partial G/\partial x$ is used to "cancel" the gradient in the directions normal to the constraint.

An alternative condition can be derived by modifying the cost function to incorporate the constraints. Defining $\widetilde{F} = F + \sum \lambda_i G_i$, the necessary condition becomes

$$\frac{\partial \widetilde{F}}{\partial x}(x^*) = 0.$$

The scalars λ_i are called *Lagrange multipliers*. Minimizing $\widetilde{F}$ is equivalent to the optimization given by

$$\min_x \left(F(x) + \lambda^\mathsf{T} G(x) \right). \tag{3.2}$$

The variables λ can be regarded as free variables, which implies that we need to choose x such that $G(x) = 0$ in order to insure the cost is minimized. Otherwise, we could choose λ to generate a large cost.

Example 3.1 Two free variables with a constraint

Consider the cost function given by

$$F(x) = F_0 + (x_1 - a)^2 + (x_2 - b)^2,$$

which has an unconstrained minimum at $x = (a, b)$. Suppose that we add a constraint $G(x) = 0$ given by

$$G(x) = x_1 - x_2.$$

With this constraint, we seek to optimize F subject to $x_1 = x_2$. Although in this case we could do this by simple substitution, we instead carry out the more general procedure using Lagrange multipliers.

The augmented cost function is given by

$$\tilde{F}(x) = F_0 + (x_1 - a)^2 + (x_2 - b)^2 + \lambda(x_1 - x_2),$$

where λ is the Lagrange multiplier for the constraint. Taking the derivative of $\tilde{F}$, we have

$$\frac{\partial \tilde{F}}{\partial x} = \begin{bmatrix} 2x_1 - 2a + \lambda & 2x_2 - 2b - \lambda \end{bmatrix}.$$

Setting each of these equations equal to zero, we have that at the minimum

$$x_1^* = a - \lambda/2, \qquad x_2^* = b + \lambda/2.$$

The remaining equation that we need is the constraint, which requires that $x_1^* = x_2^*$. Using these three equations, we see that $\lambda^* = a - b$ and we have

$$x_1^* = \frac{a+b}{2}, \qquad x_2^* = \frac{a+b}{2}.$$

To verify the geometric view described above, note that the gradients of F and G are given by

$$\frac{\partial F}{\partial x} = \begin{bmatrix} 2x_1 - 2a & 2x_2 - 2b \end{bmatrix}, \qquad \frac{\partial G}{\partial x} = \begin{bmatrix} 1 & -1 \end{bmatrix}.$$

At the optimal value of the (constrained) optimization, we have

$$\frac{\partial F}{\partial x} = \begin{bmatrix} b - a & a - b \end{bmatrix}, \qquad \frac{\partial G}{\partial x} = \begin{bmatrix} 1 & -1 \end{bmatrix}.$$

Although the derivative of F is not zero, it is pointed in a direction that is normal to the constraint, and hence we cannot decrease the cost while staying on the constraint surface. ∇

We have focused on finding the minimum of a function. We can switch back and forth between maximum and minimum by simply negating the cost function:

$$\max_x F(x) = \min_x \bigl(-F(x)\bigr)$$

We see that the conditions that we have derived are independent of the sign of F since they only depend on the gradient begin zero in approximate directions. Thus finding x^* that satisfies the conditions corresponds to finding an *extremum* for the function.

Very good software is available for numerically solving optimization problems of this sort. The NPSOL, SNOPT, and IPOPT libraries are available in FORTRAN and C. In Python, the scipy.optimize module of SciPy can be used to solve constrained optimization problems.

3.2 Optimal Control of Systems

We now return to the problem of finding a feasible trajectory for a system that satisfies some performance condition. The basic idea is to try to optimize a given cost function over all trajectories of a system that satisfy the possible dynamics of the system. The input to the system is used to "parameterize" the possible trajectories of the system.

More concretely, we consider the *optimal control problem*:

$$\min_{u(\cdot)} \int_0^T L(x,u)\,dt + V\bigl(x(T)\bigr)$$

subject to the constraint

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

Abstractly, this is a constrained optimization problem where we seek a *feasible trajectory* $(x(t), u(t))$ that minimizes the cost function

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

More formally, this problem is equivalent to the "standard" problem of minimizing a cost function $J(x, u)$ where $(x, u) \in L_2[0, T]$ (the set of square integrable functions) and $g(z) = \dot{x}(t) - f(x(t), u(t)) = 0$ models the dynamics. The term $L(x, u)$ is referred to as the integral cost and $V(x(T))$ is the final (or terminal) cost.

There are many variations and special cases of the optimal control problem. We mention a few here:

Infinite horizon optimal control. If we let $T = \infty$ and set $V = 0$, then we seek to optimize a cost function over all time. This is called the *infinite horizon* optimal control problem, versus the *finite horizon* problem with $T < \infty$. Note that if an infinite horizon problem has a solution with finite cost, then the integral cost term $L(x, u)$ must approach zero as $t \to \infty$.

Linear quadratic (LQ) optimal control. If the dynamical system is linear and the cost function is quadratic, we obtain the *linear quadratic* optimal control problem:

$$\dot{x} = Ax + Bu, \qquad J = \int_0^T \left(x^\mathsf{T} Q x + u^\mathsf{T} R u\right)\, dt + x^\mathsf{T}(T) P_1 x(T).$$

In this formulation, $Q \geq 0$ penalizes state error, $R > 0$ penalizes the input and $P_1 > 0$ penalizes terminal state. This problem can be modified to track a desired trajectory (x_d, u_d) by rewriting the cost function in terms of $(x - x_d)$ and $(u - u_d)$.

Terminal constraints. It is often convenient to ask that the final value of the trajectory, denoted x_f, be specified. We can do this by requiring that $x(T) = x_\mathrm{f}$ or by using a more general form of constraint:

$$\psi_i(x(T)) = 0, \qquad i = 1, \dots, q.$$

The fully constrained case is obtained by setting $q = n$ and defining $\psi_i(x(T)) = x_i(T) - x_{i,f}$. For a control problem with a full set of terminal constraints, $V(x(T))$ can be omitted (since its value is fixed).

Time optimal control. If we constrain the terminal condition to $x(T) = x_\mathrm{f}$, let the terminal time T be free (so that we can optimize over it) and choose $L(x, u) = 1$, we can find the *time-optimal* trajectory between an initial and final condition. This problem is usually only well-posed if we additionally constrain the inputs u to be bounded.

A very general set of conditions is available for the optimal control problem that captures most of these special cases in a unifying framework. Consider a nonlinear system

$$\begin{aligned} &\dot{x} = f(x, u), && x = \mathbb{R}^n, \\ &x(0) \text{ given}, && u \in \Omega \subset \mathbb{R}^m, \end{aligned}$$

where $f(x,u) = (f_1(x,u), \dots f_n(x,u))\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. We wish to minimize a cost function J with terminal constraints:

$$J = \int_0^T L(x,u)\,dt + V(x(T)), \qquad \psi(x(T)) = 0.$$

The function $\psi\colon \mathbb{R}^n \to \mathbb{R}^q$ gives a set of q terminal constraints. Analogous to the case of optimizing a function subject to constraints, we construct the *Hamiltonian*:

$$H = L + \lambda^{\mathsf{T}} f = L + \sum_{i=1}^{n} \lambda_i f_i.$$

The variables λ are functions of time and are often referred to as the *costate variables.* A set of necessary conditions for a solution to be optimal was derived by Pontryagin [PBGM62].

Theorem 3.1 (Maximum Principle)**.** *If (x^*, u^*) is optimal, then there exists $\lambda^*(t) \in \mathbb{R}^n$ and $\nu^* \in \mathbb{R}^q$ such that*

$$\begin{aligned} \dot{x}_i^* &= \frac{\partial H}{\partial \lambda_i}(x^*, \lambda^*), &\qquad & x(0) \textit{ given, } \psi(x(T)) = 0, \\ -\dot{\lambda}_i^* &= \frac{\partial H}{\partial x_i}(x^*, \lambda^*), &\qquad & \lambda_i^*(T) = \frac{\partial V}{\partial x_i}\big(x^*(T)\big) + \sum_{j=1}^{q} \nu_j^* \frac{\partial \psi_j}{\partial x_i}\big(x^*(T)\big), \end{aligned}$$

and

$$H(x^*(t), u^*(t), \lambda^*(t)) \leq H(x^*(t), u, \lambda^*(t)) \quad \textit{for all} \quad u \in \Omega.$$

The form of the optimal solution is given by the solution of a differential equation with boundary conditions. If $u = \arg\min H(x,u,\lambda)$ exists, we can use this to choose the control law u and solve for the resulting feasible trajectory that minimizes the cost. The boundary conditions are given by the n initial states $x(0)$, the q terminal constraints on the state $\psi(x(T)) = 0$, and the $n - q$ final values for the Lagrange multipliers

$$\lambda^{\mathsf{T}}(T) = \frac{\partial V}{\partial x}\left(x(T)\right) + \nu^{\mathsf{T}} \frac{\partial \psi}{\partial x}\left(x(T)\right).$$

In this last equation, ν is a free variable and so there are n equations in $n + q$ free variables, leaving $n - q$ constraints on $\lambda(T)$. In total, we thus have $2n$ boundary values.

The maximum principle is a very general (and elegant) theorem. It allows the dynamics to be nonlinear and the input to be constrained to lie in a set Ω, allowing the possibility of bounded inputs. If $\Omega = \mathbb{R}^m$ (unconstrained input) and H is differentiable, then a necessary condition for the optimal input is

$$\frac{\partial H}{\partial u_i} = 0, \qquad i = 1, \dots, m.$$

We note that even though we are *minimizing* the cost, this is still usually called the maximum principle (an artifact of history).

Sketch of proof. We follow the proof given by Lewis and Syrmos [LS95], omitting some of the details required for a fully rigorous proof. We use the method of Lagrange multipliers, augmenting our cost function by the dynamical constraints and the terminal constraints:

$$\begin{aligned}\tilde{J}(x(\cdot), u(\cdot), \lambda(\cdot), \nu) &= J(x, u) + \int_0^T \lambda^\mathsf{T}(t)\bigl(f(x, u) - \dot{x}(t)\bigr)\, dt + \nu^\mathsf{T}\psi(x(T)) \\ &= \int_0^T \bigl(L(x, u) + \lambda^\mathsf{T}(t)\bigl(f(x, u) - \dot{x}(t)\bigr)\, dt \\ &\qquad + V(x(T)) + \nu^\mathsf{T}\psi(x(T)).\end{aligned}$$

Note that λ is a function of time, with each $\lambda(t)$ corresponding to the instantaneous constraint imposed by the dynamics. The integral over the interval $[0, T]$ plays the role of the sum of the finite constraints in the regular optimization.

Making use of the definition of the Hamiltonian, the augmented cost becomes

$$\tilde{J}(x(\cdot), u(\cdot), \lambda(\cdot), \nu) = \int_0^T \bigl(H(x, u) - \lambda^\mathsf{T}(t)\dot{x}\bigr)\, dt + V(x(T)) + \nu^\mathsf{T}\psi(x(T)).$$

We can now "linearize" the cost function around the optimal solution $x(t) = x^*(t) + \delta x(t)$, $u(t) = u^*(t) + \delta u(t)$, $\lambda(t) = \lambda^*(t) + \delta\lambda(t)$ and $\nu = \nu^* + \delta\nu$. Taking T as fixed for simplicity (see [LS95] for the more general case), the incremental cost can be written as

$$\begin{aligned}\delta\tilde{J} &= \tilde{J}(x^* + \delta x, u^* + \delta u, \lambda^* + \delta\lambda, \nu^* + \delta\nu) - \tilde{J}(x^*, u^*, \lambda^*, \nu^*) \\ &\approx \int_0^T \Bigl(\frac{\partial H}{\partial x}\delta x + \frac{\partial H}{\partial u}\delta u - \lambda^\mathsf{T}\delta\dot{x} + \Bigl(\frac{\partial H}{\partial \lambda} - \dot{x}^\mathsf{T}\Bigr)\delta\lambda\Bigr)\, dt \\ &\qquad + \frac{\partial V}{\partial x}\delta x(T) + \nu^\mathsf{T}\frac{\partial \psi}{\partial x}\delta x(T) + \delta\nu^\mathsf{T}\psi\bigl(x(T), u(T)\bigr),\end{aligned}$$

where we have omitted the time argument inside the integral and all derivatives are evaluated along the optimal solution.

We can eliminate the dependence on $\delta\dot{x}$ using integration by parts:

$$-\int_0^T \lambda^\mathsf{T}\delta\dot{x}\, dt = -\lambda^\mathsf{T}(T)\delta x(T) + \lambda^\mathsf{T}(0)\delta x(0) + \int_0^T \dot{\lambda}^\mathsf{T}\delta x\, dt.$$

Since we are requiring $x(0) = x_0$, the $\delta x(0)$ term vanishes and substituting this into $\delta\tilde{J}$ yields

$$\begin{aligned}\delta\tilde{J} &\approx \int_0^T \Bigl[\Bigl(\frac{\partial H}{\partial x} + \dot{\lambda}^\mathsf{T}\Bigr)\delta x + \frac{\partial H}{\partial u}\delta u + \Bigl(\frac{\partial H}{\partial \lambda} - \dot{x}^\mathsf{T}\Bigr)\delta\lambda\Bigr]\, dt \\ &\qquad + \Bigl(\frac{\partial V}{\partial x} + \nu^\mathsf{T}\frac{\partial \psi}{\partial x} - \lambda^\mathsf{T}(T)\Bigr)\delta x(T) + \delta\nu^\mathsf{T}\psi\bigl(x(T), u(T)\bigr).\end{aligned}$$

To be optimal, we require $\delta\tilde{J} = 0$ for all δx, δu, $\delta\lambda$ and $\delta\nu$, and we obtain the (local) conditions in the theorem. □

3.3 Examples

To illustrate the use of the maximum principle, we consider a number of analytical examples. Additional examples are given in the exercises.

Example 3.2 Scalar linear system
Consider the optimal control problem for the system

$$\dot{x} = ax + bu, \tag{3.3}$$

where $x = \mathbb{R}$ is a scalar state, $u \in \mathbb{R}$ is the input, the initial state $x(t_0)$ is given, and $a, b \in \mathbb{R}$ are positive constants. We wish to find a trajectory $(x(t), u(t))$ that minimizes the cost function

$$J = \tfrac{1}{2}\int_{t_0}^{t_\mathrm{f}} u^2(t)\,dt + \tfrac{1}{2}cx^2(t_\mathrm{f}),$$

where the terminal time t_f is given and $c > 0$ is a constant. This cost function balances the final value of the state with the input required to get to that state.

To solve the problem, we define the various elements used in the maximum principle. Our integral and terminal costs are given by

$$L = \tfrac{1}{2}u^2(t), \qquad V = \tfrac{1}{2}cx^2(t_\mathrm{f}).$$

We write the Hamiltonian of this system and derive the following expressions for the costate λ:

$$\begin{aligned} H &= L + \lambda f = \tfrac{1}{2}u^2 + \lambda(ax + bu), \\ \dot{\lambda} &= -\frac{\partial H}{\partial x} = -a\lambda, \qquad \lambda(t_\mathrm{f}) = \frac{\partial V}{\partial x} = cx(t_\mathrm{f}). \end{aligned}$$

This is a final value problem for a linear differential equation in λ and the solution can be shown to be

$$\lambda(t) = cx(t_\mathrm{f})e^{a(t_\mathrm{f}-t)}.$$

The optimal control is given by

$$\frac{\partial H}{\partial u} = u + b\lambda = 0 \quad \Rightarrow \quad u^*(t) = -b\lambda(t) = -bcx(t_\mathrm{f})e^{a(t_\mathrm{f}-t)}.$$

Substituting this control into the dynamics given by equation (3.3) yields a first-order ODE in x:

$$\dot{x} = ax - b^2cx(t_\mathrm{f})e^{a(t_\mathrm{f}-t)}.$$

This can be solved explicitly as

$$x^*(t) = x(t_0)e^{a(t-t_0)} + \frac{b^2c}{2a}x^*(t_\mathrm{f})\left[e^{a(t_\mathrm{f}-t)} - e^{a(t+t_\mathrm{f}-2t_0)}\right].$$

Setting $t = t_\mathrm{f}$ and solving for $x(t_\mathrm{f})$ gives

$$x^*(t_\mathrm{f}) = \frac{2a\,e^{a(t_\mathrm{f}-t_0)}x(t_0)}{2a - b^2c\left(1 - e^{2a(t_\mathrm{f}-t_0)}\right)},$$

and finally we can write

$$u^*(t) = -\frac{2abc\, e^{a(2t_\mathrm{f}-t_0-t)} x(t_0)}{2a - b^2 c\left(1 - e^{2a(t_\mathrm{f}-t_0)}\right)}, \tag{3.4}$$

$$x^*(t) = x(t_0)e^{a(t-t_0)} + \frac{b^2 c\, e^{a(t_\mathrm{f}-t_0)} x(t_0)}{2a - b^2 c\left(1 - e^{2a(t_\mathrm{f}-t_0)}\right)} \left[e^{a(t_\mathrm{f}-t)} - e^{a(t+t_\mathrm{f}-2t_0)}\right]. \tag{3.5}$$

We can use the form of this expression to explore how our cost function affects the optimal trajectory. For example, we can ask what happens to the terminal state $x^*(t_\mathrm{f})$ as $c \to \infty$. Setting $t = t_\mathrm{f}$ in equation (3.5) and taking the limit we find that

$$\lim_{c\to\infty} x^*(t_\mathrm{f}) = 0.$$

∇

Example 3.3 Bang-bang control

The time-optimal control program for a linear system has a particularly simple solution. Consider a linear system with bounded input

$$\dot{x} = Ax + Bu, \qquad |u| \le 1,$$

and suppose we wish to minimize the time required to move from an initial state x_0 to a final state x_f. Without loss of generality we can take $x_\mathrm{f} = 0$. We choose the cost functions and terminal constraints to satisfy

$$J = \int_0^T 1\, dt, \qquad \psi(x(T)) = x(T).$$

In this case, the time T is not fixed and so it turns out that one additional condition is required for the optimal controller. For the case considered in Theorem 3.1, where the cost functions and constraints do not depend explicitly on time, the additional condition is

$$H(x(T), u(T)) = 0$$

(see [LS95]).

To find the optimal control, we form the Hamiltonian

$$H = 1 + \lambda^\mathsf{T}(Ax + Bu) = 1 + (\lambda^\mathsf{T}A)x + (\lambda^\mathsf{T}B)u.$$

Now apply the conditions in the maximum principle:

$$\begin{aligned}
\dot{x} &= \left(\frac{\partial H}{\partial \lambda}\right)^\mathsf{T} = Ax + Bu, \\
-\dot{\lambda} &= \left(\frac{\partial H}{\partial x}\right)^\mathsf{T} = A^\mathsf{T}\lambda, \\
u &= \text{arg min } H = -\operatorname{sgn}(\lambda^\mathsf{T}B), \\
1 &+ \lambda^\mathsf{T}(T)(Ax(T) + Bu(T)) = 0.
\end{aligned}$$

The optimal solution always satisfies this set of equations (since the maximum principle gives a necessary condition) with $x(0) = x_0$ and $x(T) = 0$. It follows that the input is always either +1 or −1, depending on $\lambda^\mathsf{T} B$. This type of control is called "bang-bang" control since the input is always on one of its limits. If $\lambda^\mathsf{T}(t)B = 0$ then the control is not well defined, but if this is only true for a specific time instant (e.g., $\lambda^\mathsf{T}(t)B$ crosses zero) then the analysis still holds. ∇

3.4 Implementation in Python[1]

The `optimal` module of the Python Control Systems Library (python-control) provides support for optimization-based controllers for nonlinear systems with state and input constraints.

The optimal control module provides a means of computing optimal trajectories for nonlinear systems and implementing optimization-based controllers, including model predictive control (described in Chapter 4). The basic optimal control problem is to solve the optimization

$$\min_{u(\cdot)} \int_0^T L(x,u)\,dt + V\bigl(x(T)\bigr)$$

subject to the constraint

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

Constraints on the states and inputs along the trajectory and at the end of the trajectory can also be specified:

$$\begin{aligned} \mathrm{lb}_i \leq g_i(x,u) \leq \mathrm{ub}_i, &\qquad i = 1,\dots,k \\ \psi_i(x(T)) = 0, &\qquad i = 1,\dots,q. \end{aligned}$$

The python-control implementation of optimal control follows the basic problem setup described here, but carries out all computations in discrete time (so that integrals become sums) and over a finite horizon.

To describe an optimal control problem we need an input/output system, a time horizon, a cost function, and (optionally) a set of constraints on the state and/or input, either along the trajectory and at the terminal time. The optimal control module operates by converting the optimal control problem into a standard optimization problem that can be solved by `scipy.optimize.minimize()`. The optimal control problem can be solved by using the `solve_ocp()` function:

```
res = ct.optimal.solve_ocp(sys, timepts, X0, cost, constraints)
```

The `sys` parameter should be an `InputOutputSystem` and the `timepts` parameter should represent a time vector that gives the list of times at which the cost and constraints should be evaluated.

The cost function has call signature `cost(x, u)` and should return the (incremental) cost at the given time, state, and input. It will be evaluated at each point in the time points vector. The `terminal_cost` parameter can be used to specify a cost function for the final point in the trajectory.

The constraints parameter is a list of constraints similar to that used by the `scipy.optimize.minimize()` function. Each constraint is in one of the following forms:

---

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

```
LinearConstraint(A, lb, ub)
NonlinearConstraint(f, lb, ub)
```

For a linear constraint, the 2D array `A` is multiplied by a vector consisting of the current state `x` and current input `u` stacked vertically, then compared with the upper and lower bound. This constraint is satisfied if

```
lb <= A @ np.hstack([x, u]) <= ub
```

A nonlinear constraint is satisfied if

```
lb <= f(x, u) <= ub
```

By default, constraints are taken to be trajectory constraints holding at all points on the trajectory. The `terminal_constraint` parameter can be used to specify a constraint that only holds at the final point of the trajectory.

The return value for `solve_ocp()` is a bundle object that has the following elements:

`res.success`: `True` if solved successfully
`res.inputs`: optimal input
`res.states`: state trajectory (if `return_x == True`)
`res.time`: copy of the time points vector

In addition, the results from `scipy.optimize.minimize()` are also available.

Providing a reasonable initial guess is often needed in order for the optimizer to find a good answer. The `initial_guess` parameter provides trajectories for the initial states and/or inputs to use as a guess for the optimal trajectory. The form of the guess depends on whether the method used to solve the problem is based on collocation or shooting (see Section 4.2 for more information on these methods). For collocation methods, the initial guess is either the input vector or a 2-tuple consisting guesses for the state and the input. For shooting methods, an array of inputs for each time point should be specified. For an optimal control problem that uses a terminal condition or a large terminal cost to get to a neighborhood of a final point, a straight line in the state space often works well.

3.5 Linear Quadratic Regulators

In addition to its use for computing optimal, feasible trajectories for a system, we can also use optimal control theory to design a feedback law $u = \alpha(x)$ that stabilizes a given equilibrium point. Roughly speaking, we do this by continuously re-solving the optimal control problem from our current state $x(t)$ and applying the resulting input $u(t)$. Of course, this approach is impractical unless we can solve explicitly for the optimal control and somehow rewrite the optimal control as a function of the current state in a simple way. In this section we explore exactly this approach for the linear quadratic optimal control problem.

We begin with the the finite horizon, linear quadratic regulator (LQR) problem, given by

$$\begin{aligned}\dot{x} &= Ax + Bu, \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^n, x_0 \text{ given},\\ \tilde{J} &= \frac{1}{2}\int_0^T \left(x^\mathsf{T} Q_x x + u^\mathsf{T} Q_u u\right)\, dt + \frac{1}{2} x^\mathsf{T}(T) P_1 x(T),\end{aligned}$$

where $Q_x \geq 0$, $Q_u > 0$, $P_1 \geq 0$ are symmetric, positive (semi-) definite matrices. Note the factor of $\frac{1}{2}$ is usually left out, but we included it here to simplify the derivation (the optimal control will be unchanged if we multiply the entire cost function by 2).

To find the optimal control, we apply the maximum principle. We begin by computing the Hamiltonian H:

$$H = \frac{1}{2} x^\mathsf{T} Q_x x + \frac{1}{2} u^\mathsf{T} Q_u u + \lambda^\mathsf{T}(Ax + Bu).$$

Applying the results of Theorem 3.1, we obtain the necessary conditions

$$\begin{aligned} \dot{x} &= \left(\frac{\partial H}{\partial \lambda}\right)^\mathsf{T} = Ax + Bu, & x(0) &= x_0, \\ -\dot{\lambda} &= \left(\frac{\partial H}{\partial x}\right)^\mathsf{T} = Q_x x + A^\mathsf{T} \lambda, & \lambda(T) &= P_1 x(T), \\ 0 &= \left(\frac{\partial H}{\partial u}\right)^\mathsf{T} = Q_u u + B^\mathsf{T} \lambda. \end{aligned} \tag{3.6}$$

The last condition can be solved to obtain the optimal controller

$$u = -Q_u^{-1} B^\mathsf{T} \lambda,$$

which can be substituted into the dynamic equation (3.6) To solve for the optimal control we must solve a *two point boundary value problem* using the initial condition $x(0)$ and the final condition $\lambda(T)$. Unfortunately, it is very hard to solve such problems in general.

Given the linear nature of the dynamics, we attempt to find a solution by setting $\lambda(t) = P(t)x(t)$ where $P(t) \in \mathbb{R}^{n\times n}$. Substituting this into the necessary condition, we obtain

$$\begin{aligned} \dot{\lambda} &= \dot{P}x + P\dot{x} = \dot{P}x + P(Ax - BQ_u^{-1}B^\mathsf{T}P)x, \\ &\implies \quad -\dot{P}x - PAx + PBQ_u^{-1}BPx = Q_x x + A^\mathsf{T} P x. \end{aligned}$$

This equation is satisfied if we can find $P(t)$ such that

$$-\dot{P} = PA + A^\mathsf{T} P - PBQ_u^{-1} B^\mathsf{T} P + Q_x, \qquad P(T) = P_1. \tag{3.7}$$

This is a *matrix differential equation* that defines the elements of $P(t)$ from a final value $P(T)$. Solving it is conceptually no different than solving the initial value problem for vector-valued ordinary differential equations, except that we must solve for the individual elements of the matrix $P(t)$ backwards in time. Equation (3.7) is called the *Riccati ODE*.

An important property of the solution to the optimal control problem when written in this form is that $P(t)$ can be solved without knowing either $x(t)$ or $u(t)$. This allows the two point boundary value problem to be separated into first solving a final-value problem and then solving a time-varying initial value problem. More specifically, given $P(t)$ satisfying equation (3.7), we can apply the optimal input

$$u(t) = -Q_u^{-1} B^\mathsf{T} P(t) x.$$

and then solve the original dynamics of the system forward in time from the initial condition $x(0) = x_0$. Note that this is a (time-varying) *feedback* control that describes how to optimally move from *any* state toward the origin in time T.

An important variation of this problem is the case when we choose $T = \infty$ and eliminate the terminal cost (set $P_1 = 0$). This gives us the cost function

$$J = \int_0^\infty (x^\mathsf{T} Q_x x + u^\mathsf{T} Q_u u)\, dt. \tag{3.8}$$

Since we do not have a terminal cost, there is no constraint on the final value of λ or, equivalently, $P(T)$. We can thus seek to find a constant P satisfying equation (3.7). In other words, we seek to find P such that

$$PA + A^\mathsf{T} P - PBQ_u^{-1} B^\mathsf{T} P + Q_x = 0. \tag{3.9}$$

This equation is called the *algebraic Riccati equation.* Given a solution, we can choose our input as

$$u = -Q_u^{-1} B^\mathsf{T} P x.$$

This represents a *constant gain* $K = Q_u^{-1} B^\mathsf{T} P$ where P is the solution of the algebraic Riccati equation.

The implications of this result are interesting and important. First, we notice that if $Q_x > 0$ and the control law corresponds to a finite minimum of the cost, then we must have that $\lim_{t\to\infty} x(t) = 0$, otherwise the cost will be unbounded. This means that the optimal control for moving from any state x toward the origin can be achieved by applying a feedback $u = -Kx$ for K chosen as described as above and letting the system evolve in closed loop. More amazingly, the gain matrix K can be written in terms of the solution to a (matrix) quadratic equation (3.9). This quadratic equation can be solved numerically: in python-control the command `K, P, E = ct.lqr(A, B, Qx, Qu)` provides the optimal feedback compensator K, the solution to the algebraic Riccati equation P, and the closed loop eigenvalues for the system E.

In deriving the optimal quadratic regulator, we have glossed over a number of important details. It is clear from the form of the solution that we must have $Q_u > 0$ since its inverse appears in the solution. We would typically also have $Q_x > 0$ so that the integral cost is only zero when $x = 0$, but in some instances we might only care about certain states, which would imply that $Q_x \geq 0$. For this case, if we let $Q_x = H^\mathsf{T} H$ (always possible), our cost function becomes

$$J = \int_0^\infty x^\mathsf{T} H^\mathsf{T} H x + u^\mathsf{T} Q_u u\, dt = \int_0^\infty \|Hx\|^2 + u^\mathsf{T} Q_u u\, dt.$$

A technical condition for the optimal solution to exist is that the pair (A, H) be *detectable* (implied by observability). This makes sense intuitively by considering $y = Hx$ as an output. If y is not observable then there may be non-zero initial conditions that produce no output and so the cost would be zero. This would lead to an ill-conditioned problem and hence we will require that $Q_x \geq 0$ satisfy an appropriate observability condition.

We summarize the main results as a theorem.

Theorem 3.2. *Consider a linear control system with quadratic cost:*

$$\dot{x} = Ax + Bu, \qquad J = \int_0^\infty x^\mathsf{T} Q_x x + u^\mathsf{T} Q_u u \, dt.$$

Assume that the system defined by (A, B) is reachable, $Q_x = Q_x^\mathsf{T} \geq 0$ and $Q_u = Q_u^\mathsf{T} > 0$. Further assume that $Q_x = H^\mathsf{T} H$ and that the linear system with dynamics matrix A and output matrix H is observable. Then the optimal controller satisfies

$$u = -Q_u^{-1} B^\mathsf{T} P x, \qquad PA + A^\mathsf{T} P - PBQ_u^{-1} B^\mathsf{T} P = -Q_x,$$

and the minimum cost from initial condition $x(0)$ is given by $J^* = x^\mathsf{T}(0) P x(0)$.

The basic form of the solution follows from the necessary conditions, with the theorem asserting that a constant solution exists for $T = \infty$ when the additional conditions are satisfied. The full proof can be found in standard texts on optimal control, such as Lewis and Syrmos [LS95] or Athans and Falb [AF06]. A simplified version, in which we first assume the optimal control is linear, is left as an exercise.

Example 3.4 Optimal control of a double integrator

Consider a double integrator system

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

with quadratic cost given by

$$Q_x = \begin{bmatrix} q^2 & 0 \\ 0 & 0 \end{bmatrix}, \qquad Q_u = 1.$$

The optimal control is given by the solution of matrix Riccati equation (3.9). Let P be a symmetric positive definite matrix of the form

$$P = \begin{bmatrix} a & b \\ b & c \end{bmatrix}.$$

Then the Riccati equation becomes

$$\begin{bmatrix} -b^2 + q^2 & a - bc \\ a - bc & 2b - c^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},$$

which has solution

$$P = \begin{bmatrix} \sqrt{2q^3} & q \\ q & \sqrt{2q} \end{bmatrix}.$$

The controller is given by

$$K = Q_u^{-1} B^\mathsf{T} P = [q \quad \sqrt{2q}].$$

The feedback law minimizing the given cost function is then $u = -Kx$.

To better understand the structure of the optimal solution, we examine the eigenstructure of the closed loop system. The closed-loop dynamics matrix is given by

$$A_{cl} = A - BK = \begin{bmatrix} 0 & 1 \\ -q & -\sqrt{2q} \end{bmatrix}.$$

The characteristic polynomial of this matrix is

$$\lambda^2 + \sqrt{2q}\lambda + q.$$

Comparing this to $\lambda^2 + 2\zeta\omega_0\lambda + \omega_0^2$, we see that

$$\omega_0 = \sqrt{q}, \qquad \zeta = \frac{1}{\sqrt{2}}.$$

Thus the optimal controller gives a closed loop system with damping ratio $\zeta = 0.707$, giving a good tradeoff between rise time and overshoot (see FBS2e). ∇

3.6 Choosing LQR weights

One of the key questions in LQR design is how to choose the weights Q_x and Q_u. To choose specific values for the cost function weights Q_x and Q_u, we must use our knowledge of the system we are trying to control. A particularly simple choice is to use diagonal weights

$$Q_x = \begin{bmatrix} q_1 & & 0 \\ & \ddots & \\ 0 & & q_n \end{bmatrix}, \qquad Q_u = \begin{bmatrix} \rho_1 & & 0 \\ & \ddots & \\ 0 & & \rho_n \end{bmatrix}.$$

For this choice of Q_x and Q_u, the individual diagonal elements describe how much each state and input (squared) should contribute to the overall cost. Hence, we can take states that should remain small and attach higher weight values to them. Similarly, we can penalize an input versus the states and other inputs through choice of the corresponding input weight ρ_j.

Choosing the individual weights for the (diagonal) elements of the Q_x and Q_u matrix can be done by deciding on a weighting of the errors from the individual terms. Bryson and Ho [BH75] have suggested the following method for choosing the matrices Q_x and Q_u in equation (3.8): (1) choose q_i and ρ_j as the inverse of the square of the maximum value for the corresponding x_i or u_j; (2) modify the elements to obtain a compromise among response time, damping, and control effort. This second step can be performed by trial and error.

It is also possible to choose the weights such that only a given subset of variable are considered in the cost function. Let $z = Hx$ be the output we want to keep small and verify that (A, H) is observable. Then we can use a cost function of the form

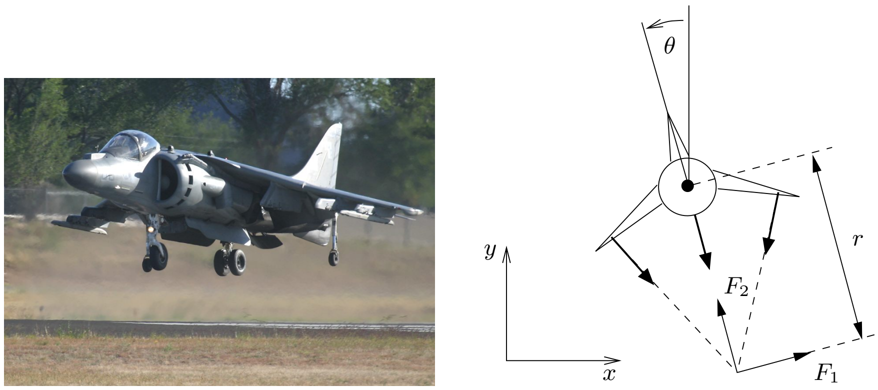
$$Q_x = H^\mathsf{T} H \qquad Q_u = \rho I.$$

The constant ρ allows us to trade off $\|z\|^2$ versus $\rho\|u\|^2$.

We illustrate the various choices through an example application.

Example 3.5 Thrust vectored aircraft

Consider the thrust vectored aircraft example introduced in FBS2e, Example 3.12. The system is shown in Figure 3.3, reproduced from FBS2e. The linear quadratic regulator problem was illustrated in FBS2e 2e, Example 7.9, where the weights were chosen as $Q_x = I$ and $Q_u = \rho I$. Figure 3.4 reproduces the step response



(a) Harrier "jump jet" (b) Simplified model

Figure 3.3: Vectored thrust aircraft. The Harrier AV-8B military aircraft (a) redirects its engine thrust downward so that it can "hover" above the ground. Some air from the engine is diverted to the wing tips to be used for maneuvering. As shown in (b), the net thrust on the aircraft can be decomposed into a horizontal force F_1 and a vertical force F_2 acting at a distance r from the center of mass.

for this case, but using the full nonlinear, multi-input/multi-output model for the system.

A more physically motivated weighted can be computing by specifying the comparable errors in each of the states and adjusting the weights accordingly. Suppose, for example that we consider a 1 cm error in x, a 10 cm error in y and a 5° error in θ to be equivalently bad. In addition, we wish to penalize the forces in the sidewards direction (F_1) since these result in a loss in efficiency. This can be accounted for in the LQR weights be choosing

$$Q_x = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 36/\pi & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad Q_u = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}.$$

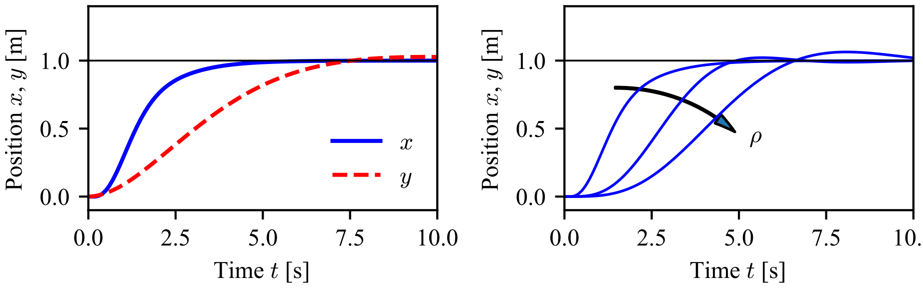
The results of this choice of weights are shown in Figure 3.5. ∇

3.7 Advanced Topics

In this section we briefly touch on some related topics in optimal control, with reference to more detailed treatments where appropriate.

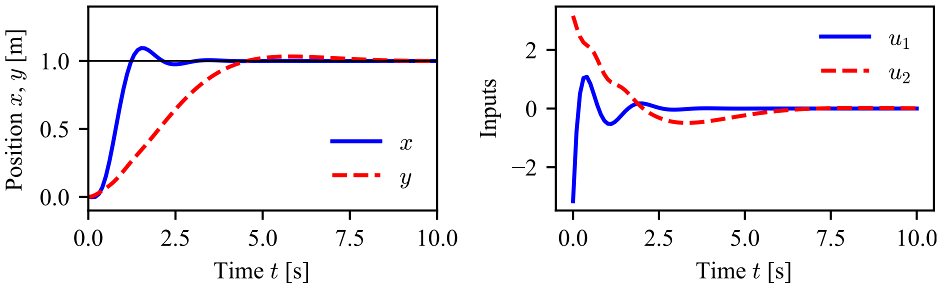
Dynamic programming

An alternative formulation to the optimal control problem is to make use of the "principle of optimality", which states (roughly) that if we are given an optimal



(a) Step response in x and y (b) Effect of control weight ρ

Figure 3.4: Step response for a vectored thrust aircraft. The plot in (a) shows the x and y positions of the aircraft when it is commanded to move 1 m in each direction. In (b) the x motion is shown for control weights $\rho = 1, 10^2, 10^4$. A higher weight of the input term in the cost function causes a more sluggish response.



(a) Step response in x and y (b) Inputs for the step response

Figure 3.5: Step response for a vector thrust aircraft using physically motivated LQR weights (a). The rise time for x is much faster than in Figure 3.4a, but there is a small oscillation and the inputs required are quite large (b).

policy, then the optimal policy from any point along the implementation of that policy must also be optimal. In the context of optimal trajectory generation, we can interpret this as saying that if we solve an optimal control problem from any state along an optimal trajectory, we will get the remainder of the optimal trajectory (how could it be otherwise!?).

The implication of this statement for trajectory generation is that we can work from the final time of our optimal control problem and compute the cost by moving backwards in time until we reach the initial time. Toward this end, we define the "cost to go" from a given state x at time t as

$$J(x,t) = \int_t^T L(x(\tau), u(\tau))\, d\tau + V(x(T)). \tag{3.10}$$

Given a state $x(t)$, We see that the cost at time T is given by $J(T,x) = V(x(T))$ and the cost at other times includes the integral of the cost from time t to T plus the terminal cost.

It can be shown that a necessary condition for a trajectory $x(\cdot)$, $u(\cdot)$ to be optimal is that the *Hamilton-Jacobi-Bellman equation* (HJB equation) be satisfied:

$$\frac{\partial J^*}{\partial t}(x,t) = -H(x, u^*, \frac{\partial J^{*\mathsf{T}}}{\partial x}(x,t)), \qquad J(x,T) = V(x), \tag{3.11}$$

where H is the Hamiltonian function, u^* is the optimal input, and $V : \mathbb{R}^n \to \mathbb{R}$ is the terminal cost. As in the case of the maximum principle, we choose u^* to minimize the Hamiltonian:

$$u^* = \arg\min_u H\bigl(x^*, u, \frac{\partial J^{*\mathsf{T}}}{\partial x}(x^*, u)\bigr).$$

Equation (3.11) is a partial differential equation for $J(x,t)$ with boundary condition $J(x,T) = V(x)$.

From the form of the Hamilton-Jacobi-Bellman equation, we see that we can interpret the costate variables λ as

$$\lambda^T = \frac{\partial J^*}{\partial x}(x,t).$$

Thus the costate variables can be thought of as the sensitivity of the cost to go at a given point along the optimal trajectory. This interpretation allows some alternative formulations of the optimal control problem, as well as additional insights.

While solving the Hamilton-Jacobi-Bellman equation is not particularly easy in the continuous case, it turns out that discrete version of the problem can make good use of the principle of optimality. In particular, for problems with variables that take on a sequence of values from a finite set (known as discrete-decision making problems), we can compute the optimal set of decisions by starting with the cost at the end of the sequence and then computing the values of the optimized cost stepping backwards in time. This technique is known as *dynamic programming* and arises in a number of different applications in computer science, economics, and other areas.

Detailed explanations of dynamic programming formulations of optimal control are available in a wide variety of textbooks. The online notes from Daniel Liberzon are a good open source resource for this material:

`http://liberzon.csl.illinois.edu/teaching/cvoc/cvoc.html`

Chapter 5 in those notes provides a more detailed explanation of the material briefly summarized here.

Integral action

Controllers based on state feedback achieve the correct steady-state response to command signals by having a good model of the system that can generate the proper feedforward signal u_d. However, if the model is incorrect or disturbances are present, there may be steady state errors. Integral feedback is a classic technique for achieving zero steady state output error in the presence of constant disturbances or errors in the feedforward signal.

The basic approach in integral feedback is to create a state within the controller that computes the integral of the error signal, which is then used as a feedback term. We do this by augmenting the description of the system with a new state z, which is the integral of the difference between the the actual output $y = h(x)$ and desired output $y_\text{d} = h(x_\text{d})$. The augmented state equations become

$$\frac{d\xi}{dt} = \frac{d}{dt}\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} f(x,u) \\ h(x) - h(x_d) \end{bmatrix} =: F(\xi, u, x_\text{d}). \tag{3.12}$$

We can now design a feedback compensator (such as an LQR controller) that stabilizes the system to the desired trajectory $\xi_\text{d} = (x_\text{d}, 0)$, which will cause y to approach $h(x_\text{d})$ in steady state.

Given the augmented system, we design a state space controller in the usual fashion, with a control law of the form

$$u = u_\text{d} - K(\xi - \xi_\text{d}) = u_\text{d} - K\begin{bmatrix} x - x_\text{d} \\ z \end{bmatrix}, \tag{3.13}$$

where K is state feedback term for the augmented system (3.12).

Integral action can be included in the python-control `create_statefbk_iosystem` function using the `integral_action` keyword. The value of this keyword can either be an matrix (2D array) or a function. If a matrix C is specified, the difference between the desired state and system state will be multiplied by this matrix and integrated. The controller gain should then consist of a set of proportional gains K_p and integral gains K_i with

$$K = \begin{bmatrix} K_\text{p} \\ K_\text{i} \end{bmatrix}.$$

If `integral_action` is a function h, that function will be called with the signature `h(t, x, u, params)` to obtain the outputs whose error should be integrated. The number of output errors that are to be integrated must match the number of additional columns in the K matrix. If an estimator is specified, $\hat{x}$ will be used in place of x. Gain scheduling can also be used, as described in Section 2.4.

Singular extremals

The necessary conditions in the maximum principle enforce the constraints through the of the Lagrange multipliers $\lambda(t)$. In some instances, we can get an extremal curve that has one or more of the λ's identically equal to zero. This corresponds to a situation in which the constraint is satisfied strictly through the minimization of the cost function and does not need to be explicitly enforced. We illustrate this case through an example.

Example 3.6 Nonholonomic integrator

Consider the minimum time optimization problem for the nonholonomic integrator introduced in Example 2.2 with input constraints $|u_i| \leq 1$. The Hamiltonian for the system is given by

$$H = 1 + \lambda_1 u_1 + \lambda_2 u_2 + \lambda_3 x_2 u_1,$$

and the resulting equations for the Lagrange multipliers are

$$\dot{\lambda}_1 = 0, \qquad \dot{\lambda}_2 = \lambda_3 x_2, \qquad \dot{\lambda}_3 = 0. \tag{3.14}$$

It follows from these equations that λ_1 and λ_3 are constant. To find the input u corresponding to the extremal curves, we see from the Hamiltonian that

$$u_1 = -\operatorname{sgn}(\lambda_1 + \lambda_3 x_2 u_1), \qquad u_2 = -\operatorname{sgn} \lambda_2.$$

These equations are well-defined as long as the arguments of $\operatorname{sgn}(\cdot)$ are non-zero and we get switching of the inputs when the arguments pass through 0.

An example of an abnormal extremal is the optimal trajectory between $x_0 = (0, 0, 0)$ to $x_\text{f} = (\rho, 0, 0)$ where $\rho > 0$. The minimum time trajectory is clearly given by moving on a straight line with $u_1 = 1$ and $u_2 = 0$. This extremal satisfies the necessary conditions but with $\lambda_2 \equiv 0$, so that the "constraint" that $\dot{x}_2 = u_2$ is not strictly enforced through the Lagrange multipliers. ∇

3.8 Further Reading

There are a number of excellent books on optimal control. One of the first (and best) is the book by Pontryagin et al. [PBGM62]. During the 1960s and 1970s a number of additional books were written that provided many examples and served as standard textbooks in optimal control classes. Athans and Falb [AF06] and Bryson and Ho [BH75] are two such texts. A very elegant treatment of optimal control from the point of view of optimization over general linear spaces is given by Luenberger [Lue97]. A modern engineering textbook that contains a very compact and concise derivation of the key results in optimal control is the book by Lewis, Vrabie, and Syrmos [LVS12]. Finally, the online (and open access) notes by Daniel Liberzon [Lib10] provide detailed coverage of all of the topics in this chapter.

Exercises

3.1. (a) Let $G_1, G_2, \dots, G_k$ be a set of row vectors on $\mathbb{R}^n$. Let F be another row vector on $\mathbb{R}^n$ such that for every $x \in \mathbb{R}^n$ satisfying $G_i x = 0$, $i = 1, \dots, k$, we have $Fx = 0$. Show that there are constants $\lambda_1, \lambda_2, \dots, \lambda_k$ such that

$$F = \sum_{i=1}^{k} \lambda_i G_i.$$

(b) Let $x^* \in \mathbb{R}^n$ be an the extremal point (maximum or minimum) of a function f subject to the constraints $g_i(x) = 0$, $i = 1, \dots, k$. Define a new function

$$\tilde{f}(x, \lambda) = f(x) + \sum_{i=1}^{k} \lambda_i g_i(x).$$

Assuming that the gradients $\partial g_i(x^*)/\partial x$ are linearly independent, show that there are k scalars λ_i, $i = 1, \dots, k$ such that x^* is the (unconstrained) extremal of $\tilde{f}(x, \lambda)$.

3.2. Consider the following control system

$$\begin{aligned} \dot{q} &= u \\ \dot{Y} &= qu^{\mathsf{T}} - uq^{\mathsf{T}} \end{aligned}$$

where $u \in \mathbb{R}^m$ and $Y \in \mathbb{R}^{m \times m}$ is a skew symmetric matrix, $Y^{\mathsf{T}} = Y$.

(a) For the fixed end point problem, derive the form of the optimal controller minimizing the following integral

$$\frac{1}{2} \int_0^1 u^{\mathsf{T}} u \, dt.$$

(b) For the boundary conditions $q(0) = q(1) = 0$, $Y(0) = 0$ and

$$Y(1) = \begin{bmatrix} 0 & -y_3 & y_2 \\ y_3 & 0 & -y_1 \\ -y_2 & y_1 & 0 \end{bmatrix}$$

for some $y \in \mathbb{R}^3$, give an explicit formula for the optimal inputs u.

(c) (Optional) Find the input u to steer the system from $(0, 0)$ to $(0, \tilde{Y}) \in \mathbb{R}^m \times \mathbb{R}^{m \times m}$ where $\tilde{Y}^{\mathsf{T}} = -\tilde{Y}$.

(Hint: if you get stuck, there is a paper by Brockett on this problem.)

3.3. In this problem, you will use the maximum principle to show that the shortest path between two points is a straight line. We model the problem by constructing a control system

$$\dot{x} = u,$$

where $x \in \mathbb{R}^2$ is the position in the plane and $u \in \mathbb{R}^2$ is the velocity vector along the curve. Suppose we wish to find a curve of minimal length connecting $x(0) = x_0$ and $x(1) = x_f$. To minimize the length, we minimize the integral of the velocity along the curve,

$$J = \int_0^1 \|\dot{x}\| \, dt = \int_0^1 \sqrt{\dot{x}^{\mathsf{T}} \dot{x}} \, dt,$$

subject to to the initial and final state constraints. Use the maximum principle to show that the minimal length path is a straight line.

3.4. Consider the optimal control problem for the system

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

where $x = \mathbb{R}$ is a scalar state, $u \in \mathbb{R}$ is the input, the initial state $x(t_0)$ is given, and $a, b \in \mathbb{R}$ are positive constants. (Note that this system is not quite the same as the one in Example 3.2.) The cost function is given by

$$J = \tfrac{1}{2} \int_{t_0}^{t_\mathrm{f}} u^2(t) \, dt + \tfrac{1}{2} c x^2(t_\mathrm{f}),$$

where the terminal time t_f is given and c is a constant.

(a) Solve explicitly for the optimal control $u^*(t)$ and the corresponding state $x^*(t)$ in terms of t_0, t_f, $x(t_0)$ and t and describe what happens to the terminal state $x^*(t_\text{f})$ as $c \to \infty$.

Hint: Once you have $u^*(t)$, you can use the convolution equation to solve for the optimal state $x^*(t)$.

(b) Let $a = 1$, $b = 1$, $c = 1$, and $t_\text{f} - t_0 = 1$. Solve the optimal control problem numerically using python-control (or MATLAB) and compare it to your analytical solution by plotting the state and input trajectories for each solution. Take the initial conditions as $x(t_0) = 1$, 5, and 10.

(c) Suppose that we wish to have the final state be exactly zero. Change the optimization problem to impose a final constraint instead of a final cost. Solve the fixed endpoint, optimal control problem using python-control (or MATLAB), plot the state and input trajectories for the initial conditions in 0b, and compare the computation times for each approach

(d) Show that the system is differentially flat with appropriate choice of output(s) and compute the state and input as a function of the flat output(s).

(e) Using the polynomial basis $\{t^k,\, k = 0, \dots, M-1\}$ with an appropriate choice of M, solve for the (non-optimal) trajectory between $x(t_0)$ and $x(t_\text{f})$. Your answer should specify the explicit input $u_d(t)$ and state $x_d(t)$ in terms of t_0, t_f, $x(t_0)$, $x(t_\text{f})$ and t.

(f) Using the same parameters and initial conditions as in part 0b, compute a flatness-based trajectory between an initial condition x_0 and $x_\text{f} = 0$. Plot the state and input trajectories for each solution.

(g) Suppose that we choose more than the minimal number of basis functions for the differentially flat output. Show how to use the additional degrees of freedom to minimize the cost of the flat trajectory and demonstrate (numerically) that you can obtain a cost that is closer to the optimal.

3.5. Repeat Exercise 3.4 using the system

$$\dot{x} = -ax^3 + bu.$$

For part (a) you need only write the conditions for the optimal cost.

3.6. Consider the problem of moving a two-wheeled mobile robot (e.g., a Segway) from one position and orientation to another. The dynamics for the system is given by the nonlinear differential equation

$$\dot{x} = \cos\theta\, v, \qquad \dot{y} \; = \sin\theta\, v, \qquad \dot{\theta} = \omega,$$

where (x, y) is the position of the rear wheels, θ is the angle of the robot with respect to the x axis, v is the forward velocity of the robot and ω is spinning rate. We wish to choose an input (v, ω) that minimizes the time that it takes to move between two configurations (x_0, y_0, θ_0) and (x_f, y_f, θ_f), subject to input constraints $|v| \leq L$ and $|\omega| \leq M$.

Use the maximum principle to show that any optimal trajectory consists of segments in which the robot is traveling at maximum velocity in either the forward or reverse direction, and going either straight, hard left ($\omega = -M$) or hard right ($\omega = +M$).

Note: one of the cases is a bit tricky and cannot be completely proven with the tools we have learned so far. However, you should be able to show the other cases and verify that the tricky case is possible.

3.7. Consider a linear system with input u and output y and suppose we wish to minimize the quadratic cost function

$$J = \int_0^\infty \left(y^\mathsf{T} y + \rho u^\mathsf{T} u\right)\, dt.$$

Show that if the corresponding linear system is observable, then the closed loop system obtained by using the optimal feedback $u = -Kx$ is guaranteed to be stable.

3.8. Consider the system transfer function

$$H(s) = \frac{s+b}{s(s+a)}, \qquad a, b > 0$$

with state space representation

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 1 \\ 0 & -a \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \\ y &= \begin{bmatrix} b & 1 \end{bmatrix} x \end{aligned}$$

and performance criterion

$$V = \int_0^\infty (x_1^2 + u^2) dt.$$

(a) Let

$$P = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix},$$

with $p_{12} = p_{21}$ and $P > 0$ (positive definite). Write the steady state Riccati equation as a system of four explicit equations in terms of the elements of P and the constants a and b.

(b) Find the gains for the optimal controller assuming the full state is available for feedback.

3.9. Consider the optimal control problem for the system

$$\dot{x} = ax + bu \qquad J = \tfrac{1}{2} \int_{t_0}^{t_f} u^2(t)\, dt + \tfrac{1}{2} c x^2(t_f),$$

where $x \in \mathbb{R}$ is a scalar state, $u \in \mathbb{R}$ is the input, the initial state $x(t_0)$ is given, and $a, b \in \mathbb{R}$ are positive constants. We take the terminal time t_f as given and let $c > 0$ be a constant that balances the final value of the state with the input required to get to that position. The optimal trajectory is derived in Example 3.2.

Now consider the infinite horizon cost

$$J = \tfrac{1}{2} \int_{t_0}^{\infty} u^2(t)\, dt$$

with $x(t)$ at $t = \infty$ constrained to be zero.

(a) Solve for $u^*(t) = -bPx^*(t)$ where P is the positive solution corresponding to the algebraic Riccati equation. Note that this gives an explicit feedback law ($u = -bPx$).

3.10. Consider a linear system of the form

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 2 \\ -1 & -0.1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} (u + d)$$

where d is a disturbance input.

(a) Design an LQR controller for the system that regulates the system to a desired equilibrium point $x_\text{d} = (1, 0)$, assuming no disturbance ($d = 0$). Plot the response of the system starting from an initial condition $x(0) = (0, 0)$ and show that the system response converges to the desired equilibrium point.

(b) Assume now that $d = 0.1$. Show that the system response with your LQR controller from part (a) no longer converges to the desired equilibrium point and construct a controller using integral action that recovers the ability to drive the system state to that point.

(c) Suppose that there is uncertainty in the input matrix, so that the B matrix becomes

$$B = \begin{bmatrix} 0 \\ \gamma \end{bmatrix}, \qquad 0.5 \leq \gamma \leq 2.$$

Show that the uncertain system response with your original LQR controller no longer converges to the desired equilibrium (even with $d = 0$), but that the controller with integral action still causes the system to converge to the desired equilibrium. (OK to just show this for $\gamma = 0.5$ and 2.)

3.11. Consider the lateral control problem for an autonomous ground vehicle from Example 2.1. We assume that we are given a reference trajectory $r = (x_\text{d}, y_\text{d})$ corresponding to the desired trajectory of the vehicle. For simplicity, we will assume that we wish to follow a straight line in the x direction at a constant velocity $v_\text{d} > 0$ and hence we focus on the y and θ dynamics:

$$\dot{y} = \sin\theta\, v_\text{d}, \qquad \dot{\theta} = \frac{1}{l} \tan\delta\, v_\text{d}.$$

We let $v_\text{d} = 10$ m/s and $l = 2$ m.

(a) Design an LQR controller that stabilizes the position y to $y_\text{d} = 0$. Plot the step and frequency response for your controller and determine the overshoot, rise time, bandwidth and phase margin for your design. (Hint: for the frequency domain specifications, break the loop just before the process dynamics and use the resulting SISO loop transfer function.)

(b) Suppose now that $y_\mathrm{d}(t)$ is not identically zero, but is instead given by $y_\mathrm{d}(t) = r(t)$. Modify your control law so that you track $r(t)$ and demonstrate the performance of your controller on a "slalom course" given by a sinusoidal trajectory with magnitude 1 meter and frequency 1 Hz.

3.12. Consider the dynamics of the vectored thrust aircraft described in Examples 2.4 and 3.5. The equations of motion are given by

$$\begin{aligned} m\ddot{x} &= F_1 \cos\theta - F_2 \sin\theta - c\dot{x}, \\ m\ddot{y} &= F_1 \sin\theta + F_2 \cos\theta - c\dot{y} - mg, \\ J\ddot{\theta} &= rF_1. \end{aligned} \tag{3.15}$$

with parameter values

$$m = 4 \text{ kg}, \quad J = 0.0475 \text{ kg m}^2, \quad r = 0.25 \text{ m}, \quad g = 9.8 \text{ m/s}^2, \quad c = 0.05 \text{ Ns/m},$$

which corresponds roughly to the values for the Caltech ducted fan flight control testbed.

We wish to generate an optimal trajectory for the system that corresponds to moving the system for an initial hovering position to a hovering position one meter to the right ($x_\mathrm{f} = x_0 + 1$).

For each of the parts below, you should solve for the optimal input, simulate the (open loop) system, and plot the xy trajectory of the system, along with the angle θ and inputs F_1 and F_2 over the time interval. In addition, create a time and record the following information for each approach:

- the computation time required;
- the final position (for the open loop system);
- the weighted integrated cost of the input along the trajectory:

$$\int_0^T \bigl(10F_1^2(\tau) + (F_2 - mg)^2(\tau)\bigr)\, d\tau. \tag{3.16}$$

(a) Solve for an optimal trajectory using a quadratic cost from the final point with weights

$$Q_x = \mathrm{diag}([1, 1, 10, 0, 0, 0]), \qquad Q_u = \mathrm{diag}([10, 1]).$$

This cost function attempts to minimize the angular deviation θ and the sideways force F_1.

(b) Re-solve the problem using Bezier curves as the basis functions for the inputs. This should give you smoother inputs and a nicer response.

(c) Re-solve the problem using a terminal cost $V(x(T)) = x(T)^\mathsf{T} P_1 x(T)$ to try to get the system closer to the final value. You should try adjusting the cost along the trajectory Q_x versus the terminal cost P_1 to minimize the weighted integrated cost (3.16).

(d) Re-solve the problem using a terminal *constraint* to try to get the system closer to the final value. Adjust the cost along the trajectory to try to minimize the cost in equation (3.16). (Hint: you may have to use an initial guess to get the optimization to converge.)

(e) If $c = 0$, it can be shown that this system is differentially flat (see Example 2.4). Setting $c = 0$, re-solve the optimization problem using differential flatness. (The flatness mappings can be found in the file `pvtol.py`, available on the companion website.)

Chapter 4

Receding Horizon Control

This chapter builds on the previous two chapters and explores the use of online optimization as a tool for control of nonlinear systems. We begin with a discussion of the technique of receding horizon control (RHC), which builds on the ideas of trajectory generation and optimization. We focus on a particular form of receding horizon control that makes use of a control Lyapunov function as a terminal cost, for which there are good stability and performance properties, and include a (optional) proof of stability. Methods for implementing receding horizon control, making use of numerical optimization possibly combined with differential flatness, are also provided. We conclude the chapter with a detailed design example, in which we explore some of the computational tradeoffs in optimization-based control as applied to a flight control experiment.

Prerequisites. Readers should be familiar with the concepts of trajectory generation and optimal control as described in Chapters 2 and 3 in this supplement. For the proof of stability for the receding horizon controller that we present, familiarity with Lyapunov stability analysis at the level given in FBS2e, Chapter 5 is assumed (but this material can be skipped if the reader is not familiar with Lyapunov stability analysis).

The material in this chapter is based in part on joint work with John Hauser, Ali Jadbabaie, Mark Milam, Nicolas Petit, William Dunbar, and Ryan Franz [MHJ$^+$03].

4.1 Overview

The use of real-time trajectory generation techniques enables a sophisticated approach to the design of control systems, especially those in which constraints must be taken into account. The ability to compute feasible trajectories quickly enables us to make use of online computation of trajectories as an "outer feedback" loop that can be used to take into account nonlinear dynamics, input constraints, and more complex descriptions of performance goals.

Figure 4.1, a version of which was shown already in Chapter 2, provides a high level view of how real-time trajectory generation can be utilized. The dashed line from the output of the process to the trajectory generation block represents the use

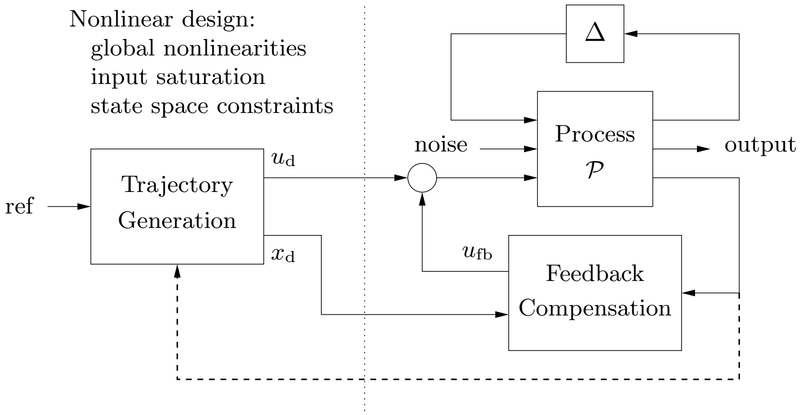


Figure 4.1: Two degree-of-freedom controller design for a process P with uncertainty Δ. The controller consists of a trajectory generator and feedback controller. The trajectory generation subsystem computes a feedforward command $u_{\rm d}$ along with the desired state $x_{\rm d}$. The state feedback 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.

of "on-the-fly" computation of the trajectory based on the current outputs of the process. This dynamically generated trajectory is then fed to the more traditional feedback controller. This same type of structure can also be seen in Figure 1.6, where the trajectory generation "layer" can make use of current measurements of the environment, as well as an online model of the process and upper level (supervisory controller) commands for the task to be accomplished.

The approach that we explore in this chapter is to make use of *receding horizon control*: a (optimal) feasible trajectory is computed from the current state to the desired state over a finite time horizon T, used for a short period of time $\Delta T < T$, and then recomputed based on the new system state starting at time $t + \Delta T$ until time $t+T+\Delta T$, as shown in Figure 4.2. As in the case of trajectory generation, we will normally compute the optimal trajectory assuming no process disturbances d, sensor noise n, or uncertainty Δ, relying on the feedback controller to compensate for those effects.

For the techniques that we will consider here, the problem that we solve at each time step t_i is a constrained, optimal trajectory generation problem of the form

$$\begin{aligned} u_{[t_i,t_i+\Delta T]} = \arg\min_{(x,u)} \int_{t_i}^{t_i+T} & L(x,u)\,d\tau + V(x(t_i+T)) \\ \text{subject to} & \\ & x(t_i) = \text{current state} \\ & \dot{x} = f(x,u) \\ & g_j(x,u) \le 0, \qquad j = 1,\dots,r, \\ & \psi_k(x(t_i+T)) = 0, \quad k = 1,\dots,q. \end{aligned} \tag{4.1}$$

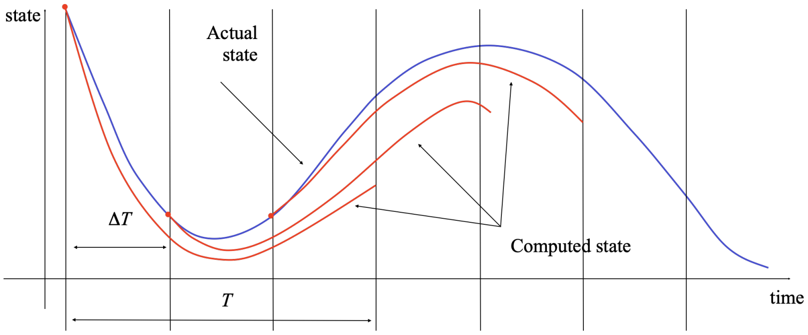


Figure 4.2: Receding horizon control. Every ΔT seconds, a trajectory generation problem is solved over a T second horizon, starting from the current state. In reality, the system will not follow the predicted path exactly, so that the red (computed) and blue (actual) trajectories will diverge. We then recompute the optimal path from the new state at time $t + \Delta T$, extending our horizon by an additional ΔT units of time.

We note that the cost consists of a trajectory cost $L(x, u)$ as well as an end-of-horizon (terminal) cost $V(x(t + T))$. In addition, we allow for the possibility of trajectory constraints on the states and inputs given by a set of functions $g_j(x, u)$ and a set of terminal constraints given by $\psi_k(x(t + T))$.

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 Exercise 4.4 for an example). To address this problem, the terminal cost $V(x(t + T))$ and/or the terminal constraints $\psi_k(x(t + T))$ must have certain properties to ensure stability (see [MRRS00] for details). In this chapter we focus on the use of terminal costs since these have certain advantages in terms of the underlying optimization problem to be solved.

Development and application of receding horizon control (also called model predictive control, or MPC) originated in process control industries where the processes being controlled are often sufficiently slow to permit its implementation with only modest computational resources. The rapid advances in computation over the last several decades have enabled receding horizon control to be used in many new applications, and they are especially prevalent in autonomous vehicles, where the trajectory generation layer is particularly important for achieving safe operation in complex environments. Proper formulation of the problem to enable rapid computation is still often required, and the use of differential flatness and other techniques (such as motion primitives) can be very important.

Finally, we note that it is often the case that the trajectory generation problems to be solved may have non-unique or non-smooth solutions, and that these can often be very sensitive to small changes in the inputs to the algorithm. Full implementation of receding horizon control techniques thus often requires careful attention to details in how the problems are solved and the use of additional methods to ensure that good solutions are obtained in specific application scenarios. Despite all of these warnings, receding horizon control is one of the dominant methods used for control of nonlinear systems and one of the few methods that works in the presence of input (and state) constraints, leading to its wide popularity.

4.2 Receding Horizon Control with Terminal Cost

One of the earliest techniques for ensuring stability of the closed loop system under receding horizon control was to impose a terminal constraint on the optimization through use of the function $\psi(x(t+T))$ in equation (4.1). While this technique can be shown to be sound from a theoretical point of view, it can be very difficult to satisfy from a computational point of view, since imposing equality constraints on the terminal state of a trajectory generation problem can be computationally expensive. In this section we explore an alternative formulation, making use of an appropriate terminal cost function.

Stability of Receding Horizon Control using Terminal Costs

We consider the following special case of receding horizon control problem in equation (4.1):

$$\begin{aligned} \dot{x} &= f(x,u), \qquad x(0) \text{ given}, u \in \mathcal{U} \subset \mathbb{R}^m \\ u_{[t,t+\Delta T]} &= \arg\min_{(x,u)} \int_t^{t+T} L(x,u)\,d\tau + V(x(t+T)). \end{aligned} \tag{4.2}$$

The main differences between equation (4.2) and equation (4.1) are that we only consider constraints on the input ($u \in \mathcal{U}$) and we do not impose any terminal constraints.

Stability of this system is not guaranteed in general, but if we choose the trajectory and terminal cost functions carefully, it is possible to provide stability guarantees.

To illustrate how the choice of the terminal condition can provide stability, consider the case where we have an infinite horizon optimal control problem. If we start at state x at time t, we can seek to minimize the "cost to go" function

$$J(t,x) = \int_t^\infty L(x,u)\,dt.$$

If we let $J^*(x,t)$ represent the optimal cost to go function, then a natural choice for the terminal cost is $V(x(t+T)) = J^*(x(t+T),T)$, since then the optimal finite and infinite horizon costs are the same:

$$\begin{aligned} \min \int_t^\infty L(x,u)\,d\tau &= \int_t^{t+T} L(x^*,u^*)\,dt + \int_{t+T}^\infty L(x^*,u^*)\,dt \\ &= \int_t^{t+T} L(x^*,u^*)\,dt + \underbrace{J^*(t+T,x^*)}_{V(x^*(t+T))}. \end{aligned}$$

Intuitively, if we solve the infinite horizon problem at each update ΔT and ΔT is sufficiently small, then we anticipate that the system trajectory should converge to the optimal trajectory, and hence to the origin (otherwise the cost would not converge to a finite value, assuming $Q_x > 0$).

Of course, if the optimal value function were available there would be no need to solve a trajectory optimization problem, since we could just use the gradient of the cost function to choose the input u. Only in special cases (such as the linear quadratic regulator problem) can the infinite horizon optimal control problem be solved in closed form, and we thus seek to find a simpler set of conditions under which we can guarantee stability of closed loop, receding horizon controller. The following theorem summarizes on such set of conditions:

Theorem 4.1 (based on [JYH01])**.** *Consider the receding horizon control problem in equation* (4.2) *and suppose that the trajectory cost* $L(x, u)$ *and terminal cost* $V(x(t + T))$ *satisfy*

$$\min_{u \in \mathcal{U}} \left(\frac{\partial V}{\partial x} f(x, u) + L(x, u) \right) \leq 0 \tag{4.3}$$

for all x *in a neighborhood of the origin. Then, for* every $T > 0$ *and* $\Delta T \in (0, T]$*, there exist constants* $M > 0$ *and* $c > 0$ *such that the resulting receding horizon trajectories converge to 0 exponentially fast:*

$$\|x(t)\| \leq M e^{-ct} \|x(0)\|.$$

Before providing more insights into when these conditions can be satisfied, it is useful to take think about the implications of Theorem 4.1. In particular, it provides us a method for defining a stabilizing feedback controller for a fully nonlinear system in the presence of input constraints. This latter feature is particularly important, since it turns out that input constraints are ubiquitous in control systems and must other methods, including LQR and gain scheduling, are not able to take them into account in a systematic and rigorous way. It is because of this ability to handle constraints that receding horizon control is so widely used.

Control Lyapunov Functions

To provide insight into the conditions in Theorem 4.1, we need to define the concept of a *control Lyapunov function.* The material in this subsection is rather advanced in nature and can be skipped on first reading. Readers should be familiar with (regular) Lyapunov stability analysis at the level given in FBS2e, Chapter 5 prior to tackling the concepts in this section.

Control Lyapunov functions are an extension of standard Lyapunov functions and were originally introduced by Sontag [Son83]. They allow constructive design of nonlinear controllers and the Lyapunov function that proves their stability. We give a brief description of the basic idea of control Lyapunov functions here; a more complete treatment is given in [KKK95].

Consider a nonlinear control system

$$\dot{x} = f(x, u), \qquad x \in \mathbb{R}^n,\ u \in \mathbb{R}^m, \tag{4.4}$$

and recall that a function $V(x)$ is a *positive definite function* if $V(x) \geq 0$ for all $x \in \mathbb{R}^n$ and $V(x) = 0$ if and only if $x = 0$. A function $V(x)$ is *locally positive definite* if it is positive definite on a ball of radius ϵ around the origin, $B_\epsilon(0) = \{x \colon \|x\| < \epsilon\}$.

Definition 4.1 (Control Lyapunov Function)**.** A locally positive function $V\colon \mathbb{R}^n \to \mathbb{R}_+$ is called a *control Lyapunov function (CLF)* for a control system (4.4) if

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

Intuitively, a control Lyapunov function is a positive definite function for which it is always possible to choose an input u that makes the function decrease if we apply that input to the control system (4.4). Since the function V is positive definite, if we always choose a u that makes it decrease then eventually the value of the Lyapunov function must converge to 0 and hence the state $x(t)$ must also converge to zero. It turns out that this property is enough to show that the system is *stabilizable* using continuous (though not necessarily linear) feedback laws of the form $u = -k(x)$.

In general, it is difficult to find a control Lyapunov function for a given system. However, for many classes of systems, there are specialized methods that can be used. One of the simplest is to use the Jacobian linearization of the system around the desired equilibrium point and generate a control Lyapunov function by solving an LQR problem. To see how this works, we consider first the case of a linear system with quadratic cost function.

As described in Chapter 3, the problem of minimizing the quadratic performance index

$$J = \int_0^\infty (x^\mathsf{T}(t) Q_x x(t) + u^\mathsf{T}(t) Q_u u(t))\, dt \qquad \text{subject to} \qquad \begin{aligned} \dot x &= Ax + Bu,\\ x(0) &= x_0, \end{aligned} \tag{4.5}$$

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

$$A^\mathsf{T} P + PA - PBR^{-1}B^\mathsf{T} P + Q = 0. \tag{4.6}$$

The optimal control action is given by

$$u^* = -R^{-1}B^\mathsf{T} P x$$

and $V = x^\mathsf{T} P x$ is a control Lyapunov function for the system since it can be shown (with a bit of algebra) that

$$\min_u \frac{\partial V}{\partial x} f(x,u) \le \frac{\partial V}{\partial x} f(x,u^*) = -x^\mathsf{T}(Q_x + PBQ_u^{-1}B^\mathsf{T}P)x \le 0.$$

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

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

where the pairs (A,B) and $(Q_x^{\frac{1}{2}}, A)$ are assumed to be stabilizable and detectable respectively. The control Lyapunov function $V(x) = x^\mathsf{T} P x$ is valid in a region around the equilibrium $(0,0)$, as shown in Exercise 4.1.

More complicated methods for finding control Lyapunov functions are often required and many techniques have been developed. An overview of some of these methods can be found in [Jad01].

Solving Receding Horizon Optimal Control Problems

We now return to the problem of implementing the receding horizon controller in equation (4.3). As illustrated in Figure 4.2, at every time instant t_i we compute the optimal trajectory that minimizes the cost function

$$J(x,t) = \int_t^{t+T} L(x,u)\,d\tau + V(x(t+T))$$

subject to the satisfying the equations of motion with constrained inputs:

$$\dot{x} = f(x,u), \qquad u \in \mathcal{U} \subset \mathbb{R}^m.$$

This is precisely the optimal control problem that we considered in Chapter 3 and so the numerical methods used in that chapter can be utilized.

One conceptually simple way to implement the optimization required to solve this optimal control problem is to parameterize the inputs of the system u, either by setting the values of u at discrete time points or by choosing a set of basis functions for u and searching over linear combinations of the basis functions. These methods are often referred to as "shooting" methods, since they integrate ("shoot") the equations forward in time and then attempt to compute the changes in parameter values to allow the system to minimize the cost and satisfy any constraints. While crude, this approach does work for simple systems and can be used to gain insights into the properties of a receding horizon controller based on simulations (where real-time computation is not needed).

Example 4.1 Double integrator with bounded input
To illustrate the implementation of a receding horizon controller, we consider a linear system corresponding to a double integrator with bounded input:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \operatorname{clip}(u) \qquad \text{where} \qquad \operatorname{clip}(u) = \begin{cases} -1 & u < -1, \\ u & -1 \leq u \leq 1, \\ 1 & u > 1. \end{cases}$$

We implement a model predictive controller by choosing

$$Q_x = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad Q_u = \begin{bmatrix} 1 \end{bmatrix}, \qquad P_1 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}.$$

Figure 4.3 shows the results of this computation, with the inputs plotted for the planning horizon, showing that the final computed input differs from the planned inputs over the horizon. (The code for computing these solutions is given in Section 4.3.) ∇

Implementing receding horizon control for realistic problems requires care in formulating the optimization optimization so that it can be done in real-time. For example, for a typical autonomous vehicle (land, air, or sea), a reasonable optimization horizon might be the next 10-60 s and a typical update period might be as short as 10-100 ms. There are a variety of methods for speeding up computations, as well as taking into account finite computation times. Some of these are

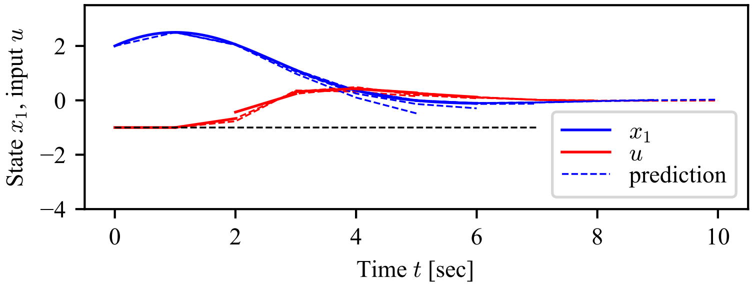


Figure 4.3: Receding horizon controller for a double integrator. Dashed lines show the planned trajectory over each horizon; solid lines show the closed loop trajectory. The horizontal dashed line on each plot shows the lower limit of the input.

described in more detail in Section 4.6, where we describe a specific implementation of receding horizon control on a flight control testbed.

A relatively efficient numerical approach to solving the optimal control problem is the direct collocation method (see [Kel17] for a good overview). The idea behind this approach is to transform the optimal control problem into a constrained nonlinear programming problem. This is accomplished by discretizing time into a set of $N-1$ intervals defined by grid points

$$t_0 = t_1 < t_2 < \ldots < t_N = t_\text{f}$$

and approximating the state x and the control input u as piecewise polynomials $\tilde{x}$ and $\tilde{u}$. For example, on each interval the states can be approximated by a cubic polynomial and the control can be approximated by a linear polynomial. The value of the polynomial at the midpoint of each interval is then used to satisfy the dynamics

$$\dot{x} = f(x, u).$$

To solve the problem, let $\tilde{x}(x(t_1), \ldots, x(t_N))$ and $\tilde{u}(u(t_1), \ldots, u(t_N))$ denote the approximations to x and u, which depend on $(x(t_1), \ldots, x(t_N)) \in \mathbb{R}^{nN}$ and $(u(t_1), \ldots, u(t_N)) \in \mathbb{R}^N$, representing the value of x and u at the grid points. We then solve the following finite dimension approximation of the original control problem (4.1):

$$\min_{\xi \in \mathbb{R}^M} F(\xi) = J(\tilde{x}(\xi), \tilde{u}(\xi))$$

$$\text{subject to} \quad \begin{cases} \dot{\tilde{x}} - f(\tilde{x}(\xi, \tilde{u}(\xi) = 0, \\ lb \leq c(\tilde{x}(\xi), \tilde{u}(\xi)) \leq ub, \\ \qquad \forall t = \dfrac{t_j + t_{j+1}}{2} \quad j = 1, \ldots, N-1 \end{cases} \tag{4.7}$$

where $\xi = (x(t_1), u(t_1), \ldots, x(t_N), u(t_N))$, and $M = \dim \xi = (n+1)N$. Note here that we are optimizing over both the x and u variables at the grid points and then seeking to (approximately) satisfying the differential equation, compared to the shooting method, which optimizes just over the inputs at the grid points and then integrates the differential equation.

Collocation techniques turn out to be much more numerically well-conditioned than shooting methods, and there are excellent software packages available for solving collocation-based optimization problems.

Proof of Stability (with J. E. Hauser and A. Jadbabaie)

In this final (optional) subsection, we return to Theorem 4.1 and provide a mathematically rigorous version of the theorem and a sketch of its proof. In order to show the stability of the proposed approach, and give full conditions on the terminal cost $V(x(T))$, we briefly review the problem of optimal control over a finite time horizon as presented in Chapter 3 to establish some notation and set some more specific conditions required for receding horizon control. This material is based on [MHJ$^+$03].

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

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

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

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

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

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

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

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

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

To ensure that the solutions of the optimization problems of interest are well behaved, we impose some convexity conditions. We require the set $f(x, \mathbb{R}^m) \subset \mathbb{R}^n$ to be convex for each $x \in \mathbb{R}^n$. Letting $\lambda \in \mathbb{R}^n$ represent the co-state, we also require that the pre-Hamiltonian function $\lambda^{\mathsf{T}} f(x, u) + L(x, u) =: K(x, u, \lambda)$ be strictly convex for each $(x, \lambda) \in \mathbb{R}^n \times \mathbb{R}^n$ and that there is a C^2 function $\bar{u}^*\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$ providing the global minimum of $K(x, u, \lambda)$. The Hamiltonian $H(x, \lambda) := K(x, \bar{u}^*(x, \lambda), \lambda)$ is then C^2, ensuring that extremal state, co-state, *and* control trajectories will all be sufficiently smooth (C^1 or better). Note that these conditions are automatically satisfied for control affine f and quadratic L.

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

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

The optimal cost (from x) is given by

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

where the control function $u(\cdot)$ belongs to some reasonable class of admissible controls (e.g., piecewise continuous). The function $J^*_\infty(x)$ is often called the *optimal value function* for the infinite horizon optimal control problem. For the class of f and L considered, it can be verified that $J^*_\infty(\cdot)$ is a positive definite C^2 function in a neighborhood of the origin [HO01].

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

$$J_T(x,u(\cdot)) = \int_0^T L(x^u(\tau;x),u(\tau))\,d\tau + V(x^u(T;x)), \tag{4.8}$$

and denote the optimal cost (from x) as

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

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

Let Γ^∞ denote the domain of $J^*_\infty(\cdot)$ (the subset of $\mathbb{R}^n$ on which J^*_∞ is finite). It is not too difficult to show that the cost functions $J^*_\infty(\cdot)$ and $J^*_T(\cdot)$, $T \geq 0$, are continuous functions on Γ_∞ [Jad01]. For simplicity, we will allow $J^*_\infty(\cdot)$ to take values in the extended real line so that, for instance, $J^*_\infty(x) = +\infty$ means that there is no control taking x to the origin.

We will assume that f and L are such that the minimum value of the cost functions $J^*_\infty(x)$, $J^*_T(x)$, $T \geq 0$, is attained for each (suitable) x. That is, given x and $T > 0$ (including $T = \infty$ when $x \in \Gamma^\infty$), there is a (C^1 in t) optimal trajectory $(x^*_T(t;x), u^*_T(t;x))$, $t \in [0,T]$, such that $J_T(x, u^*_T(\cdot;x)) = J^*_T(x)$. For instance, if f is such that its trajectories can be bounded on finite intervals as a function of its input size, e.g., there is a continuous function β such that $\|x^u(t;x_0)\| \leq \beta(\|x_0\|, \|u(\cdot)\|_{L_1[0,t]})$, then (together with the conditions above) there will be a minimizing control (cf. [LM67]). Many such conditions may be used to good effect; see [Jad01] for a more complete discussion.

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

$$\Gamma^\infty_r := \{x \in \Gamma^\infty : J^*_\infty(x) \leq r^2\}$$

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

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

and

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

These results provide the technical framework needed for receding horizon control. The following restated version of Theorem 4.1 provides a more rigorous description of the results of this section.

Theorem 1'. [JYH01] *Consider the receding horizon control problem in equation* (4.2) *and suppose that the terminal cost $V(\cdot)$ is a control Lyapunov function such that*

$$\min_{u \in \mathbb{R}^m} (\dot{V} + L)(x, u) \leq 0 \tag{4.9}$$

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

Theorem 1' shows that for *any* horizon length $T > 0$ and *any* sampling time $\delta \in (0, T]$, the receding horizon scheme is exponentially stabilizing over the set $\Gamma^\mathsf{T}_{r_v}$. For a given T, the region of attraction estimate is enlarged by increasing r beyond r_v to $\bar{r}(T)$ according to the requirement that $V(x_T^*(T; x)) \leq r_v^2$ on that set. An important feature of the above result is that, for operations with the set $\Gamma^\mathsf{T}_{\bar{r}(T)}$, there is no need to impose stability ensuring constraints which would likely make the online optimizations more difficult and time consuming to solve.

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

$$J_T^*\big(x_T^*(\delta; x)\big) \leq J_T^*(x) - \int_0^\delta q\big(L(x_T^*(\tau; x), u_T^*(\tau; x)\big)\, d\tau. \tag{4.10}$$

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

To show equation (4.10) holds, consider a trajectory in which we apply the optimal control for the first T seconds and then apply a closed loop controller using a stabilizing feedback $u = -k(x)$ for another T seconds. (The stabilizing compensator is guaranteed to exist since V is a control Lyapunov function.) Let $(x^*_T, u^*_T)(t; x)$, $t \in [0, T]$ represent the optimal control and $(x^k, u^k)(t - T; x^*_T(T; x))$, $t \in [T, 2T]$ represent the control with $u = -k(x)$ applied where k satisfies $(\dot V + L)(x, -k(x)) \leq 0$. Finally, let $(\tilde x(t), \tilde u(t))$, $t \in [0, 2T]$ represent the trajectory obtained by concatenating the optimal trajectory (x^*_T, u^*_T) with the control Lyapunov function trajectory (x^k, u^k).

We now proceed to show that the inequality (4.10) holds. The cost of using $\tilde u(\cdot)$ for the first T seconds starting from the initial state $x^*_T(\delta; x))$, $\delta \in [0, , T]$ is given by

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

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

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

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

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

Finally, using the optimality of u^*_T we have that $J^*_T(x^*_T(\delta; x)) \leq J_T(x^*_T(\delta; x), \tilde u(\cdot))$ and we obtain equation (4.10). □

An important benefit of receding horizon control is its ability to handle state and control constraints. While the above theorem provides stability guarantees when there are no constraints present, it can be modified to include constraints on states and controls as well. In order to ensure stability when state and control constraints are present, the terminal cost $V(\cdot)$ should be a local control Lyapunov function satisfying $\min_{u \in \mathcal{U}} \dot V + L(x, u) \leq 0$ where $\mathcal{U}$ is the set of controls where the control constraints are satisfied. Moreover, one should also require that the resulting state trajectory $x^{\mathrm{CLF}}(\cdot) \in \mathcal{X}$, where $\mathcal{X}$ is the set of states where the constraints are satisfied. (Both $\mathcal{X}$ and $\mathcal{U}$ are assumed to be compact with origin in their interior). Of course, the set Ω_{r_v} will end up being smaller than before, resulting in a decrease in the size of the guaranteed region of operation (see [MRRS00] for more details).

4.3 Implementation in Python

The optimal control module of the python-control package allows implementation of receding horizon control by solving an optimization problem based on the current state of a nonlinear system. We begin by defining an optimal control problem using the `OptimalControlProblem` class:

```
ocp = obc.OptimalControlProblem(
    sys, timepts, cost, constraints, terminal_cost)
```

To describe an optimal control problem we need an input/output system, a list of time points, a cost function, and (optionally) a set of constraints on the state and/or input, either along the trajectory (via the `trajectory_constraint` keyword) or at the terminal time (via the `terminal_constraint` keyword). The `OptimalControlProblem` class sets up an optimization over the inputs at each point in time, using the integral and terminal costs as well as the trajectory and terminal constraints.

Once an optimal control problem has been defined, the `compute_trajectory` method can be used to solve for an optimal trajectory from a given state x:

```
res = ocp.compute_trajectory(x)
t, u = res.time, res.inputs
```

This is the method that the `opt.solve_ocp` function uses to compute an optimal trajectory. In the context of model predictive control, we would repeatedly call `compute_trajectory` from the state at each update time t_i and then apply the input u for the next ΔT seconds.

For discrete time systems, the `create_mpc_iosystem` method can be used to create an input/output system that implements the control law:

```
ctrl = ocp.create_mpc_iosystem()
```

The resulting object takes as input the current state of the system as returns as output the commanded input from the MPC controller.

For continuous time system, receding horizon control must be implemented "manually", by computing the optimal input at time instant t_i and then simulating the system over the interval $[t_i, t_i{+}\Delta T]$. This is illustrated in the following example.

Example 4.2 Double integrator with bounded input
We the consider linear system corresponding to a double integrator with bounded input described in Example 4.1. The equations of motion are given by

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathrm{clip}(u) \qquad \text{where} \qquad \mathrm{clip}(u) = \begin{cases} -1 & u < -1, \\ u & -1 \leq u \leq 1, \\ 1 & u > 1. \end{cases}$$

This function can be created with the Python code

```
def doubleint_update(t, x, u, params):
    return np.array([x[1], np.clip(u, -1, 1)])

proc = ct.NonlinearIOSystem(
    doubleint_update, None, name="double integrator",
    inputs = ['u'], outputs=['x[0]', 'x[1]'], states=2, dt=True)
```

We now define an optimal control problem with quadratic costs and input constraints:

```
# Cost function
Qx = np.diag([1, 0])         # state cost
Qu = np.diag([1])            # input cost
P1 = np.diag([0.1, 0.1])     # terminal cost

# Constraints (using input_poly_constraint to illustrate its use)
traj_constraints = opt.input_poly_constraint(
    proc, np.array([[1, 0], [-1, 0]]), np.array([1, 1]))

# Horizon
T = 5
timepts = np.linspace(0, T, 5, endpoint=True)

# Set up the optimal control problem
ocp = opt.OptimalControlProblem(
    proc, timepts,
    opt.quadratic_cost(proc, Qx, Qu),
    trajectory_constraints=traj_constraints,
    terminal_cost=opt.quadratic_cost(proc, P1, None)
)
```

This optimal control problem contains all of the information required to compute the optimal input from a state x over the specified time horizon.

To use this optimal control problem in a receding horizon fashion, we manually compute the trajectory at each time point:

```
x = X0                       # initial condition (updated as we go)
Tf = 10                      # total simulation time
for t in np.linspace(0, Tf-T, 6, endpoint=True):
    # Compute the optimal trajectory over the horizon
    res = ocp.compute_trajectory(x, return_states=True)

    # Simulate the system for the update period
    time = np.linspace(0, res.time[1], 20)
    soln = ct.input_output_response(proc, time, inputs, x)

    # Update the state for the next iteration
    x = soln.states[:, -1]
```

Figure 4.3 in the previous section shows the results of this computation, with the inputs plotted for the planning horizon, showing that the final computed input differs from the planned inputs over the horizon. ∇

4.4 Receding Horizon Control Using Differential Flatness

For systems that are differentially flat, it is possible to use the flatness-based structure of the system to implement receding horizon control. In this section we summarize how to exploit differential flatness to find fast numerical algorithms for solving the optimal control problems required for the receding horizon control results of the previous section.

We consider the affine nonlinear control system

$$\dot{x} = f(x) + g(x)u, \tag{4.11}$$

where all vector fields and functions are smooth. For simplicity, we focus on the single input case, $u \in \mathbb{R}$. We wish to find a trajectory of equation (4.11) that minimizes the performance index (4.8), subject to a vector of initial, final, and trajectory constraints

$$\begin{aligned} &lb_0 \le \psi_0(x(t_0), u(t_0)) \le ub_0, \\ &lb_\mathrm{f} \le \psi_\mathrm{f}(x(t_\mathrm{f}), u(t_\mathrm{f})) \le ub_\mathrm{f}, \\ &lb_t \le S(x, u) \le ub_t, \end{aligned} \tag{4.12}$$

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

$$\min_{(x,u)} J(x,u) \qquad \text{subject to} \qquad \begin{cases} \dot{x} = f(x) + g(x)u, \\ lb \le c(x,u) \le ub. \end{cases} \tag{4.13}$$

For a system that is differentially flat, we can compute feasible trajectories in terms of the flat output trajectory $z(\cdot)$, as described in Section 2.3. When the parameterization is only partial, the dimension of the subspace spanned by the output and its derivatives is given by r the *relative degree* of this output [Isi89]. In this case, it is possible to write the system dynamics as

$$\begin{aligned} &x = \alpha(z, \dot{z}, \ldots, z^{(q)}), \\ &u = \beta(z, \dot{z}, \ldots, z^{(q)}), \\ &\Phi(z, \dot{z}, \ldots, z^{n-r}) = 0, \end{aligned} \tag{4.14}$$

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

A computationally attractive representation for the flat outputs is to use B-splines, which are a set of piecewise polynomials defined across a set of knot points with given degree and smoothness. On each interval between two knot points, we have a polynomial of a given degree and the spline is continuous up to a given smoothness at interior breakpoints. B-splines are chosen as basis functions because of their ease of enforcing continuity across knot points and ease of computing their derivatives. A pictorial representation of such an approximation is given in Figure 4.4. In this formulation, each flat output z_j is represented as

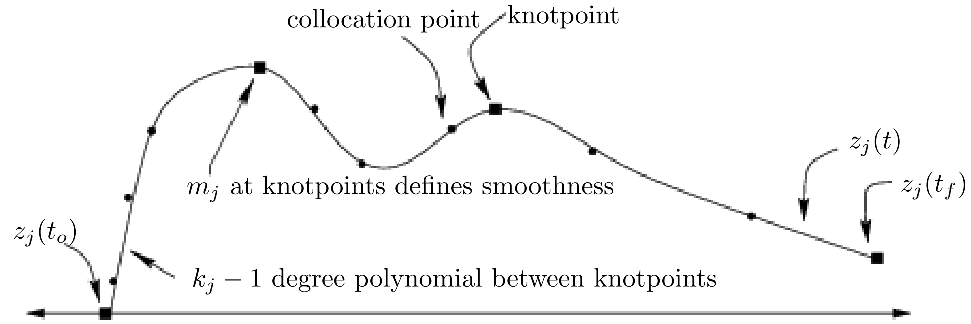

collocation point
knotpoint
m_j at knotpoints defines smoothness
$z_j(t)$
$z_j(t_f)$
$z_j(t_o)$
$k_j - 1$ degree polynomial between knotpoints

Figure 4.4: Spline representation of a variable.

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

where $B_{i,k_j}(t)$ is the B-spline basis function defined in [dB78] for the output z_j with order k_j, C_i^j are the coefficients of the B-spline, l_j is the number of knot intervals, and m_j is number of smoothness conditions at the knots. The set $(z_1, z_2, \ldots, z_{n-r})$ is thus represented by $M = \sum_{j \in \{1, r+1, \ldots, n\}} p_j$ coefficients.

In general, w collocation points are chosen uniformly over the time interval $[t_o, t_{\mathrm{f}}]$ (though non-uniform knots placements may also be considered). Both dynamics and constraints will be enforced at the collocation points. The problem can be stated as the following nonlinear programming form:

$$\min_{y \in \mathbb{R}^M} F(y) \qquad \text{subject to} \qquad \begin{cases} \Phi(z(y), \dot{z}(y), \ldots, z^{(n-r)}(y)) = 0, \\ lb \le c(y) \le ub, \end{cases} \tag{4.15}$$

where

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

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

4.5 Choosing Cost Functions

The receding horizon control methodology is a very powerful tool for design of feedback controllers for constrained, nonlinear control systems. While the controllers that it produces are guaranteed to be stable under appropriate conditions, the choice of cost functions is left to the designer and can often require substantial trial and error. In this section we describe some tools for helping obtain cost functions based on insights from linear systems theory.

Design approach

The basic philosophy that we propose is illustrated in Figure 4.5. We begin with

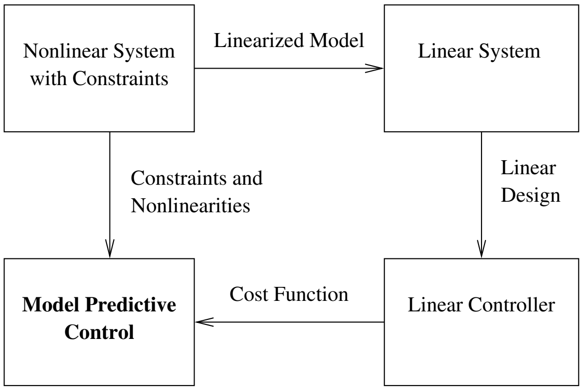


Figure 4.5: Optimization-based control approach.

a nonlinear system, including a description of the constraint set. We linearize this system about a representative equilibrium point and perform a linear control design using standard control design tools. Such a design can provide provably robust performance around the equilibrium point and, more importantly, allows the designer to meet a wide variety of formal and informal performance specifications through experience and the use of sophisticated linear design tools.

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

The advantage of the proposed approach is that it exploits the power of humans in designing sophisticated control laws in the absence of constraints with the power of computers to rapidly compute trajectories that optimize a given cost function in the presence of constraints. New advances in online trajectory generation serve as an enabler for this approach and their demonstration on representative flight control experiments shows their viability [MFHM05]. This approach can be extended to existing nonlinear paradigms as well, as we describe in more detail below.

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

A number of approaches in receding horizon control employ the use of terminal state equality or inequality constraints, often together with a terminal cost, to ensure closed loop stability. In Primbs et al. [PND99], aspects of a stability-guaranteeing, global control Lyapunov function (CLF) were used, via state and control constraints, to develop a stabilizing receding horizon scheme. Many of the nice characteristics of the control Lyapunov function controller together with better cost performance were realized. Unfortunately, a global control Lyapunov function is rarely available and often not possible.

Motivated by the difficulties in solving constrained optimal control problems, researchers have developed an alternative receding horizon control strategy for the stabilization of nonlinear systems [JYH01]. In this approach, closed loop stability is ensured through the use of a terminal cost consisting of a control Lyapunov function that is an incremental upper bound on the optimal cost to go. This terminal cost eliminates the need for terminal constraints in the optimization and gives a dramatic speed-up in computation. Also, questions of existence and regularity of optimal solutions (very important for online optimization) can be dealt with in a rather straightforward manner.

Inverse Optimality

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

One central question that must be addressed when considering the usefulness of this philosophy is: *Given a control law, how does one find an equivalent optimal control formulation?* The paper by Kalman [Kal64] lays a solid foundation for this class of problems, known as *inverse optimality.* In this paper, Kalman considers the class of linear time-invariant (LTI) processes with full-state feedback and a single input variable, with an associated cost function that is quadratic in the input and state variables. These assumptions set up the well-known linear quadratic regulator (LQR) problem, by now a staple of optimal control theory.

In Kalman's paper, the mathematical framework behind the LQR problem is laid out, and necessary and sufficient algebraic criteria for optimality are presented in terms of the algebraic Riccati equation, as well as in terms of a condition on the return difference of the feedback loop. In terms of the LQR problem, the task of synthesizing the optimal control problem comes down to finding the integrated cost weights Q_x and Q_u given only the dynamical description of the process represented by matrices A and B and of the feedback controller represented by K. Kalman delivers a particularly elegant frequency characterization of this map [Kal64], which we briefly summarize here.

We consider a linear system

$$\dot{x} = Ax + Bu \qquad x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{4.16}$$

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

$$u = -Kx$$

we wish to find a cost functional of the form

$$J = \int_0^{\mathsf{T}} x^{\mathsf{T}} Q_x x + u^{\mathsf{T}} Q_u u \, dt + x^{\mathsf{T}}(T) P_T x(T) \tag{4.17}$$

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

The optimal control law for the quadratic cost function (4.17) is given by

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

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

$$-\dot{P} = A^{\mathsf{T}} P + PA - PBR^{-1} B^{\mathsf{T}} P + Q \tag{4.18}$$

with terminal condition $P(T) = P_T$. In order for this to give a control law of the form $u = -Kx$ for a constant matrix K, we must find P_T, Q_x, and Q_u that give a constant solution to the Riccati equation (4.18) and satisfy $R^{-1} B^{\mathsf{T}} P = K$. It follows that P_T, Q_x and Q_u should satisfy

$$\begin{gathered} A^{\mathsf{T}} P_T + P_T A - P_T B Q_u^{-1} B^{\mathsf{T}} P_T + Q = 0 \\ -Q_u^{-1} B^{\mathsf{T}} P_T = K. \end{gathered} \tag{4.19}$$

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

Equation (4.19) is exactly the same equation that one would obtain if we had considered an infinite time horizon problem, since the given control was constant and hence $P(t)$ was forced to be constant. This infinite horizon problem is precisely the one that Kalman considered in 1964, and hence his results apply directly. Namely, in the single-input single-output case, we can always find a solution to the coupled equations (4.19) under standard conditions on reachability and observability [Kal64]. The equations can be simplified by substituting the second relation into the first to obtain

$$A^{\mathsf{T}} P_T + P_T A - K^{\mathsf{T}} R K + Q = 0.$$

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

The implication of these results is that any state feedback control law satisfying these assumptions can be realized as the solution to an appropriately defined receding horizon control law. Thus, we can implement the design framework summarized in Figure 4.5 for the case where our (linear) control design results in a state feedback controller.

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

The history of inverse optimal control for nonlinear systems goes back to the early work of Moylan and Anderson [MA73]. More recently, Sepulchre et al. [SJK97] showed that a nonlinear state feedback obtained by Sontag's formula from a control Lyapunov function (CLF) is inverse optimal. The connections of this inverse optimality result to passivity and robustness properties of the optimal state feedback are discussed in Jankovic *et al.* [JSK99]. Most results on inverse optimality do not consider the constraints on control or state. However, the results on the unconstrained inverse optimality justify the use of a more general nonlinear loss function in the integrated cost of a finite horizon performance index combined with a real-time optimization-based control approach that takes the constraints into account.

4.6 Implementation on the Caltech Ducted Fan (with M. Milam and N. Petit)

To demonstrate the use of the techniques described in the previous section, we present an implementation of optimization-based control on the Caltech Ducted Fan, a real-time, flight control experiment that mimics the longitudinal dynamics of an aircraft. The experiment is show in Figure 4.6. The work in this section is based on the work of Mark Milam and Ryan Franz [Mil03, MFHM05] and is drawn from [MHJ$^+$03]. It illustrates some of the practical considerations in implementing receding horizon control and describes techniques for addressing computational limitations.

The experimental results in this section were implemented using the Nonlinear Trajectory Generation (NTG) [MM02] software package. The sequential quadratic programming package NPSOL by [GMSW] is used as the nonlinear programming solver in NTG. When specifying a problem to NTG, the user is required to state the problem in terms of some choice of outputs and its derivatives. The user is also required to specify the regularity of the variables, the placement of the knot points, the order and regularity of the B-splines, and the collocation points for each output. The python-control package also includes an implementation of B-splines and these can be used for both flat and non-flat optimal trajectory generation problems.

Description of the Caltech Ducted Fan Experiment

The Caltech ducted fan is an experimental testbed designed for research and development of nonlinear flight guidance and control techniques for Uninhabited Aerial Vehicles (UAVs). The fan is a scaled model of the longitudinal axis of a flight vehicle and flight test results validate that the dynamics replicate qualities of actual flight vehicles [MM99].

The ducted fan has three degrees of freedom: the boom holding the ducted fan is allowed to operate on a cylinder, 2 m high and 4.7 m in diameter, permitting horizontal and vertical displacements. A counterweight is connected to the vertical axis of the stand, allowing the effective mass of the fan to be adjusted. Also, the wing/fan assembly at the end of the boom is allowed to rotate about its center of

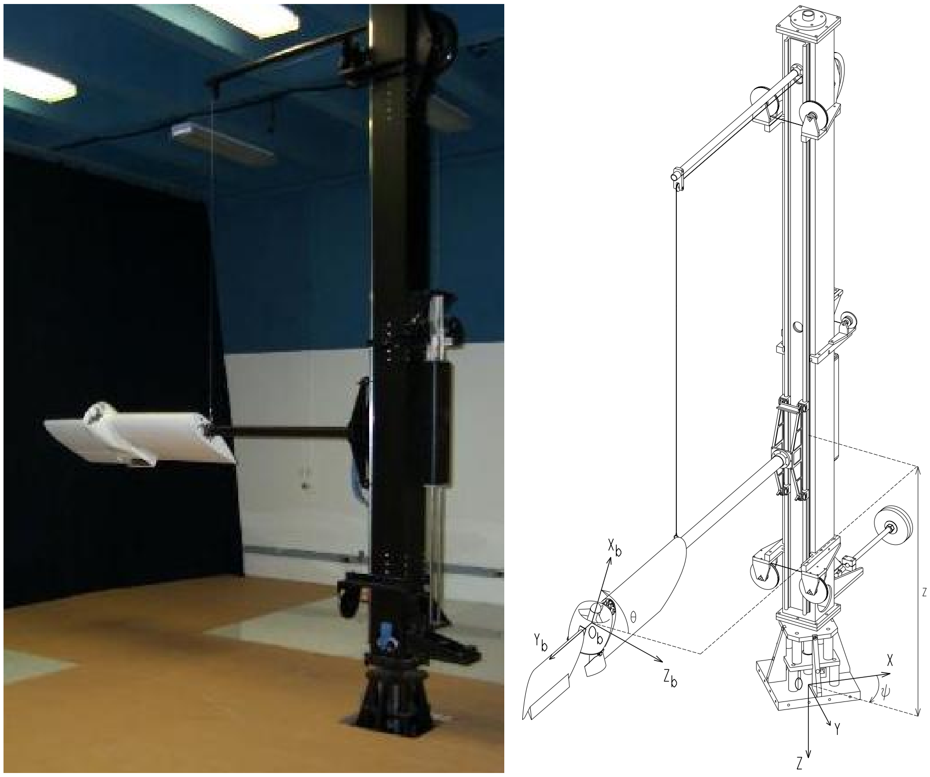


Figure 4.6: Caltech ducted fan.

mass. Optical encoders mounted on the ducted fan, counterweight pulley, and the base of the stand measure the three degrees of freedom. The fan is controlled by commanding a current to the electric motor for fan thrust and by commanding RC servos to control the thrust vectoring mechanism.

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

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

$$\begin{aligned} m\ddot{x} + F_{X_a} - F_{X_b}\cos\theta - F_{Z_b}\sin\theta &= 0, \\ m\ddot{z} + F_{Z_a} + F_{X_b}\sin\theta - F_{Z_b}\cos\theta &= mg_{\text{eff}}, \\ J\ddot{\theta} - M_a + \frac{1}{r_s}I_p\Omega\dot{x}\cos\theta - F_{Z_b}r_f &= 0, \end{aligned} \tag{4.20}$$

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

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

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

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

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

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

respectively. The dynamic pressure is given by $q = \frac{1}{2}\rho V^2$. The norm of the velocity is denoted by V, S the surface area of the wings, and ρ is the atmospheric density. The coefficients of lift ($C_L(\alpha)$), drag ($C_D(\alpha)$) and the moment coefficient ($C_M(\alpha)$) are determined from a combination of wind tunnel and flight testing and are described in more detail in [MM99], along with the values of the other parameters.

Real-Time Trajectory Generation

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

Stabilization around a reference trajectory

The results in this section rely on the traditional two degree of freedom design paradigm described in Chapter 2. In this approach, a local control law (inner loop) is used to stabilize the system around the trajectory computed based on a nominal model. This compensates for uncertainties in the model, which are predominantly due to aerodynamics and friction. Elements such as the ducted fan flying through its own wake, ground effects and velocity- and angle-of-attack dependent thrust contribute to the aerodynamic uncertainty. Actuation models are not used when generating the reference trajectory, resulting in another source of uncertainty.

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

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

Nonlinear trajectory generation parameters

We solve a minimum time optimal control problem to generate a feasible trajectory for the system. The system is modeled using the nonlinear equations described above and computed the open loop forces and state trajectories for the nominal system. This system is not known to be differentially flat (due to the aerodynamic forces) and hence we cannot completely eliminate the differential constraints.

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

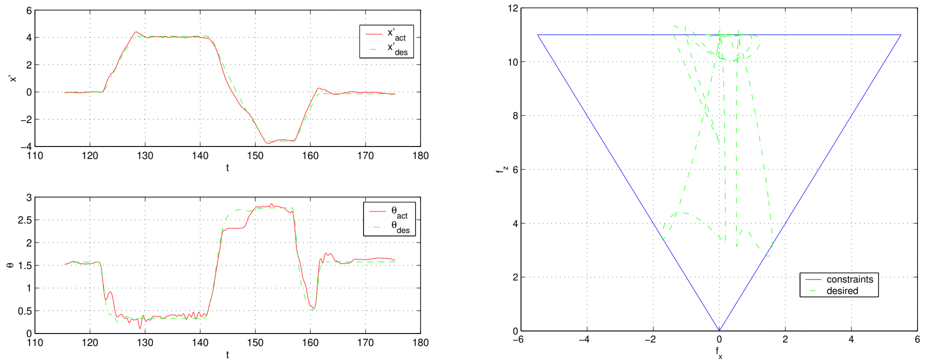
There are many considerations in the choice of the parameterization of the outputs. Clearly there is a trade between the parameters (variables, initial values of the variables, and breakpoints) and measures of performance (convergence, run-time, and conservative constraints). Extensive simulations were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

Forward flight

To obtain the forward flight test data, an operator commanded a desired forward velocity and vertical position with joysticks. We set the trajectory update time δ to 2 seconds. By rapidly changing the joysticks, NTG produces high angle of attack maneuvers. Figure 4.7aa depicts the reference trajectories and the actual θ and $\dot{x}$ over 60 s. Figure 4.7b shows the commanded forces for the same time interval. The sequence of maneuvers corresponds to the ducted fan transitioning from near hover to forward flight, then following a command from a large forward velocity to a large negative velocity, and finally returning to hover.

Figure 4.8 is an illustration of the ducted fan altitude and x position for these maneuvers. The air-foil in the figure depicts the pitch angle (θ). It is apparent from this figure that the stabilizing controller is not tracking well in the z direction. This is due to the fact that unmodeled frictional effects are significant in the vertical direction. This could be corrected with an integrator in the stabilizing controller.

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



(a) System state (b) Input forces

Figure 4.7: Forward flight test case: (a) θ and $\dot{x}$ desired and actual, (b) desired F_{X_b} and F_{Z_b} with bounds.

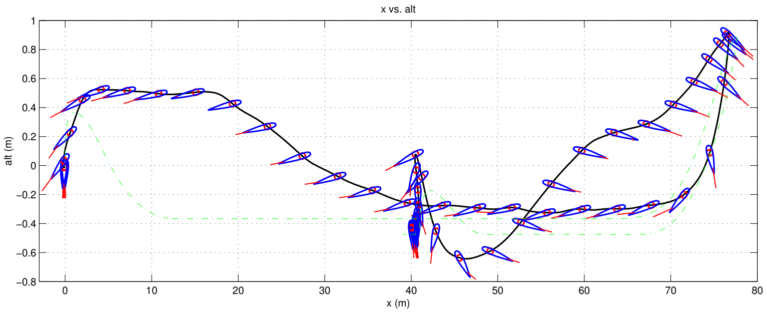


Figure 4.8: Forward flight test case: altitude and x position (actual (solid) and desired (dashed)). Airfoil represents actual pitch angle (θ) of the ducted fan.

Receding Horizon Control Implementation

The results of the previous section demonstrate the ability to compute optimal trajectories in real time, although the computation time was not sufficiently fast for closing the loop around the optimization. In this section, we make use of a shorter update time δ, a fixed horizon time T with a quadratic integral cost, and a control Lyapunov function terminal cost to implement a receding horizon controller as described in Section 4.2. We focus on the operation of the system to near hover, so that we can use the local linearization to find the terminal control Lyapunov function.

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

$$\begin{aligned} L(x,u) &= \frac{1}{2}\hat{x}^{\mathsf{T}} Q \hat{x} + \frac{1}{2}\hat{u}^{\mathsf{T}} R \hat{u}, \\ V(x) &= \gamma \hat{x}^{\mathsf{T}} P \hat{x}, \end{aligned} \tag{4.21}$$

where

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

For the terminal cost, we choose $\gamma = 0.075$ and P is the unique stable solution to the algebraic Riccati equation corresponding to the linearized dynamics of equation (4.20) at hover and the weights Q and R. Note that if $\gamma = 1/2$, then $V(\cdot)$ is the control Lyapunov function for the system corresponding to the LQR problem. Instead V is a relaxed (in magnitude) control Lyapunov function, which achieved better performance in the experiment. In either case, V is valid as a control Lyapunov function only in a neighborhood around hover since it is based on the linearized dynamics. We do not try to compute off-line a region of attraction for this control Lyapunov function. Experimental tests omitting the terminal cost and/or the input constraints leads to instability. The results in this section show the success of this choice for V for stabilization. An inner-loop PD controller on $\theta, \dot{\theta}$ is implemented to stabilize to the receding horizon states $\theta^*_T, \dot{\theta}^*_T$. The θ dynamics are the fastest for this system and although most receding horizon controllers were found to be nominally stable without this inner-loop controller, small disturbances could lead to instability.

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

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

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

A natural choice for updating the optimal trajectories for stabilization is to do so as fast as possible. This is achieved here by constantly resolving the optimization. When computation i is done, computation $i + 1$ is immediately started, so $t_{i+1} = t_i + \delta_c(i)$. Figure 4.9 gives a graphical picture of the timing set-up as the optimal input trajectories $u^*_T(\cdot)$ are updated. As shown in the figure, any computation i for $u^*_T(i)(\cdot)$ occurs for $t \in [t_i, t_{i+1}]$ and the resulting trajectory is applied for $t \in [t_{i+1}, t_{i+2}]$. At $t = t_{i+1}$ computation $i + 1$ is started for trajectory $u^*_T(i + 1)(\cdot)$, which is applied as soon as it is available ($t = t_{i+2}$). For the experimental runs

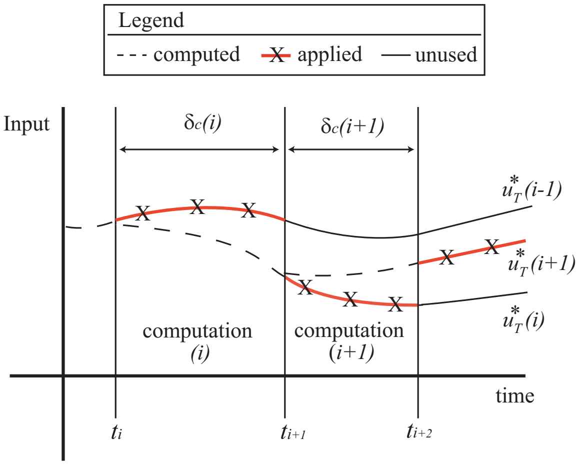
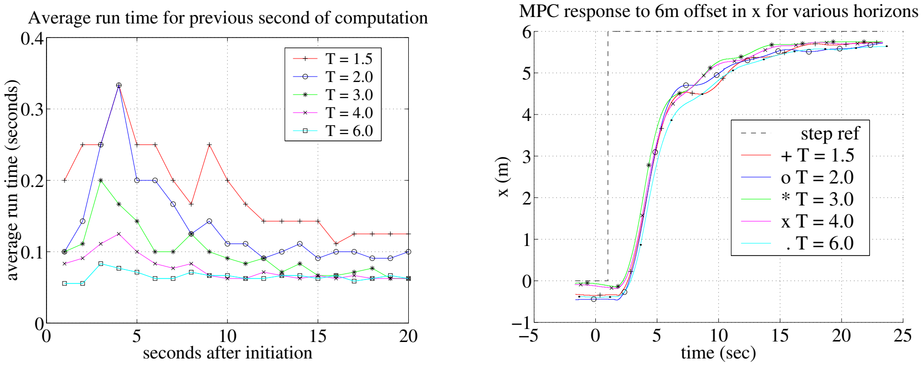


Figure 4.9: Receding horizon input trajectories.

detailed in the results, $\delta_c(i)$ is typically in the range of $[0.05, 0.25]$ seconds, meaning 4 to 20 optimal control computations per second. Each optimization i requires the current measured state of the ducted fan and the value of the previous optimal input trajectories $u^*_T(i-1)$ at time $t = t_i$. This corresponds to, respectively, 6 initial conditions for state vector x and 2 initial constraints on the input vector u. Figure 4.9 shows that the optimal trajectories are advanced by their computation time prior to application to the system. A dashed line corresponds to the initial portion of an optimal trajectory and is not applied since it is not available until that computation is complete. The figure also reveals the possible discontinuity between successive applied optimal input trajectories, with a larger discontinuity more likely for longer computation times. The initial input constraint is an effort to reduce such discontinuities, although some discontinuity is unavoidable by this method. Also note that the same discontinuity is present for the 6 open-loop optimal state trajectories generated, again with a likelihood for greater discontinuity for longer computation times. In this description, initialization is not an issue because we assume the receding horizon computations are already running prior to any test runs. This is true of the experimental runs detailed in the results.

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

The first comparison is between different receding horizon controllers, where time horizon is varied to be 1.5, 2.0, 3.0, 4.0 or 6.0 seconds. Each controller uses 16 breakpoints. Figure 4.10a shows a comparison of the average computation time as time proceeds. For each second after the offset was initiated, the data correspond to the average run time over the previous second of computation. Note that these computation times are substantially smaller than those reported for real-time trajectory generation, due to the use of the control Lyapunov function terminal cost versus the terminal constraints in the minimum-time, real-time trajectory genera-



(a) Average run time

(b) Step responses

Figure 4.10: Receding horizon control: (a) moving one second average of computation time for RHC implementation with varying horizon time, (b) response of RHC controllers to 6 meter offset in x for different horizon lengths.

tion experiments.

There is a clear trend toward shorter average computation times as the time horizon is made longer. There is also an initial transient increase in average computation time that is greater for shorter horizon times. In fact, the 6 second horizon controller exhibits a relatively constant average computation time. One explanation for this trend is that, for this particular test, a 6 second horizon is closer to what the system can actually do. After 1.5 seconds, the fan is still far from the desired hover position and the terminal cost control Lyapunov function is large, likely far from its region of attraction. Figure 4.10b shows the measured x response for these different controllers, exhibiting a rise time of 8–9 seconds independent of the controller. So a horizon time closer to the rise time results in a more feasible optimization in this case.

4.7 Further Reading

Receding horizon control (more commonly referred to as model predictive control or MPC) has a long history and there are many good textbooks available. The book by Rawlings, Mayne, and Diehl [RMD17] is an excellent resource (available for free download), as well as the textbook (also freely available) by Borelli, Bemporad, and Morari [BBM17], which matches the material in the MATLAB-based MPT toolbox. An overview of the early evolution of commercially available MPC technology is given in [QB97] and a survey of the state of stability theory of MPC circa 2000 is given in [MRRS00].

Exercises

4.1. Consider a nonlinear control system

$$\dot{x} = f(x, u)$$

with linearization

$$\dot{x} = Ax + Bu.$$

Show that if the linearized system is reachable, then there exists a (local) control Lyapunov function for the nonlinear system. (Hint: start by proving the result for a stable system.)

4.2. Consider the optimal control problem given in Example 3.2:

$$\dot{x} = ax + bu, \qquad J = \tfrac{1}{2}\int_{t_0}^{t_f} u^2(t)\,dt + \tfrac{1}{2}cx^2(t_f),$$

where $x \in \mathbb{R}$ is a scalar state, $u \in \mathbb{R}$ is the input, the initial state $x(t_0)$ is given, and $a, b \in \mathbb{R}$ are positive constants. We take the terminal time t_f as given and let $c > 0$ be a constant that balances the final value of the state with the input required to get to that position. The optimal control for a finite time $t_\text{f} > 0$ is derived in Example 3.2. Now consider the infinite horizon cost

$$J = \tfrac{1}{2}\int_{t_0}^{\infty} u^2(t)\,dt$$

with $x(t)$ at $t = \infty$ constrained to be zero.

(a) Solve for $u = -Q_u^{-1}B^TPx$ where P is the positive solution corresponding to the algebraic Riccati equation, and compute the optimal trajectory $x^*(t)$ from an initial condition $x(0)$.

(b) Plot the state solution of the finite time optimal controller for the following parameter values

$$\begin{aligned} a &= 2, & b &= 0.5, & x(t_0) &= 4,\\ c &= 0.1,\ 10, & ,t_0 &= 0, & t_\text{f} &= 0.5,\ 1,\ 10 \end{aligned}$$

(this should give you a total of 6 curves.) Compare these to the infinite time optimal control solution. Which finite time solution is closest to the infinite time solution? Why?

(c) Using the solution given in equation (3.5), implement the finite-time optimal controller in a receding horizon fashion with an update time of $\delta = 0.5$. Using the parameter values in part (b), compare the responses of the receding horizon controllers to the LQR controller you designed in part (a), from the same initial condition. What do you observe as c and t_f increase?

4.3. In this problem we will explore the effect of constraints on control of the linear unstable system given by

$$\dot{x}_1 = 0.8x_1 - 0.5x_2 + 0.5u, \qquad \dot{x}_2 \quad = x_1 + 0.5u,$$

subject to the constraint that $|u| \leq a$ where a is a postive constant.

(a) Ignore the constraint ($a = \infty$) and design an LQR controller to stabilize the system. Plot the response of the closed system from the initial condition given by $x = (1, 0)$.

(b) Simulate the initial condition response of system for some finite value of a with an initial condition $x(0) = (1, 0)$. Numerically (trial and error) determine the smallest value of a for which the system goes unstable.

(c) Let $a_{\text{min}}(\rho)$ be the smallest value of a for which the system is unstable from $x(0) = (\rho, 0)$. Plot $a_{\text{min}}(\rho)$ for $\rho = 1, 4, 16, 64, 256$.

(d) *Optional*: Given $a > 0$, design and implement a receding horizon control law for this system. Show that this controller has larger region of attraction than the controller designed in part (b). (Hint: solve the finite horizon LQ problem analytically, using the bang-bang example as a guide to handle the input constraint.)

4.4. [Instability of MPC with short horizons (Mark Cannon, Oxford University, 2020)] Consider a linear, discrete time system with dynamics

$$x[k+1] = \begin{bmatrix} 1 & 0.1 \\ 0 & 2 \end{bmatrix} x[k] + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} u[k], \qquad y[k] = \begin{bmatrix} 1 & 0 \end{bmatrix} x[k]$$

with finite time horizon cost given by

$$J(x[k], u) = \sum_{i=0}^{N-1} \Big(y^2[k+i] + u^2[k+i] \Big) + y^2[k+N].$$

(a) Show that the predicted state of the system can be written in the form

$$\begin{bmatrix} x[k] \\ x[k+1] \\ \vdots \\ x[k+N] \end{bmatrix} = \mathcal{M} x[k] + \mathcal{L} \begin{bmatrix} u[k] \\ u[k+1] \\ \vdots \\ u[k+N-1] \end{bmatrix}$$

and give formulas for $\mathcal{M}$ and $\mathcal{L}$ in terms of A, B, and C for the case $N = 3$.

(b) Show that the cost function can be written as

$$J(x[k], u) = \bar{u}^\mathsf{T}[k] H \bar{u}[k] + 2x^\mathsf{T}[k] F \bar{u}[k] + x^\mathsf{T}[k] G x[k],$$

where $\bar{u}[k] = (u[k], u[k+1], \dots, u[k+N-1]]$, and give expressions for F, G, and H.

(c) Show that the RHC controller that minimizes the cost function for a horizon length of N can be written as $u = -Kx$ and find an expression for K in terms of F, G, and H. Show that for $N = 3$ the feedback gain is given by

$$K = \begin{bmatrix} 0.1948 & 0.1168 \end{bmatrix}.$$

(d) Compute the closed loop eigenvalues for the system with a receding horizon controller with $N = 3$ and show that the system is unstable. What is the smallest value of N such that the system is stable?

(e) Change the terminal cost to use the optimal cost-to-go function returned by the `dlqr` command in MATLAB or Python. Verify that the closed loop system is stable for $N = 1, \dots, 5$.

4.5. Consider the double integrator system from Example 4.1. A discrete time representation of this system with sampling time of 1 second is given by

$$x[k+1] = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \operatorname{clip}(u), \qquad \text{where} \qquad \operatorname{clip}(u) = \begin{cases} -1 & u < -1, \\ u & -1 \leq u \leq 1, \\ 1 & u > 1. \end{cases}$$

We choose the same weighting matrices as in Example 4.1:

$$Q_x = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad Q_u = \begin{bmatrix} 1 \end{bmatrix}, \qquad P_1 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}.$$

(a) Construct a discrete-time receding horizon control law for the system and recreate Figure 4.3 using $x_0 = (2, 1)$ as the initial condition. Your plot should should show the actual trajectory for x and u as solid lines and the predicted trajectories from the optimization as dashed lines.

(b) The discrete time equivalent of the conditions in Theorem 4.1 are

$$\min_u V(f(x,u)) - V(x) + L(x,u) \leq 0 \quad \text{for all} \quad x,$$

where f represents the discrete time dynamics $x[k+1] = f(x[k], u[k])$. Check to see if these conditions are satisfied for this system using the weights above along the states that are visited along the trajectory of the system in (a). (For Theorem 4.1 to hold you would need to show this condition at all states x, so we are just checking a subset in this problem.)

(c) Replace the terminal cost P_1 with the solution to the discrete time algebraic Riccati equation (which can be obtained using the `dlqr` command in MATLAB or Python), recompute and plot the initial condition response of the receding horizon controller, and check that whether satisfies the stability condition along the states in the trajectory.

(d) Modify the terminal cost P_1 obtained in part (c) by 10X in each direction ($P_1' = 0.1 P_1$ and $P_1' = 10 P_1$), recompute and plot the initial condition response of the receding horizon controller, and check that whether satisfies the stability condition along the trajectory.

4.6. Consider the dynamics of the vectored thrust aircraft described in Examples 2.4 and 3.5. Assume that the inputs must satisfy the constraints

$$|F_1| \leq 0.1\,|F_2|, \qquad 0 \leq F_2 \leq 1.5\,mg.$$

(a) Design a receding horizon controller for the system that stabilizes the origin using an optimization horizon of $T = 3$ s and an update period of $\Delta T = 1$ s. Demonstrate the performance of your controller from initial conditions starting at initial position $(x_0,\, y_0) = (0 \text{ m},\, 5 \text{ m})$ and desired final position $(x_\text{f},\, y_\text{f}) = (10 \text{ m},\, 5 \text{ m})$ (all other states should be zero).

(b) Suppose that the system is subject to a sinusoidal disturbance force due to wind blowing in the horizontal direction, so that the dynamics in the x coordinate become

$$m\ddot{x} = F_1 \cos\theta - F_2 \sin\theta - c\dot{x} + d$$

with $d = \sin(t)$. Design a two-layer (inner/outer) feedback controller that uses the trajectories from (a) as inputs to an LQR controller that provides disturbance rejection. Compare the performance of the RHC controller in (a) in the presence of the disturbance to a two-layer controller using the initial and final conditions from (a).

(The Python function `pvtol-windy` in `pvtol.py` provides a model of this system using a third input corresponding to d.)

Chapter 5

Stochastic Systems

In this chapter we present a focused overview of stochastic systems, suitable for use in estimation theory. After a brief review of random variables, we define discrete-time and continuous-time random processes, including the expectation, (co-)variance, and correlation functions for a random process. These definitions are used to describe linear stochastic systems (in continuous time) and the stochastic response of a linear system to a random process (e.g., noise). We initially derive the relevant quantities in the state space, followed by a presentation of the equivalent frequency domain concepts.

Prerequisites. Readers should be familiar with basic concepts in probability, including random variables and standard distributions. We do not assume any prior familiarity with random processes.

Caveats. This chapter is written to provide a brief introduction to stochastic processes that can be used to derive the results in other subject areas. In order to keep the presentation compact, we gloss over several mathematical details that are required for rigorous presentation of the results. A more detailed (and mathematically precise) derivation of this material is available in the book by Åström [Åst06a].

5.1 Brief Review of Random Variables

To help fix the notation that we will use, we briefly review the key concepts of random variables. A more complete exposition is available in standard books on probability, such as Grimmett and Stirzaker [GS01].

Random variables and processes are defined in terms of an underlying probability space that captures the nature of the stochastic system we wish to study. A *probability distribution* is used to describe the likelihood that a random variable takes a given set of values.

More formally, a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ consists of:

- a *sample space* Ω that represents the set of all possible outcomes;
- a set of *events* $\mathcal{F}$ the captures combinations of elementary outcomes that are of interest; and

- a *probability measure* $\mathbb{P}$ that describes the likelihood of a given event occurring.

Ω can be any set, either with a finite, countable or infinite number of elements. The event space $\mathcal{F}$ consists of subsets of Ω. There are some mathematical limits on the properties of the sets in $\mathcal{F}$, but these are not critical for our purposes here. The probability measure $\mathbb{P}$ is a mapping from $\mathbb{P}\colon \mathcal{F} \to [0,1]$ that assigns a probability to each event. It must satisfy the property that given any two disjoint sets $A, B \in \mathcal{F}$, $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B)$.

With these definitions, we can model many different stochastic phenomena. Given a probability space, we can choose samples $\omega \in \Omega$ and identify each sample with a collection of events chosen from $\mathcal{F}$. These events should correspond to phenomena of interest and the probability measure $\mathbb{P}$ should capture the likelihood of that event occurring in the system that we are modeling. This definition of a probability space is very general and allows us to consider a number of situations as special cases.

A *random variable* X is a function $X\colon \Omega \to S$ that gives a value in S, called the state space, for any sample $\omega \in \Omega$. Given a subset $A \subset S$, we can write the probability that $X \in A$ as

$$\mathbb{P}(X \in A) = \mathbb{P}(\{\omega \in \Omega : X(\omega) \in A\}).$$

We will often find it convenient to omit ω when working random variables and hence we write $X \in S$ rather than the more correct $X(\omega) \in S$. The term *probability distribution* is used to describe the set of possible values that X can take.

A *discrete random variable* X is a variable that can take on any value from a discrete set S with some probability for each element of the set. We model a discrete random variable by its *probability mass function* $p_X(s)$, which gives the probability that the random variable X takes on the specific value $s \in S$:

$$p_X(s) = \text{probability that } X \text{ takes on the value } s \in S.$$

The sum of the probabilities over the entire set of states must be unity, and so we have that

$$\sum_{s \in S} p_X(s) = 1.$$

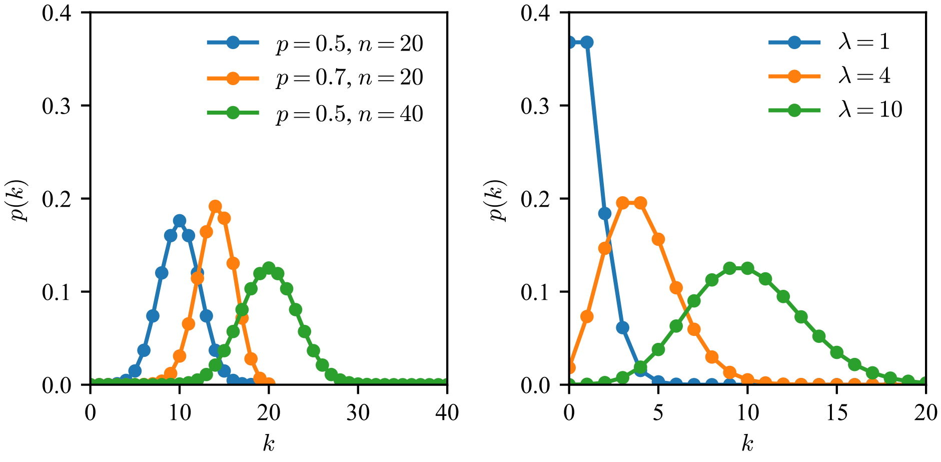
If A is a subset of S, then we can write $\mathbb{P}(X \in A)$ for the probability that X will take on some value in the set A. It follows from our definition that

$$\mathbb{P}(X \in A) = \sum_{s \in A} p_X(s).$$

Note that we use the convention that capital letters to refer to a random variable and lower case letters to refer to a specific value of the variable.

Definition 5.1 (Bernoulli distribution)**.** The Bernoulli distribution is used to model a random variable that takes the value 1 with probability p and 0 with probability $1-p$:

$$\mathbb{P}(X = 1) = p, \qquad \mathbb{P}(X = 0) = 1 - p.$$



(a) Binomial distribution (b) Poisson distribution

Figure 5.1: Probability mass functions for common discrete distributions.

Alternatively, it can be written in terms of its probability mass function

$$p(s) = \begin{cases} p & s = 1 \\ 1 - p & s = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Bernoulli distributions are used to model independent experiments with binary outcomes, such as flipping a coin.

Definition 5.2 (Binomial distribution)**.** The *binomial distribution* models the probability of successful trials in n experiments, given that a single experiment has probability of success p. If we let X_n be a random variable that indicates the number of success in n trials, then the binomial distribution is given by

$$p_{X_n}(k) = \mathbb{P}(X_n = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 1, \ldots, n$. The probability mass function is shown in Figure 5.1a.

Definition 5.3 (Poisson distribution)**.** The *Poisson distribution* is used to describe the probability that a given number of events will occur in a fixed interval of time t. The Poisson distribution is defined as

$$p_{N_t}(k) = \mathbb{P}(N_t = k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}, \tag{5.1}$$

where N_t is the number of events that occur in a period t and λ is a real number parameterizing the distribution. This distribution can be considered as a model for a counting process, where we assume that the average rate of occurrences in a period t is given by λt and λ represents the rate of the counting process. Figure 5.1b shows the form of the distribution for different values of k and λt.

When studying dynamical systems, we will make use of random variables that take on values in $\mathbb{R}$. A *continuous (real-valued) random variable* X is a variable that can take on any value in the set of real numbers $\mathbb{R}$. We can model the random variable X according to its *probability distribution function* $F\colon \mathbb{R} \to [0,1]$:

$$F(x) = \mathbb{P}(X \leq x) = \text{probability that } X \text{ takes on a value in the range } (-\infty, x].$$

More generally, we write $\mathbb{P}(A)$ as the probability that an event A will occur (e.g., $A = \{X \leq x\}$). It follows from the definition that if X is a random variable in the range $[L, U]$ then $\mathbb{P}(l \leq X \leq u) = 1$. Similarly, if $m \in [l, u]$ then $\mathbb{P}(l \leq X < m) = 1 - \mathbb{P}(m \leq X \leq u)$.

We characterize a random variable in terms of the *probability density function* (pdf) $p(x)$. The density function is defined so that its integral over an interval gives the probability that the random variable takes its value in that interval:

$$\mathbb{P}(x_l \leq X \leq x_u) = \int_{x_l}^{x_u} p(x) dx. \tag{5.2}$$

It is also possible to compute $p(x)$ given the distribution function $F(x) = \mathbb{P}(X \leq x)$ as long as the distribution function is suitably smooth:

$$p(x) = \frac{\partial F}{\partial x}(x).$$

We will sometimes write $p_X(x)$ when we wish to make explicit that the pdf is associated with the random variable X.

Probability distributions provide a general way to describe stochastic phenomena. Some common distributions include the uniform distribution, the Gaussian distribution, and the exponential distribution.

Definition 5.4 (Uniform distribution)**.** The *uniform distribution* on an interval $[L, U]$ assigns equal probability to any number in the interval. Its pdf is given by

$$p(x) = \frac{1}{U - L}. \tag{5.3}$$

The uniform distribution is illustrated in Figure 5.2a.

Definition 5.5 (Gaussian distribution)**.** The *Gaussian distribution* (also called a *normal distribution*) has a pdf of the form

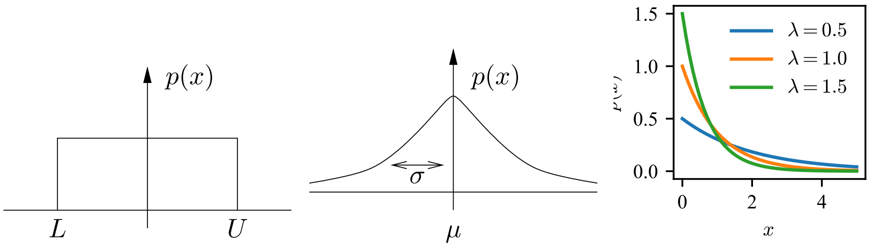
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{5.4}$$

The parameter μ is called the *mean* of the distribution and σ is called the *standard deviation* of the distribution. Figure 5.2b shows a graphical representation a Gaussian pdf.

Definition 5.6 (Exponential distribution)**.** The *exponential distribution* is defined for positive numbers and has a pdf of the form

$$p(x) = \lambda e^{-\lambda x}, \qquad x \geq 0$$

where λ is a parameter defining the distribution. A plot of the pdf for an exponential distribution is shown in Figure 5.2c.



(a) Uniform (b) Gaussian (c) Exponential

Figure 5.2: Probability density function (pdf) for uniform, Gaussian and exponential distributions.

There many other distributions that arise in applications, but for the purpose of these notes we focus on uniform distributions and Gaussian distributions.

We now define a number of properties of collections of random variables. We focus on the continuous random variable case, but unless noted otherwise these concepts can all be defined similarly for discrete random variables (using the probability mass function in place of the probability density function).

If two random variables are related, we can talk about their *joint probability distribution*: $\mathbb{P}_{X,Y}(A,B)$ is the probability that both event A occurs for X and B occurs for Y. This is sometimes written as $P(A \cap B)$, where we abuse notation by implicitly assuming that A is associated with X and B with Y. For continuous random variables, the joint probability distribution can be characterized in terms of a *joint probability density function*

$$F_{X,Y}(x,y) = \mathbb{P}(X \leq x,\, Y \leq y) = \int_{-\infty}^{y} \int_{-\infty}^{x} p(u,v)\, du\, dv. \tag{5.5}$$

The joint pdf thus describes the relationship between X and Y, and for sufficiently smooth distributions we have

$$p(x,y) = \frac{\partial^2 F}{\partial x \partial y}.$$

We say that X and Y are *independent* if $p(x,y) = p(x)\,p(y)$, which implies that $F_{X,Y}(x,y) = F_X(x)\,F_Y(y)$ for all x, y. Equivalently, $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$ if A and B are independent events.

The *conditional probability* for an event A given that an event B has occurred, written as $\mathbb{P}(A \mid B)$, is given by

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B)}{P(B)}. \tag{5.6}$$

If the events A and B are independent, then $\mathbb{P}(A \mid B) = \mathbb{P}(A)$. Note that the individual, joint and conditional probability distributions are all different, so if we are talking about random variables we can write $P_{X,Y}(A,B)$, $P_{X|Y}(A \mid B)$ and $P_Y(B)$, where A and B are appropriate subsets of $\mathbb{R}$.

If X is dependent on Y then Y is also dependent on X. *Bayes' theorem* relates the conditional and individual probabilities:

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(B \mid A)\mathbb{P}(A)}{\mathbb{P}(B)}, \qquad P(B) \neq 0. \tag{5.7}$$

Bayes' theorem gives the conditional probability of event A on event B given the inverse relationship (B given A). It can be used in situations in which we wish to evaluate a hypothesis H given data D when we have some model for how likely the data is given the hypothesis, along with the unconditioned probabilities for both the hypothesis and the data.

The analog of the probability density function for conditional probability is the *conditional probability density function* $p(x \mid y)$

$$p(x \mid y) = \frac{p(x,y)}{p(y)}, \qquad p(y) > 0. \tag{5.8}$$

It follows that

$$p(x,y) = p(x \mid y)p(y) \tag{5.9}$$

and

$$\begin{aligned} \mathbb{P}(X \leq x \mid y) &:= P(X \leq x \mid Y = y) \\ &= \int_{-\infty}^{x} p(u \mid y)du = \frac{\int_{-\infty}^{x} p(u,y)du}{p(y)}. \end{aligned} \tag{5.10}$$

If X and Y are independent than $p(x \mid y) = p(x)$ and $p(y \mid x) = p(y)$. Note that $p(x,y)$ and $p(x \mid y)$ are different density functions, though they are related through equation (5.9). If X and Y are related with joint probability density function $p(x,y)$ and conditional probability density function $p(x \mid y)$ then

$$p(x) = \int_{-\infty}^{\infty} p(x,y)dy = \int_{-\infty}^{\infty} p(x \mid y)p(y)dy.$$

Example 5.1 Conditional probability for a sum

Consider three random variables X, Y, and Z related by the expression

$$Z = X + Y.$$

In other words, the value of the random variable Z is given by choosing values from two random variables X and Y and adding them. We assume that X and Y are independent Gaussian random variables with mean μ_1 and μ_2 and standard deviation $\sigma = 1$ (the same for both variables).

Clearly the random variable Z is not independent of X (or Y) since if we know the values of X then it provides information about the likely value of Z. To see this, we compute the joint probability between Z and X. Let

$$A = \{x_l \leq x \leq x_u\}, \qquad B = \{z_l \leq z \leq z_u\}.$$

The joint probability of both events A and B occurring is given by

$$\begin{aligned} \mathbb{P}_{X,Z}(A \cap B) &= \mathbb{P}(x_l \leq x \leq x_u,\, z_l \leq x + y \leq z_u) \\ &= \mathbb{P}(x_l \leq x \leq x_u,\, z_l - x \leq y \leq z_u - x). \end{aligned}$$

We can compute this probability by using the probability density functions for X and Y:

$$\begin{aligned}\mathbb{P}(A \cap B) &= \int_{x_l}^{x_u} \Big(\int_{z_l - x}^{z_u - x} p_Y(y) dy \Big) p_X(x) dx \\ &= \int_{x_l}^{x_u} \int_{z_l}^{z_u} p_Y(z - x) p_X(x) dz dx =: \int_{z_l}^{z_u} \int_{x_l}^{x_u} p_{Z,X}(z, x) dx dz.\end{aligned}$$

Using Gaussians for X and Y we have

$$\begin{aligned}p_{Z,X}(z, x) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(z - x - \mu_Y)^2} \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x - \mu_X)^2} \\ &= \frac{1}{2\pi} e^{-\frac{1}{2}\big((z - x - \mu_Y)^2 + (x - \mu_X)^2\big)}.\end{aligned}$$

A similar expression holds for $p_{Z,Y}$. ∇

Given a random variable X, we can define various standard measures of the distribution. The *expectation* or *mean* of a random variable is defined as

$$\mathbb{E}(X) = \langle X \rangle = \int_{-\infty}^{\infty} x\, p(x)\, dx,$$

and the *mean square* of a random variable is

$$\mathbb{E}(X^2) = \langle X^2 \rangle = \int_{-\infty}^{\infty} x^2\, p(x)\, dx.$$

If we let μ represent the expectation (or mean) of X then we define the *variance* of X as

$$\mathbb{E}((X - \mu)^2) = \langle (X - \langle X \rangle)^2 \rangle = \int_{-\infty}^{\infty} (x - \mu)^2\, p(x)\, dx.$$

We will often write the variance as σ^2. As the notation indicates, if we have a Gaussian random variable with mean μ and (stationary) standard deviation σ, then the expectation and variance as computed above return μ and σ^2.

Example 5.2 Exponential distribution
The exponential distribution has mean and variance given by

$$\mu = \frac{1}{\lambda}, \qquad \sigma^2 = \frac{1}{\lambda^2}.$$

∇

Several useful properties follow from the definitions.

Proposition 5.1 (Properties of random variables)**.**

1. *If X is a random variable with mean μ and variance σ^2, then αX is random variable with mean $\alpha\mu$ and variance $\alpha^2\sigma^2$.*
2. *If X and Y are two random variables, then $\mathbb{E}(\alpha X + \beta Y) = \alpha\mathbb{E}(X) + \beta\mathbb{E}(Y)$.*

3. If X and Y are Gaussian random variables with means μ_X, μ_Y and variances σ_X^2, σ_Y^2,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} e^{-\frac{1}{2}\left(\frac{x-\mu_X}{\sigma_X}\right)^2}, \qquad p(y) = \frac{1}{\sqrt{2\pi\sigma_Y^2}} e^{-\frac{1}{2}\left(\frac{y-\mu_Y}{\sigma_Y}\right)^2},$$

then $X + Y$ is a Gaussian random variable with mean $\mu_Z = \mu_X + \mu_Y$ and variance $\sigma_Z^2 = \sigma_X^2 + \sigma_Y^2$,

$$p(x+y) = \frac{1}{\sqrt{2\pi\sigma_Z^2}} e^{-\frac{1}{2}\left(\frac{x+y-\mu_Z}{\sigma_Z}\right)^2}.$$

Proof. The first property follows from the definition of mean and variance:

$$\begin{aligned}
\mathbb{E}(\alpha X) &= \int_{-\infty}^{\infty} \alpha x \, p(x)\, dx = \alpha \int_{-\infty}^{\infty} \alpha x \, p(x)\, dx = \alpha \mathbb{E}(X) \\
\mathbb{E}((\alpha X)^2) &= \int_{-\infty}^{\infty} (\alpha x)^2 \, p(x)\, dx = \alpha^2 \int_{-\infty}^{\infty} x^2 \, p(x)\, dx = \alpha^2 \mathbb{E}(X^2).
\end{aligned}$$

The second property follows similarly, remembering that we must take the expectation using the joint distribution (since we are evaluating a function of two random variables):

$$\begin{aligned}
\mathbb{E}(\alpha X + \beta Y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (\alpha x + \beta y)\, p_{X,Y}(x,y)\, dxdy \\
&= \alpha \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x\, p_{X,Y}(x,y)\, dxdy + \beta \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} y\, p_{X,Y}(x,y)\, dxdy \\
&= \alpha \int_{-\infty}^{\infty} x\, p_X(x)\, dx + \beta \int_{-\infty}^{\infty} y\, p_Y(y)\, dy = \alpha \mathbb{E}(X) + \beta \mathbb{E}(Y).
\end{aligned}$$

The third item is left as an exercise. □

5.2 Introduction to Random Processes

A *random process* is a collection of time-indexed random variables. Formally, we consider a random process X to be a joint mapping of sample and a time to a state: $X : \Omega \times \mathcal{T} \to S$, where $\mathcal{T}$ is an appropriate time set. We view this mapping as a generalized random variable: a sample corresponds to choosing an entire function of time. Of course, we can always fix the time and interpret $X(\omega, t)$ as a regular random variable, with $X(\omega, t')$ representing a different random variable if $t \neq t'$. Our description of random processes will consist of describing how the random variable at a time t_2 relates to the value of the random variable at an earlier time t_1.

To build up some intuition about random processes, we will begin with the discrete time case, where the calculations are a bit more straightforward, and then proceed to the continuous time case.

A *discrete-time random process* is a stochastic system characterized by the *evolution* of a sequence of random variables $X[k]$, where k is an integer. As an example, consider a discrete-time linear system with dynamics

$$X[k+1] = AX[k] + Bu[k] + FV[k], \qquad Y[k] = CX[k] + W[k]. \tag{5.11}$$

As in FBS2e, $X \in \mathbb{R}^n$ represents the state of the system, $u \in \mathbb{R}^m$ is the vector of inputs (considered to be non-stochastic and therefore represented as u instead of U), and $Y \in \mathbb{R}^p$ is the vector of outputs. The (possibly vector-valued) signal V represents disturbances to the process dynamics and W represents noise in the measurements. To try to fix the basic ideas, we will take $u = 0$, $n = 1$ (single state) and $F = 1$ for now.

We wish to describe the evolution of the dynamics when the disturbances and noise are not given as deterministic signals, but rather are chosen from some probability distribution. Thus we will let $V[k]$ be a collection of random variables where the values at each instant k are chosen from a probability distribution with pdf $p_{V,k}(x)$. As the notation indicates, the distributions might depend on the time instant k, although the most common case is to have a *stationary* distribution in which the distributions are independent of k (defined more formally below).

In addition to stationarity, we will often also assume that distribution of values of V at time k is independent of the values of V at time l if $k \neq l$. In other words, $V[k]$ and $V[l]$ are two separate random variables that are independent of each other. We say that the corresponding random process is *uncorrelated* (also defined more formally below). As a consequence of our independence assumption, we have that

$$\mathbb{E}(V[k]V[l]) = \mathbb{E}(V^2[k])\delta(k-l) = \begin{cases} \mathbb{E}(V^2[k]) & k = l \\ 0 & k \neq l. \end{cases}$$

In the case that $V[k]$ is a Gaussian with mean zero and (stationary) standard deviation σ, then $\mathbb{E}(V[k]V[l]) = \sigma^2\,\delta(k-l)$.

We next wish to describe the evolution of the state x in equation (5.11) in the case when V is a random variable. In order to do this, we describe the state x as a sequence of random variables $X[k]$, $k = 1, \cdots, N$. Looking back at equation (5.11), we see that even if $V[k]$ is an uncorrelated sequence of random variables, then the states $X[k]$ are not uncorrelated since

$$X[k+1] = AX[k] + FV[k],$$

and hence the probability distribution for X at time $k+1$ depends on the value of X at time k (as well as the value of V at time k), similar to the situation in Example 5.1.

Since each $X[k]$ is a random variable, we can define the mean and variance as $\mu[k]$ and $\sigma^2[k]$ using the previous definitions at each time k:

$$\mu[k] := \mathbb{E}(X[k]) = \int_{-\infty}^{\infty} x\,p(x,k)\,dx,$$

$$\sigma^2[k] := \mathbb{E}((X[k]-\mu[k])^2) = \int_{-\infty}^{\infty} (x-\mu[k])^2\,p(x,k)\,dx.$$

To capture the relationship between the current state and the future state, we define the *correlation function* for a random process as

$$r(k_1, k_2) := \mathbb{E}(X[k_1]X[k_2]) = \int_{-\infty}^{\infty} x_1 x_2\, p(x_1, x_2; k_1, k_2)\, dx_1 dx_2$$

The function $p(x_i, x_j; k_1, k_2)$ is the *joint probability density function*, which depends on the times k_1 and k_2. A process is *stationary* if $p(x, k+d) = p(x, d)$ for all k, $p(x_i, x_j; k_1+d, k_2+d) = p(x_i, x_j; k_1, k_2)$, etc. In this case we can write $p(x_i, x_j; d)$ for the joint probability distribution. We will almost always restrict to this case. Similarly, we will write $r(k_1, k_2)$ as $r(d) = r(k, k+d)$.

We can compute the correlation function by explicitly computing the joint pdf (see Example 5.1) or by directly computing the expectation. Suppose that we take a random process of the form (5.11) with $\mathbb{E}(X[0]) = 0$ and V having zero mean and standard deviation σ. The correlation function is given by

$$\begin{aligned}
\mathbb{E}(X[k_1]X[k_2]) &= \mathbb{E}\Big\{\big(\sum_{i=0}^{k_1-1} A^{k_1-i}BV[i]\big)\big(\sum_{j=0}^{k_2-1} A^{k_2-j}BV[j]\big)\Big\} \\
&= \mathbb{E}\Big\{\sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} A^{k_1-i}BV[i]V[j]BA^{k_2-j}\Big\}.
\end{aligned}$$

We can now use the linearity of the expectation operator to pull this inside the summations:

$$\begin{aligned}
\mathbb{E}(X[k_1]X[k_2]) &= \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} A^{k_1-i}B\mathbb{E}(V[i]V[j])BA^{k_2-j} \\
&= \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} A^{k_1-i}B\sigma^2\delta(i-j)BA^{k_2-j} \\
&= \sum_{i=0}^{k_1-1} A^{k_1-i}B\sigma^2 BA^{k_2-i}.
\end{aligned}$$

Note that the correlation function depends on k_1 and k_2.

We can see the dependence of the correlation function on the time more clearly by letting $d = k_2 - k_1$ and writing

$$\begin{aligned}
r(k, k+d) = \mathbb{E}(X[k]X[k+d]) &= \sum_{i=0}^{k_1-1} A^{k-i}B\sigma^2 BA^{d+k-i} \\
&= \sum_{j=1}^{k} A^j B\sigma^2 BA^{j+d} = \Big(\sum_{j=1}^{k} A^j B\sigma^2 BA^j\Big)A^d.
\end{aligned}$$

In particular, if the discrete time system is asymptotically stable then $|A| < 1$ and the correlation function decays as we take points that are further departed in time (d large). Furthermore, if we let $k \to \infty$ (i.e., look at the steady state solution) then the correlation function only depends on d (assuming the sum converges) and hence the steady state random process is stationary.

In our derivation so far, we have assumed that $X[k+1]$ only depends on the value of the state at time k (this was implicit in our use of equation (5.11) and the assumption that $V[k]$ is independent of X). This particular assumption is known as the *Markov property* for a random process: a Markovian process is one in which the distribution of possible values of the state at time k depends only on the values of the state at the prior time and not earlier. Written more formally, we say that a discrete random process is Markovian if

$$p_{X,k}(x \mid X[k-1], X[k-2], \ldots, X[0]) = p_{X,k}(x \mid X[k-1]).$$

Markov processes are roughly equivalent to state space dynamical systems, where the future evolution of the system can be completely characterized in terms of the current value of the state (and not its history of values prior to that).

5.3 Continuous-Time, Vector-Valued Random Processes

We now consider the case where our time index is no longer discrete, but instead varies continuously. A fully rigorous derivation requires careful use of measure theory and is beyond the scope of this text, so we focus here on the concepts that will be useful for modeling and analysis of important physical properties.

A *continuous-time random process* is a stochastic system characterized by the evolution of a random variable $X(t)$, $t \in [0,T]$. We are interested in understanding how the (random) state of the system is related at separate times. The process is defined in terms of the "correlation" of $X(t_1)$ with $X(t_2)$. We assume, as above, that the process is described by continuous random variables, but the discrete state case (with time still modeled as a real variable) can be handled in a similar fashion.

We call $X(t) \in \mathbb{R}^n$ the *state* of the random process at time t. For the case $n > 1$, we have a vector of random processes:

$$X(t) = \begin{bmatrix} X_1(t) \\ \vdots \\ X_n(t) \end{bmatrix}$$

We can characterize the state in terms of a (joint) time-varying pdf,

$$\mathbb{P}(\{x_{i,l} \leq X_i(t) \leq x_{i,u}\}) = \int_{x_{1,l}}^{x_{1,u}} \cdots \int_{x_{n,l}}^{x_{n,u}} p_{X_1,\ldots,X_n}(x;t) dx_n \ldots dx_1.$$

Note that the state of a random process is not enough to determine the exact next state, but only the distribution of next states (otherwise it would be a deterministic process). We typically omit indexing of the individual states unless the meaning is not clear from context.

We can characterize the dynamics of a random process by its statistical characteristics, written in terms of joint probability density functions:

$$\begin{aligned} &\mathbb{P}(x_{1l} \leq X_i(t_1) \leq x_{1u},\, x_{2l} \leq X_j(t_2) \leq x_{2u}) \\ &\qquad\qquad = \int_{x_{2l}}^{x_{2u}} \int_{x_{1l}}^{x_{1u}} p_{X_i,Y_i}(x_1, x_2; t_1, t_2)\, dx_1 dx_2 \end{aligned}$$

The function $p(x_i, x_j; t_1, t_2)$ is called a *joint probability density function* and depends both on the individual states that are being compared and the time instants over which they are compared. Note that if $i = j$, then p_{X_i,X_i} describes how X_i at time t_1 is related to X_i at time t_2.

In general, the distributions used to describe a random process depend on the specific time or times that we evaluate the random variables. However, in some cases the relationship only depends on the difference in time and not the absolute times (similar to the notion of time invariance in deterministic systems, as described in FBS2e). A process is *stationary* if $p(x, t + s) = p(x, t)$ for all s, $p(x_i, x_j; t_1 + s, t_2 + s) = p(x_i, x_j; t_1, t_2)$, etc. In this case we can write $p(x_i, x_j; \tau)$ for the joint probability distribution, where $\tau = t_2 - t_1$. Stationary distributions roughly correspond to the steady state properties of a random process and we will often restrict our attention to this case.

We are often interested in random processes in which changes in the state occur when a random event occurs. In this case, it is natural to describe the state of the system in terms of a set of times $t_0 < t_1 < t_2 < \cdots < t_n$ and $X(t_i)$ is the random variable that corresponds to the possible states of the system at time t_i. Note that time time instants do not have to be uniformly spaced and most often (for physical systems) they will not be. All of the definitions above carry through, and the process can now be described by a probability distribution of the form

$$\mathbb{P}\Big(X(t_i) \in [x_i, x_i + dx_i],\ i = 1, \dots, n\Big) = \\ p(x_n, x_{n-1}, \dots, x_0;\ t_n, t_{n-1}, \dots, t_0)\, dx_n\, dx_{n-1}\, dx_1,$$

where dx_i are taken as infinitesimal quantities.

Just as in the case of discrete time processes, we define a continuous time random process to be a Markov process if the probability of being in a given state at time t_n depends *only* on the state that we were in at the previous time instant t_{n-1} and not the entire history of states prior to t_{n-1}:

$$\begin{aligned}\mathbb{P}\Big(X(t_n) &\in [x_n, x_n + dx_n] \mid X(t_i) \in [x_i, x_i + dx_i], i = 1, \dots, n-1\Big) \\ &= \mathbb{P}\Big(X(t_n) \in [x_n, x_n + dx_n] \mid X(t_{n-1}) \in [x_{n-1}, x_{n-1} + dx_{n-1}]\Big). \qquad (5.12)\end{aligned}$$

In practice we do not usually specify random processes via the joint probability distribution $p(x_i, x_j; t_1, t_2)$ but instead describe them in terms of a *propagator function.* Let $X(t)$ be a Markov process and define the Markov propagate as

$$\Xi(dt; x, t) = X(t + dt) - X(t), \quad \text{given } X(t) = x.$$

The propagate function describes how the random variable at time t is related to the random variable at time $t + dt$. Since both $X(t + dt)$ and $X(t)$ are random variables, $\Xi(dt; x, t)$ is also a random variable and hence it can be described by its density function, which we denote as $\Pi(\xi, x; dt, t)$:

$$\mathbb{P}\big(x \le X(t + dt) \le x + \xi\big) = \int_x^{x+\xi} \Pi(dx, x; dt, t)\, dx.$$

The previous definitions for mean, variance, and correlation can be extended to the continuous time, vector-valued case by indexing the individual states:

$$\mu(t) := \mathbb{E}(X(t)) = \begin{bmatrix} \mathbb{E}(X_1(t)) \\ \vdots \\ \mathbb{E}(X_n(t)) \end{bmatrix}$$

$$\Sigma(t) := \mathbb{E}((X(t)-\mu(t))(X(t)-\mu(t))^{\mathsf{T}}) = \begin{bmatrix} \mathbb{E}(X_1(t)X_1(t)) & \dots & \mathbb{E}(X_1(t)X_n(t)) \\ & \ddots & \vdots \\ & & \mathbb{E}(X_n(t)X_n(t)) \end{bmatrix}$$

$$R(t_1,t_2) := \mathbb{E}(X(t_1)X^{\mathsf{T}}(t_2)) = \begin{bmatrix} \mathbb{E}(X_1(t_1)X_1(t_2)) & \dots & \mathbb{E}(X_1(t_1)X_n(t_2)) \\ & \ddots & \vdots \\ & & \mathbb{E}(X_n(t_1)X_n(t_2)) \end{bmatrix}$$

Note that the random variables and their statistical properties are all indexed by the time t (or t_1 and t_2). The matrix $R(t_1,t_2)$ is called the *correlation matrix* for $X(t) \in \mathbb{R}^n$. If $t_1 = t_2 = t$ then $R(t,t)$ describes how the elements of x are correlated at time t (with each other) and in the case that the processes have zero mean, $R(t,t) = \Sigma(t)$. The elements on the diagonal of $\Sigma(t)$ are the variances of the corresponding scalar variables. A random process is uncorrelated if $R(t_1,t_2) = 0$ for all $t \neq s$. This implies that $X(t_1)$ and $X(t_2)$ are independent random events and is equivalent to $p_{X,Y}(x,y) = p_X(x)p_Y(y)$.

If a random process is stationary, then it can be shown that $R(t_1+\tau, t_2+\tau) = R(t_1,t_2)$ and it follows that the correlation matrix depends only on $t_2 - t_1$. In this case we will often write $R(t_1,t_2) = R(t_2-t_1)$ or simply $R(\tau)$ where τ is the correlation time. The covariance matrix in this case is simply $R(0)$.

In the case where X is also scalar random process, the correlation matrix is also a scalar and we will write $r(\tau)$, which we refer to as the (scalar) correlation function. Furthermore, for stationary scalar random processes, the correlation function depends only on the absolute value of the correlation function, so $r(\tau) = r(-\tau) = r(|\tau|)$. This property also holds for the diagonal entries of the correlation matrix since $R_{ii}(t_2,t_1) = R_{ii}(t_1,t_2)$ from the definition.

Definition 5.7 (Ornstein-Uhlenbeck process)**.** Consider a scalar random process defined by a Gaussian pdf with $\mu = 0$,

$$p(x,t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}},$$

and a correlation function given by

$$r(t_1,t_2) = \frac{Q}{2\omega_0} e^{-\omega_0|t_2-t_1|}.$$

The correlation function is illustrated in Figure 5.3. This process is known as an *Ornstein-Uhlenbeck process* and it is a stationary process.

Note on terminology. The terminology and notation for covariance and correlation varies between disciplines. The term covariance is often used to refer to both the

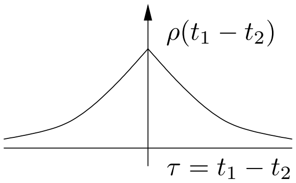


Figure 5.3: Correlation function for a first-order Markov process.

relationship between different variables X and Y and the relationship between a single variable at different times, $X(t_1)$ and $X(t_2)$. The term "cross-covariance" is used to refer to the covariance between two random vectors X and Y, to distinguish this from the covariance of the elements of X with each other. The term "cross-correlation" is sometimes also used. Finally, the term "correlation coefficient" refers to the normalized correlation.

We will also make use of a special type of random process referred to as "white noise". A *white noise process* $X(t)$ satisfies $\mathbb{E}(X(t)) = 0$ and $R(t_1, t_2) = Q\delta(t_2 - t_1)$, where $\delta(\tau)$ is the impulse function and $Q > 0$ is called the *noise intensity*. White noise is an idealized process, similar to the impulse function or Heaviside (step) function in deterministic systems. In particular, we note that $r(0) = \mathbb{E}(X^2(t)) = \infty$, so the covariance is infinite and we never see this signal in practice. However, like the step and impulse functions, it is very useful for characterizing the response of a linear system, as described in the following proposition. It can be shown that the integral of a white noise process is a Wiener process, and so often white noise is described as the derivative of a Wiener process.

5.4 Linear Stochastic Systems with Gaussian Noise

We now consider the problem of how to compute the response of a linear system to a random process. We assume we have a linear system described in state space as

$$\dot{X} = AX + FV, \qquad Y = CX \tag{5.13}$$

Given an "input" V, which is itself a Gaussian random process with mean $\mu(t)$, variance $\sigma^2(t)$ and correlation $r(t, t+\tau)$, what is the description of the random process Y?

Let V be a white noise process, with zero mean and noise intensity Q:

$$r(\tau) = Q\delta(\tau).$$

We can write the output of the system in terms of the convolution integral

$$Y(t) = \int_0^t h(t-\tau)V(\tau)\,d\tau,$$

where $h(t-\tau)$ is the impulse response for the system

$$h(t-\tau) = Ce^{A(t-\tau)}F$$

We now compute the statistics of the output, starting with the mean:

$$\begin{aligned}\mathbb{E}(Y(t)) &= \mathbb{E}\Big(\int_0^t h(t-\eta)V(\eta)\,d\eta\,\Big)\\ &= \int_0^t h(t-\eta)\mathbb{E}(V(\eta))\,d\eta = 0.\end{aligned}$$

Note here that we have relied on the linearity of the convolution integral to pull the expectation inside the integral.

We can compute the covariance of the output by computing the correlation $r(\tau)$ and setting $\sigma^2 = r(0)$. The correlation function for y is

$$\begin{aligned}r_Y(t_1,t_2) = \mathbb{E}\big(Y(t_1)Y(t_2)\big) &= \mathbb{E}\Big(\int_0^{t_1} h(t_1-\eta)V(\eta)\,d\eta\cdot\int_0^{t_2} h(t_2-\xi)V(\xi)\,d\xi\,\Big)\\ &= \mathbb{E}\Big(\int_0^{t_1}\int_0^{t_2} h(t_1-\eta)V(\eta)V(\xi)h(t_2-\xi)\,d\eta d\xi\,\Big).\end{aligned}$$

Once again linearity allows us to exchange expectation and integration:

$$\begin{aligned}r_Y(t_1,t_2) &= \int_0^{t_1}\int_0^{t_2} h(t_1-\eta)\mathbb{E}(V(\eta)V(\xi))h(t_2-\xi)\,d\eta d\xi\\ &= \int_0^{t_1}\int_0^{t_2} h(t_1-\eta)Q\delta(\eta-\xi)h(t_2-\xi)\,d\eta d\xi\\ &= \int_0^{t_1} h(t_1-\eta)Qh(t_2-\eta)\,d\eta.\end{aligned}$$

Now let $\tau = t_2 - t_1$ and write

$$\begin{aligned}r_Y(\tau) = r_Y(t,t+\tau) &= \int_0^t h(t-\eta)Qh(t+\tau-\eta)\,d\eta\\ &= \int_0^t h(\xi)Qh(\xi+\tau)\,d\xi \qquad (\text{setting } \xi = t-\eta).\end{aligned}$$

Finally, we let $t \to \infty$ (steady state) to obtain

$$\lim_{t\to\infty} r_Y(t,t+\tau) = \bar{r}_Y(\tau) = \int_0^\infty h(\xi)Qh(\xi+\tau)d\xi. \tag{5.14}$$

If this integral exists, then we can compute the second order statistics for the output Y.

We can provide a more explicit formula for the correlation function r in terms of the matrices A, F and C by expanding equation (5.14). We will consider the general case where $V \in \mathbb{R}^m$ and $Y \in \mathbb{R}^p$ and use the correlation matrix $R(t_1,t_2)$ instead of the correlation function $r(t_1,t_2)$. Define the *state transition matrix* $\Phi(t,t_0) = e^{A(t-t_0)}$ so that the solution of system (5.13) is given by

$$X(t) = \Phi(t,t_0)X(t_0) + \int_{t_0}^t \Phi(t,\lambda)FV(\lambda)d\lambda$$

Proposition 5.2 (Stochastic response to white noise)**.** *Let* $\mathbb{E}(X(t_0)X^{\mathsf{T}}(t_0)) = P(t_0)$ *and* V *be white noise with* $\mathbb{E}(V(\lambda)V^{\mathsf{T}}(\xi)) = R_V\delta(\xi-\lambda)$. *Then the correlation matrix for* X *is given by*

$$R_X(t_1,t_2) = P(t_1)\Phi^{\mathsf{T}}(t_2,t_1)$$

where $P(t)$ *satisfies the linear matrix differential equation*

$$\dot P(t) = AP + PA^{\mathsf{T}} + FR_VF^{\mathsf{T}}, \qquad P(0) \quad = P_0.$$

Proof. Using the definition of the correlation matrix, we have

$$\begin{aligned}\mathbb{E}(X(t_1)X^{\mathsf{T}}(t_2)) &= \mathbb{E}\big(\Phi(t,0)X(0)X^{\mathsf{T}}(0)\Phi^{\mathsf{T}}(t,0) + \text{cross terms}\\ &\qquad + \int_0^{t_1}\Phi(t_1,\xi)FV(\xi)\,d\xi\int_0^{t_2}V^{\mathsf{T}}(\lambda)F^{\mathsf{T}}\Phi(t_2,\lambda)\,d\lambda\Big)\\ &= \Phi(t_1,0)\mathbb{E}(X(0)X^{\mathsf{T}}(0))\Phi(t_2,0)\\ &\qquad + \int_0^{t_1}\int_0^{t_2}\Phi(t_1,\xi)F\mathbb{E}(V(\xi)V^{\mathsf{T}}(\lambda))F^{\mathsf{T}}\Phi(t_2,\lambda)\,d\xi\,d\lambda\\ &= \Phi(t_1,0)P(0)\Phi^{\mathsf{T}}(s_2,0) + \int_0^{t_1}\Phi(t_1,\lambda)FR_V(\lambda)F^{\mathsf{T}}\Phi(t_2,\lambda)\,d\lambda.\end{aligned}$$

Now use the fact that $\Phi(t_2,0) = \Phi(t_2,t_1)\Phi(t_1,0)$ (and similar relations) to obtain

$$R_X(t_1,t_2) = P(t_1)\Phi^{\mathsf{T}}(t_2,t_1)$$

where

$$P(t) = \Phi(t,0)P(0)\Phi^{\mathsf{T}}(t,0) + \int_0^{\mathsf{T}}\Phi(t,\lambda)FR_VF^{\mathsf{T}}(\lambda)\Phi^{\mathsf{T}}(t,\lambda)d\lambda$$

Finally, differentiate to obtain

$$\dot P(t) = AP + PA^{\mathsf{T}} + FR_VF^{\mathsf{T}}, \qquad P(0) \quad = P_0$$

(see Friedland [Fri04] for details). □

The correlation matrix for the output Y can be computed using the fact that $Y = CX$ and hence $R_Y = C^{\mathsf{T}}R_XC$. We will often be interested in the steady state properties of the output, which are given by the following proposition.

Proposition 5.3 (Steady state response to white noise)**.** *For a time-invariant linear system driven by white noise, the correlation matrices for the state and output converge in steady state to*

$$R_X(\tau) = R_X(t,t+\tau) = Pe^{A^{\mathsf{T}}\tau}, \qquad R_Y(\tau) = CR_X(\tau)C^{\mathsf{T}}$$

where P *satisfies the algebraic equation*

$$AP + PA^{\mathsf{T}} + FR_VF^{\mathsf{T}} = 0 \qquad P > 0. \tag{5.15}$$

Equation (5.15) is called the *Lyapunov equation* and can be solved in Python using the function `ct.lyap(A, Q)`, where $Q = FR_VF^{\mathsf{T}}$.

Example 5.3 First-order system
Consider a scalar linear process

$$\dot{X} = -aX + V, \qquad Y = cX,$$

where V is a white, Gaussian random process with noise intensity Q. Using the results of Proposition 5.2, the correlation function for X is given by

$$R_X(t, t+\tau) = p(t)e^{-a\tau}$$

where $p(t) > 0$ satisfies

$$\dot{p}(t) = -2ap + Q.$$

We can solve explicitly for $p(t)$ since it is a (non-homogeneous) linear differential equation:

$$p(t) = e^{-2at}p(0) + (1 - e^{-2at})\frac{Q}{2a}.$$

Finally, making use of the fact that $Y = cX$ we have

$$r(t, t+\tau) = c^2(e^{-2at}p(0) + (1 - e^{-2at})\frac{Q}{2a})e^{-a\tau}.$$

In steady state, the correlation function for the output becomes

$$r(\tau) = \frac{c^2 Q}{2a}e^{-a\tau}.$$

Note that the correlation function has the same form as the Ornstein-Uhlenbeck process in Example 5.7 (with $Q = c^2\sigma^2$). This process is also called a first-order Markov process, corresponding to the fact that white noise is passed through a first-order (low-pass) filter. ∇

5.5 Random Processes in the Frequency Domain

As in the case of deterministic linear systems, we can analyze a stochastic linear system either in the state space or the frequency domain. The frequency domain approach provides a very rich set of tools for modeling and analysis of interconnected systems, relying on the frequency response and transfer functions to represent the flow of signals around the system.

Given a random process $X(t)$, we can look at the frequency content of the properties of the response. In particular, if we let $r(\tau)$ be the correlation function for a (scalar) random process, then we define the *power spectral density function* as the Fourier transform of r:

$$S(\omega) = \int_{-\infty}^{\infty} r(\tau)e^{-j\omega\tau}\, d\tau, \qquad r(\tau) \;\; = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega)e^{j\omega\tau}\, d\tau.$$

The power spectral density provides an indication of how quickly the values of a random process can change through the frequency content: if there is high frequency content in the power spectral density, the values of the random variable can change quickly in time.

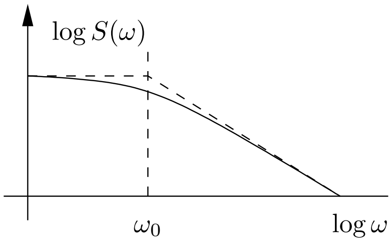


Figure 5.4: Power spectral density for a first-order Markov process.

Example 5.4 First-order Markov process
To illustrate the use of these measures, consider a first-order Markov process as defined in Example 5.7. The correlation function is

$$r(\tau) = \frac{Q}{2\omega_0} e^{-\omega_0(\tau)}.$$

The power spectral density becomes

$$\begin{aligned} S(\omega) &= \int_{-\infty}^{\infty} \frac{Q}{2\omega_0} e^{-\omega|\tau|} e^{-j\omega\tau}\, d\tau \\ &= \int_{-\infty}^{0} \frac{Q}{2\omega_0} e^{(\omega - j\omega)\tau}\, d\tau + \int_{0}^{\infty} \frac{Q}{2\omega_0} e^{(-\omega - j\omega)\tau}\, d\tau = \frac{Q}{\omega^2 + \omega_0^2}. \end{aligned}$$

We see that the power spectral density is similar to a transfer function and we can plot $S(\omega)$ as a function of ω in a manner similar to a Bode plot, as shown in Figure 5.4. Note that although $S(\omega)$ has a form similar to a transfer function, it is a real-valued function and is not defined for complex s. ∇

Using the power spectral density, we can more formally define "white noise": a *white noise process* is a zero-mean, random process with power spectral density $S(\omega) = Q =$ constant for all ω. If $X(t) \in \mathbb{R}^n$ (a random vector), then $Q \in \mathbb{R}^{n \times n}$. We see that a random process is white if all frequencies are equally represented in its power spectral density; this spectral property is the reason for the terminology "white". The following proposition verifies that this formal definition agrees with our previous (time domain) definition.

Proposition 5.4. *For a white noise process,*

$$r(\tau) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\omega) e^{j\omega\tau}\, d\tau = Q\delta(\tau),$$

where $\delta(\tau)$ is the unit impulse function.

Proof. If $\tau \neq 0$ then

$$r(\tau) = \frac{1}{2\pi} \int_{-\infty}^{\infty} Q(\cos(\omega\tau) + j\sin(\omega\tau)\, d\tau = 0$$

If $\tau = 0$ then $r(\tau) = \infty$. Can show that

$$r(0) = \lim_{\epsilon \to 0} \int_{-\epsilon}^{\epsilon} \int_{-\infty}^{\infty} (\cdots)\, d\omega d\tau = Q\delta(0)$$

$\square$

Given a linear system

$$\dot{X} = AX + FV, \qquad Y = CX,$$

with V given by white noise, we can compute the spectral density function corresponding to the output Y. We start by computing the Fourier transform of the steady state correlation function (5.14):

$$\begin{aligned} S_Y(\omega) &= \int_{-\infty}^{\infty} \left[\int_0^{\infty} h(\xi) Q h(\xi + \tau) d\xi \right] e^{-j\omega\tau}\, d\tau \\ &= \int_0^{\infty} h(\xi) Q \left[\int_{-\infty}^{\infty} h(\xi + \tau) e^{-j\omega\tau}\, d\tau \right] d\xi \\ &= \int_0^{\infty} h(\xi) Q \left[\int_0^{\infty} h(\lambda) e^{-j\omega(\lambda - \xi)}\, d\lambda \right] d\xi \\ &= \int_0^{\infty} h(\xi) e^{j\omega\xi}\, d\xi \cdot Q H(j\omega) = H(-j\omega) Q H(j\omega) \end{aligned}$$

This is then the (steady state) response of a linear system to white noise.

As with transfer functions, one of the advantages of computations in the frequency domain is that the composition of two linear systems can be represented by multiplication. In the case of the power spectral density, if we pass white noise through a system with transfer function $H_1(s)$ followed by transfer function $H_2(s)$, the resulting power spectral density of the output is given by

$$S_Y(\omega) = H_1(-j\omega) H_2(-j\omega) Q H_2(j\omega) H_1(j\omega).$$

As stated earlier, white noise is an idealized signal that is not seen in practice. One of the ways to produced more realistic models of noise and disturbances is to apply a filter to white noise that matches a measured power spectral density function. Thus, we wish to find a noise intensity Q and filter $H(s)$ such that we match the statistics $S(\omega)$ of a measured noise or disturbance signal. In other words, given $S(\omega)$, find $Q > 0$ and $H(s)$ such that $S(\omega) = H(-j\omega) Q H(j\omega)$. This problem is know as the *spectral factorization problem.*

Figure 5.5 summarizes the relationship between the time and frequency domains.

5.6 Implementation in Python

NumPy and SciPy have a number of functions to implement covariance and correlation, which mostly match the terminology here:

- `numpy.cov(Xs)` - returns the sample variance of the vector random variable $X \in \mathbb{R}^n$ where each column of `Xs` represents samples of X.

$$p(v) = \frac{1}{\sqrt{2\pi R_V}} e^{-\frac{v^2}{2R_V}} \qquad V \longrightarrow \boxed{H} \longrightarrow Y \qquad p(y) = \frac{1}{\sqrt{2\pi R_Y}} e^{-\frac{y^2}{2R_Y}}$$

$$S_V(\omega) = R_V \qquad\qquad S_Y(\omega) = H(-j\omega) R_V H(j\omega)$$

$$R_V(\tau) = R_V \delta(\tau) \qquad \begin{aligned} \dot{X} &= AX + FV \\ Y &= CX \end{aligned} \qquad \begin{aligned} R_Y(\tau) &= CPe^{-A|\tau|}C^{\mathsf{T}} \\ AP + PA^{\mathsf{T}} &+ FR_VF^{\mathsf{T}} = 0 \end{aligned}$$

Figure 5.5: Summary of steady state stochastic response.

- `numpy.cov(Xs, Ys)` - returns the (cross-)covariance of the variables X and Y where `Xs` and `Ys` represent samples of the given random variables.
- `scipy.correlate(X, Y)` - the "cross-correlation" between two random (1D) sequences. If these sequences came from a random process, this is a single sample approximation of the (discrete-time) correlation function. Use the function `scipy.correlation_lags(len(X), len(Y))` to compute the lag τ and `scipy.correlate(X, X)` to get the (auto) correlation function $r_X(\tau)$.

The python-control package has variants of these functions that do appropriate processing for continuous time models.

The `white_noise` function generates a (multi-variable) white noise signal of specified intensity as either a sampled continuous time signal or a discrete time signal. A white noise signal along a 1D array of linearly spaced set of times `timepts` can be computing using

```
V = ct.white_noise(timepts, Q[, dt])
```

where `Q` is a positive definite matrix providing the noise intensity and `dt` is the sampling time (or 0 for continuous time).

In continuous time, the white noise signal is scaled such that the integral of the covariance over a sample period is `Q`, thus approximating a white noise signal. In discrete time, the white noise signal has covariance `Q` at each point in time (without any scaling based on the sample time).

The `correlation` function computes the correlation matrix $\mathbb{E}(X^{\mathsf{T}}(t+\tau)X(t))$ or the cross-correlation matrix $\mathbb{E}(X^{\mathsf{T}}(t+\tau)Y(t))$:

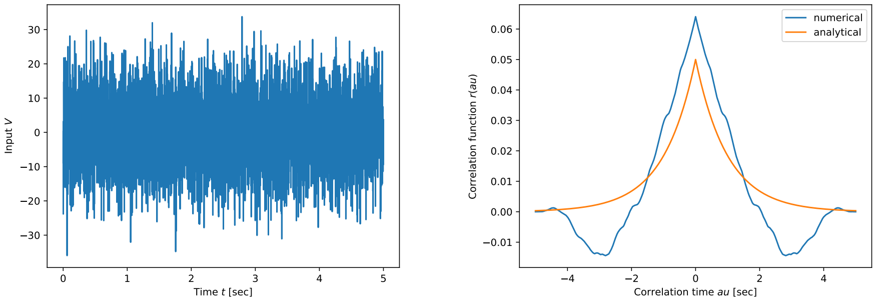
```
tau, Rtau = correlation(timepts, X[, Y])
```

The signal `X` (and `Y`, if present) represents a continuous time signal sampled at regularly spaced times `timepts`. The return value provides the correlation R_τ between $X(t+\tau)$ and $X(t)$ at a set of time offsets τ (determined based on the spacing of entries in the `timepts` vector.

Note that the computation of the correlation function is based on a single time signal (or pair of time signals) and is thus a very crude approximation to the true correlation function between two random processes.

To compute the response of a linear (or nonlinear) system to a white noise input, use the `forced_response` (or `input_output_response`) function:

```
a, c = 1, 1
sys = ct.ss([[-a]], [[1]], [[c]], 0)
timepts = np.linspace(0, 5, 1000)
```



(a) White noise input

(b) Output correlation function, $r(\tau)$

Figure 5.6: Stochastic response of a first order system.

```
Q = np.array([[0.1]])
V = ct.white_noise(timepts, Q)
resp = ct.forced_response(sys, timepts, V)
```

The correlation function for the output can be computed using the `correlation` function and compared to the analytical expression:

```
tau, r_Y = ct.correlation(timepts, resp.outputs)
plt.plot(tau, r_Y)
plt.plot(tau, c**2 * Q.item() / (2 * a) * np.exp(-a * np.abs(tau)))
```

An example set of outputs is shown in Figure 5.6.

5.7 Further Reading

There are several excellent books on stochastic systems that cover the results in this chapter in much more detail. For discrete-time systems, the textbook by Kumar and Varaiya [KV86] provides an derivation of the key results. Results for continuous-time systems can be found in the textbook by Friedland [Fri04]. Åström [Åst06a] gives a very elegant derivation in a unified framework that integrates discrete-time and continuous-time systems.

Exercises

5.1. Let Z be a random random variable that is the sum of two independent normally (Gaussian) distributed random variables X_1 and X_2 having means m_1, m_2 and variances σ_1^2, σ_2^2 respectively. Show that the probability density function for Z is

$$p(z) = \frac{1}{2\pi\sigma_1\sigma_2}\int_{-\infty}^{\infty} \exp\left\{-\frac{(z-x-m_1)^2}{2\sigma_1^2} - \frac{(x-m_2)^2}{2\sigma_2^2}\right\} dx$$

and confirm that this is normal (Gaussian) with mean m_1+m_2 and variance $\sigma_1^2+\sigma_2^2$. (Hint: Use the fact that $p(z|x_2) = p_{X_1}(x_1) = p_{X_1}(z-x_2)$.)

5.2 (*Feedback Systems*, 1st edition [ÅM08], Exercise 7.13). Consider the motion of a particle that is undergoing a random walk in one dimension (i.e., along a line). We model the position of the particle as

$$x[k+1] = x[k] + u[k],$$

where x is the position of the particle and u is a white noise process with $\mathbb{E}(u[i]) = 0$ and $\mathbb{E}(u[i]\, u[j]) R_u \delta(i-j)$. Show that the expected value of the particle as a function of k is given by

$$\mathbb{E}(x[k]) = x[0] =: \mu_x$$

and the covariance is given by

$$\mathbb{E}((x[k] - \mu_x)^2) = kR_u$$

5.3. Consider a second order system with dynamics

$$\begin{bmatrix} \dot{X}_1 \\ \dot{X}_2 \end{bmatrix} = \begin{bmatrix} -a & 0 \\ 0 & -b \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} V, \qquad Y = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

that is forced by Gaussian white noise with zero mean and variance σ^2. Assume $a, b > 0$ and $\mathbb{E}(X(0)) = 0$.

(a) Compute the (steady state) correlation function $r(\tau)$ for the output of the system. Your answer should be an explicit formula in terms of a, b and σ.

(b) Assuming that the input transients have died out, compute the mean and variance of the output.

5.4. Find a constant matrix A and vectors F and C such that for

$$\dot{X} = AX + FW, \qquad Y = CX$$

the power spectrum of Y is given by

$$S(\omega) = \frac{1 + \omega^2}{(1 - 7\omega^2)^2 + 1}$$

Describe the sense in which your answer is unique.

5.5. Consider the dynamics of the vectored thrust aircraft described in Examples 2.4 and 3.5 with disturbances added in the x and y coordinates:

$$\begin{aligned} m\ddot{x} &= F_1 \cos\theta - F_2 \sin\theta - c\dot{x} + D_x, \\ m\ddot{y} &= F_1 \sin\theta + F_2 \cos\theta - c\dot{y} - mg + D_y, \\ J\ddot{\theta} &= rF_1. \end{aligned} \tag{5.16}$$

The measured values of the system are the position and orientation, with added noise N_x, N_y, and N_θ:

$$\vec{Y} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} N_x \\ N_y \\ N_z \end{bmatrix}. \tag{5.17}$$

Assume that the input disturbances are modeled by independent, first order Markov (Ornstein-Uhlenbeck) processes with $Q_D = \text{diag}(0.01, 0.01)$ and $\omega_0 = 1$ (see Definition 5.7) and that the noise is modeled as white noise with covariance matrix

$$Q_N = \begin{bmatrix} 2 \times 10^{-4} & 0 & 1 \times 10^{-5} \\ 0 & 2 \times 10^{-4} & 1 \times 10^{-5} \\ 1 \times 10^{-5} & 1 \times 10^{-5} & 1 \times 10^{-4} \end{bmatrix}$$

(a) Create disturbance and noise vectors with the desired characteristics and then compute the sample mean, covariance, and correlation, showing that they match the specifications.

(b) Create a simulation of the PVTOL system with noise and disturbance inputs and plot the response of the system from an initial equilibrium position $(x_0, y_0) = (2, 1)$ to the origin using an LQR compensator with weights

$$Q_x = \text{diag}([1, 1, 10, 0, 0, 0]), Q_u = \text{diag}([10, 1]).$$

(c) Compute the linearization of the system and find the (steady state) mean and variance of the output of the linearized system.

(d) Use the linearization to compute the (analytical) correlation function for the output vector $\vec{Y}$. Plot the (auto) correlation function for x, y, and θ.

(e) Using the "stationary" (post-initial transient) portion of your simulation from part (b), compute the sample mean, sample variance, and sample correlation for the output Y. Compare these to the calculations from parts (c) and (d).

Chapter 6

Kalman Filtering

In this chapter we derive the optimal estimator for a linear system in continuous time (also referred to as the Kalman-Bucy filter). This estimator minimizes the covariance and can be implemented as a recursive filter. We also show how to combine optimal estimation with state feedback to solve the linear quadratic Gaussian (LQG) control problem, and explore extensions of Kalman filtering for continuous time systems, such as the extended Kalman filter. Optimal estimation of discrete time systems is described in more detail in Chapter 7, in the context of sensor fusion.

Prerequisites. Readers should have basic familiarity with continuous-time stochastic systems at the level presented in Chapter 5 as well as the material in FBS2e, Chapter 8 on state space observability and estimators.

6.1 Linear Quadratic Estimators

Consider a stochastic system

$$\dot{X} = AX + Bu + FV, \qquad Y \quad = CX + W,$$

where X represents that state, u is the (deterministic) input, V represents disturbances that affect the dynamics of the system and W represents measurement noise. Assume that the disturbance V and noise W are zero-mean, Gaussian white noise (but not necessarily stationary):

$$\begin{aligned} p(w) &= \frac{1}{\sqrt{\det(2\pi R_V)}} e^{-\frac{1}{2} w^{\mathsf{T}} R_V^{-1} w} & \qquad \mathbb{E}(V(t_1)V^{\mathsf{T}}(t_2)) &= R_V(t_1)\delta(t_2 - t_1) \\ p(v) &= \frac{1}{\sqrt{\det(2\pi R_V)}} e^{-\frac{1}{2} v^{\mathsf{T}} R_W^{-1} v} & \qquad \mathbb{E}(W(t_1)W^{\mathsf{T}}(t_2)) &= R_W(t_1)\delta(t_2 - t_1) \end{aligned}$$

We also assume that the cross correlation between V and W is zero, so that the disturbances are not correlated with the noise. Note that we use multi-variable Gaussians here, with noise intensities $R_V \in \mathbb{R}^{m \times m}$ and $R_W \in \mathbb{R}^{p \times p}$. In the scalar case, $R_V = \sigma_V^2$ and $R_W = \sigma_W^2$.

We formulate the optimal estimation problem as finding the estimate $\hat{x}(t)$ that minimizes the mean square error $\mathbb{E}((x(t) - \hat{X}(t))(X(t) - \hat{x}(t))^\mathsf{T})$ given $\{y(\tau) : 0 \leq \tau \leq t\}$ where X and $\hat{X}$ satisfy the dynamics for the system and y is the measured outputs of the system. Note that our system state is not known, but we do have a description of X as a random process, and hence we can reason over the distribution of possible states of that process that are consistent with the output measurements.

The estimation problem be viewed as solving a *least squares* problem: given all previous $y(t)$, find the estimate $\hat{X}(t)$ that satisfies the dynamics and minimizes the square error between the system state and the estimated state. It can be shown that this is equivalent to finding the expected value of X subject to the "constraint" given by all of the previous measurements, so that $\hat{X}(t) = \mathbb{E}(X(t) \mid Y(\tau), \tau \leq t)$. (This was the way that Kalman originally formulated the problem, and is explored in Exercise 6.1.)

The following theorem provides the solution to the optimal estimation problem for a linear system driven by disturbances and noise that are modeled as white noise processes.

Theorem 6.1 (Kalman-Bucy, 1961)**.** *The optimal estimator has the form of a linear observer*

$$\dot{\hat{x}} = A\hat{x} + Bu - L(C\hat{x} - y)$$

where $L(t) = P(t)C^\mathsf{T} R_W^{-1}$ *and* $P(t) = \mathbb{E}((X(t) - \hat{x}(t))(X(t) - \hat{x}(t))^\mathsf{T})$ *satisfies*

$$\begin{aligned} \dot{P} &= AP + PA^\mathsf{T} - PC^\mathsf{T} R_W^{-1}(t) CP + FR_V(t)F^\mathsf{T}, \\ P(0) &= \mathbb{E}(X(0)X^\mathsf{T}(0)). \end{aligned}$$

Sketch of proof. The error dynamics are given by

$$\dot{E} = (A - LC)E + \xi, \qquad \xi = FV - LW, \qquad R_\xi = FR_VF^\mathsf{T} + LR_WL^\mathsf{T}$$

The covariance matrix $P_E = P$ for this process satisfies

$$\begin{aligned} \dot{P} &= (A - LC)P + P(A - LC)^\mathsf{T} + FR_VF^\mathsf{T} + LR_WL^\mathsf{T} \\ &= AP + PA^\mathsf{T} + FR_VF^\mathsf{T} - LCP - PC^\mathsf{T}L^\mathsf{T} + LR_WL^\mathsf{T} \\ &= AP + PA^\mathsf{T} + FR_VF^\mathsf{T} + (LR_W - PC^\mathsf{T})R_W^{-1}(LR_W - PC^\mathsf{T})^\mathsf{T} \\ &\qquad - PC^\mathsf{T}R_W^{-1}CP, \end{aligned}$$

where the last line follows by completing the square. We need to find L such that $P(t)$ is as small as possible, which can be done by choosing L so that $\dot{P}$ decreases by the maximum amount possible at each instant in time. This is accomplished by setting

$$LR_W = PC^\mathsf{T} \qquad \Longrightarrow \qquad L = PC^\mathsf{T}R_W^{-1},$$

and the final form of the update law for P follows by substitution of L. □

Note that the Kalman filter has the form of a *recursive* filter: given $P(t) = \mathbb{E}(E(t)E^\mathsf{T}(t))$ at time t, can compute how the estimate and covariance *change*. Thus we do not need to keep track of old values of the output. Furthermore, the Kalman filter gives the estimate $\hat{X}(t)$ *and* the covariance $P_E(t)$, so you can see how well the error is converging.

Another observation is that form of the covariance update can be considered to consist of a *prediction* step and a *correction* step. If we had no information about the output, then the covariance matrix would change just as in the case of the stochastic response from Chapter 5:

$$\dot{P} = AP + PA^{\mathsf{T}} + FR_V(t)F^{\mathsf{T}}.$$

If A is stable then the first two terms tend to decrease the error covariance, but the third term will increase the covariance (because of the effect of disturbances). The remaining term in the covariance update is

$$-PC^{\mathsf{T}}R_W^{-1}(t)CP,$$

which we can regard as a correction term due to the feedback term $-L(C\hat{x} - y)$. This term decreases the covariance (because we have new data), but the amount to which it does so is limited by the noisiness of the measurement (hence the scaling by R_W^{-1}).

Example 6.1 First-order system
Consider a first-order linear system of the form

$$\dot{X} = -aX + V, \qquad Y = cX + W,$$

where V is white noise with variance σ_V^2 and W is white noise with variance σ_W^2. The optimal estimator has the form

$$\dot{\hat{x}} = -a\hat{x} - L(\hat{x} - y) \quad \text{where} \quad L = p(t)c/\sigma_W^2,$$

and the error covariance $p(t)$ satisfies the differential equation

$$\dot{p} = -2ap - \frac{c^2p^2}{\sigma_W^2} + \sigma_V^2, \qquad p(0) = \mathbb{E}(x(0)^2).$$

Figure 6.1 shows a sample plot of $p(t)$ and the estimate $\hat{x}$ versus x for an instance of the noise and disturbance signals. We see that while there is a large initial error in the state estimate, it quickly reduces the error and then (roughly) tracks the state of the underlying (noisy) process. (Since the disturbances are large and unknown, it is not possible to exactly track the actual system state.) ∇

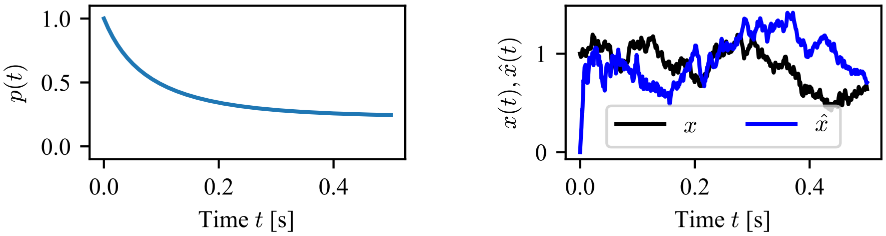
If the noise is stationary (R_V, R_W constant) and if the dynamics for $P(t)$ are stable, then the observer gain converges to a constant and satisfies the *algebraic Riccati equation*:

$$L = PC^{\mathsf{T}}R_W^{-1} \qquad AP + PA^{\mathsf{T}} - PC^{\mathsf{T}}R_W^{-1}CP + FR_VF^{\mathsf{T}}.$$

This is the most commonly used form of the controller since it gives an explicit formula for the estimator gains that minimize the error covariance. The gain matrix for this case can solved use the `control.lqe` command in Python or MATLAB.

Another property of the Kalman filter is that it extracts the maximum possible information about output data. To see this, consider the *residual random process*

$$R = Y - C\hat{X}$$



(a) Covariance $p(t)$ (b) State x and estimate $\hat{x}$

Figure 6.1: Optimal estimator for a first-order linear system with parameter values $a = 1$, $c = 1$, $\sigma_V = 1$, $\sigma_W = 0.1$, starting from initial condition $x(0) = 1$.

(this process is also called the *innovations process*). It can be shown for the Kalman filter that the correlation matrix of R is given by

$$R_R(t_1, t_2) = V(t_1)\delta(t_2 - t_1).$$

This implies that the residuals are a white noise process and so the output error has no remaining dynamic information content.

6.2 Extensions of the Kalman Filter

The Kalman filter has a number of extensions that are used to extend its utility to cases where the noise and disturbances are not white noise and when the process is not linear. We summarize some of these extensions briefly here, with additional extensions provided in the next chapter.

Correlated disturbances and noise

The derivation of the Kalman filter assumes that the disturbances and noise are independent and white. Removing the assumption of independence is straightforward and simply results in a cross term ($\mathbb{E}(V(t)W(s)) = R_{VW}\delta(s-t)$) being carried through all calculations.

To remove the assumption of white noise for the process disturbances or sensor noise, we can construct a filter that takes white noise as an input and produces a random process with the appropriate correlation function (or equivalently, spectral power density function). The intuition behind this approach is that we must have an internal model of the noise and/or disturbances in order to capture the correlation between different times.

Extended Kalman filters

Consider a *nonlinear* system

$$\begin{aligned} \dot{X} &= f(X, u, V), & X &\in \mathbb{R}^n, u \in \mathbb{R}^m, \\ Y &= CX + W, & Y &\in \mathbb{R}^p, \end{aligned}$$

where V and W are Gaussian white noise processes with covariance matrices R_V and R_W. A nonlinear observer for the system can be constructed by using the process

$$\dot{\hat{X}} = f(\hat{X}, u, 0) + L(Y - C\hat{X}).$$

If we define the error as $E = X - \hat{X}$, the error dynamics are given by

$$\begin{aligned}\dot{E} &= f(X, u, V) - f(\hat{X}, u, 0) - LC(X - \hat{X}) \\ &= F(E, \hat{X}, u, V) - LCe,\end{aligned}$$

where

$$F(E, \hat{X}, u, V) = f(E + \hat{X}, u, V) - f(\hat{X}, u, 0).$$

We can now linearize around *current* estimate $\hat{X}$:

$$\begin{aligned}\hat{E} &= \frac{\partial F}{\partial E}E + \underbrace{F(0, \hat{X}, u, 0)}_{=0} + \underbrace{\frac{\partial F}{\partial V}V}_{\text{noise}} - \underbrace{LCe}_{\text{observer gain}} + \text{h.o.t} \\ &\approx \tilde{A}E + \tilde{F}V - LCE,\end{aligned}$$

where the matrices

$$\begin{aligned}\tilde{A} &= \left.\frac{\partial F}{\partial e}\right|_{(0,\hat{X},u,0)} = \left.\frac{\partial f}{\partial X}\right|_{(\hat{X},u,0)}, \\ \tilde{F} &= \left.\frac{\partial F}{\partial V}\right|_{(0,\hat{X},u,0)} = \left.\frac{\partial f}{\partial V}\right|_{(\hat{X},u,0)}\end{aligned}$$

depend on current estimate $\hat{X}$. We can now design an observer for the linearized system around the *current* estimate:

$$\begin{aligned}\dot{\hat{X}} &= f(\hat{X}, u, 0) + L(Y - C\hat{X}), \qquad L = PC^{\mathsf{T}}R_V^{-1}, \\ \dot{P} &= (\tilde{A} - LC)P + P(\tilde{A} - LC)^{\mathsf{T}} + \tilde{F}R_V\tilde{F}^{\mathsf{T}} + LR_WL^{\mathsf{T}}, \\ P(t_0) &= \mathbb{E}(X(t_0)X^{\mathsf{T}}(t_0)).\end{aligned}$$

This is called the (Schmidt) *extended Kalman filter* (EKF).

The intuition in the Kalman filter is that we replace the prediction portion of the filter with the nonlinear modeling while using the instantaneous linearization to compute the observer gain. Although we lose optimality, in applications the extended Kalman filter often works well and it is very versatile, as illustrated in the following example.

Example 6.2 Online parameter estimation
Consider a linear system with unknown parameters ξ

$$\begin{aligned}\dot{X} &= A(\xi)X + B(\xi)u + FV, \qquad \xi \in \mathbb{R}^p, \\ Y &= C(\xi)X + W.\end{aligned}$$

We wish to solve the parameter identification problem: given $u(t)$ and $Y(t)$, estimate the value of the parameters ξ.

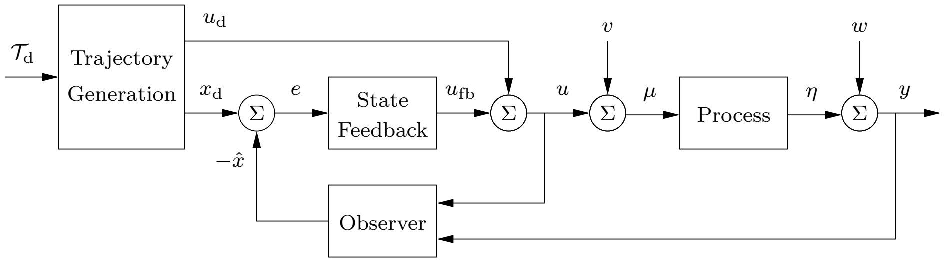


Figure 6.2: Block diagram of a basic feedback loop.

One approach to this online parameter estimation problem is to treat ξ as an unknown *state* that has zero derivative:

$$\dot{X} = A(\xi)X + B(\xi)u + FV, \qquad \dot{\xi} = 0.$$

We can now write the dynamics in terms of the extended state $Z = (X, \xi)$:

$$\frac{d}{dt}\begin{bmatrix}X\\ \xi\end{bmatrix} = \overbrace{\begin{bmatrix}A(\xi) & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}X\\ \xi\end{bmatrix} + \begin{bmatrix}B(\xi)\\ 0\end{bmatrix}u}^{f(\left[\begin{smallmatrix}X\\ \xi\end{smallmatrix}\right],u,V)} + \begin{bmatrix}F\\ 0\end{bmatrix}V,$$

$$Y = \underbrace{C(\xi)X + W}_{h(\left[\begin{smallmatrix}X\\ \xi\end{smallmatrix}\right],V)}.$$

This system is nonlinear in the extended state Z, but we can use the extended Kalman filter to *estimate* Z. If this filter converges, then we obtain both an estimate of the original state X and an estimate of the unknown parameter $\xi \in \mathbb{R}^p$.

Remark: need various observability conditions on augmented system in order for this to work. ∇

6.3 LQG Control

We now return to the full control problem, in which we wish to design a controller that uses the estimated state and tracks a trajectory. Figure 1.4 shows the high level view of the system, which we replicate in Figure 6.2, leaving out the unmodeled dynamics for simplicity. We assume that all processes are linear and hence it will suffice to consider the problem of stabilizing the origin.

The model for our process dynamics now must include the control input u and so we write

$$\begin{aligned}\dot{X} &= AX + Bu + FV,\\ Y &= CX + W,\end{aligned}$$

where V and W are white noise processes with appropriate covariances.

The *linear quadratic Gaussian control problem* is to find a controller that will minimize

$$J = \mathbb{E}\left(\int_0^\infty \left[(Y - y_\mathrm{d})^\mathsf{T} Q_y (Y - y_\mathrm{d}) + (U - u_\mathrm{d})^\mathsf{T} Q_u (U - u_\mathrm{d})\right] dt\right),$$

where U is now considered as a random variable. While in general we might imagine that the optimal controller could require some complex combination of state estimation and state feedback, it turns out that it can be shown that the structure of the optimal control separates into an optimal controller assuming perfect state knowledge and an optimal estimator that is independent of the control system gains.

Theorem 6.2 (Separation principle)**.** *The optimal controller for a linear system with white noise process disturbances and sensor noise has the form*

$$\begin{aligned}\dot{\hat{x}} &= A\hat{x} + Bu - L(C\hat{x} - y)\\ u &= u_d - K(\hat{x} - x_d)\end{aligned}$$

where L is the optimal observer gain ignoring the controller and K is the optimal controller gain ignoring the noise.

This is called the *separation principle* (for H_2 control). A proof of this theorem can be found in Friedland [Fri04] (and many other textbooks).

6.4 Implementation in Python

Stationary Kalman gains can be computed in python-control using the `lqe` function, which constructs an optimal estimator gain and covariance for a linear system. The Python command

```
L, P, E = ct.lqe(sys, Qv, Qw[, Qvw])
```

computes the optimal estimator gain `L`, steady state error covariance matrix `P`, and closed loop poles for the estimator `E` given the system dynamics and covariance of the process disturbances (`Qv`) and sensor noise (`Qw`), as well as any cross-covariance between the two sets of signals (`Qvw`).

The `create_estimator_iosystem` function can be used to create an I/O system implementing a Kalman filter, including integration of the Riccati ODE. The command has the form

```
estim = ct.create_estimator_iosystem(sys, Qv, Qw)
```

The input to the estimator is the measured outputs y and the system input u. To run the estimator on a noisy signal, use the command

```
resp = ct.input_output_response(est, timepts, [Y, U], [X0, P0])
```

If desired, the `correct` parameter can be set to `False` to allow prediction with no additional sensor information:

```
resp = ct.input_output_response(
    estim, timepts, 0, [X0, P0], param={'correct': False})
```

The `create_statefbk_iosystem` function can be used to combine an estimator with a state feedback controller:

```
K, _, _ = ct.lqr(sys, Qx, Qu)
estim = ct.create_estimator_iosystem(sys, Qv, Qw, P0)
ctrl, clsys = ct.create_statefbk_iosystem(sys, K, estimator=estim)
```

The controller will have the same form as a full state feedback controller, but with the system state x input replaced by the estimated state $\hat{x}$ (output of `estim`):

$$u = u_{\mathrm{d}} - K(\hat{x} - x_{\mathrm{d}}).$$

The closed loop controller `clsys` includes both the state feedback and the estimator dynamics and takes as its input the desired state x_{d} and input u_{d}:

```
resp = ct.input_output_response(
    clsys, timepts, [Xd, Ud], [X0, np.zeros_like(X0), P0])
```

6.5 Application to a Thrust Vectored Aircraft

To illustrate the use of the Kalman filter, we consider the problem of estimating the state for the Caltech ducted fan, described already in Section 4.6. We use the simplified model described in Example 3.5, with added disturbances and noise.

We begin by defining an extended Kalman filter that uses the nonlinear dynamics to estimate the current state. The dynamics of the system with disturbances on the x and y variables is given by

$$\begin{aligned} m\ddot{x} &= F_1 \cos\theta - F_2 \sin\theta - c\dot{x} + d_x, \\ m\ddot{y} &= F_1 \sin\theta + F_2 \cos\theta - c\dot{y} - mg + d_y, \\ J\ddot{\theta} &= rF_1. \end{aligned} \tag{6.1}$$

The measured values of the system are the position and orientation, with added noise n_x, n_y, and n_θ:

$$\vec{y} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}. \tag{6.2}$$

We assume that the disturbances are represented by white noise with intensity $\sigma^2 = 0.01$ and that the sensor noise has noise intensity matrix

$$Q_N = \begin{bmatrix} 2\times10^{-4} & 0 & 1\times10^{-5} \\ 0 & 2\times10^{-4} & 1\times10^{-5} \\ 1\times10^{-5} & 1\times10^{-5} & 1\times10^{-4} \end{bmatrix}.$$

To compute the update for the Kalman filter, we require the linearization of the system at a state $\vec{x} = (x, y, \theta, \dot{x}\dot{y}, \dot{z})$, which can be computed from equation (6.1) to be

$$\dot{X} = AX + Bu + FV,$$

where

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{F_1}{m}s_\theta - \frac{F_2}{m}c_\theta & -\frac{c}{m} & 0 & 0 \\ 0 & 0 & \frac{F_1}{m}c_\theta - \frac{F_2}{m}s_\theta & 0 & -\frac{c}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{m}c_\theta & -\frac{1}{m}s_\theta \\ \frac{1}{m}s_\theta & \frac{1}{m}c_\theta \\ r/J & 0 \end{bmatrix}, \quad F = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix},$$

with $c_\theta = \cos\theta$ and $s_\theta = \sin\theta$.

The state estimate is given by using the nonlinear dynamics for the prediction of the state error with a linear correction term, and the linearized dynamics for the update of the covariance matrix. If we let $\xi = (x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}) \in \mathbb{R}^6$ represent the states and $\eta = (x, y, \theta) \in \mathbb{R}^3$ represent write the the output, the dynamics of the state estimate written as

$$\dot{\hat{\xi}} = f(\xi, u) - L(C\hat{\xi} - \eta),$$

where $f(\xi, u)$ represents the full nonlinear dynamics in equation (6.1) and $C = \begin{bmatrix} I & 0 \end{bmatrix} \in \mathbb{R}^{3\times 6}$ represents the output matrix. The gain matrix $L = P(t)C^\mathsf{T} R_W^{-1}$ is chosen based on the time-varying error covariance matrix $P(t)$, which evolves using the linearized dynamics:

$$\begin{aligned} \dot{P} &= A(\xi)P + PA(\xi)^\mathsf{T} - PC^\mathsf{T} R_W^{-1}(t)CP + FR_V(t)F^\mathsf{T}, \\ P(0) &= \mathbb{E}(X(0)X^\mathsf{T}(0)). \end{aligned}$$

To show how this estimator can be used, consider the problem of stabilizing the system to the origin with an LQR controller that uses the estimated state. We compute the LQR controller as if the entire state ξ were available directly, so the computations are identical to those in Example 3.5. We choose the physically motivated set of weights given by

$$Q_\xi = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 36/\pi & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad Q_u = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}.$$

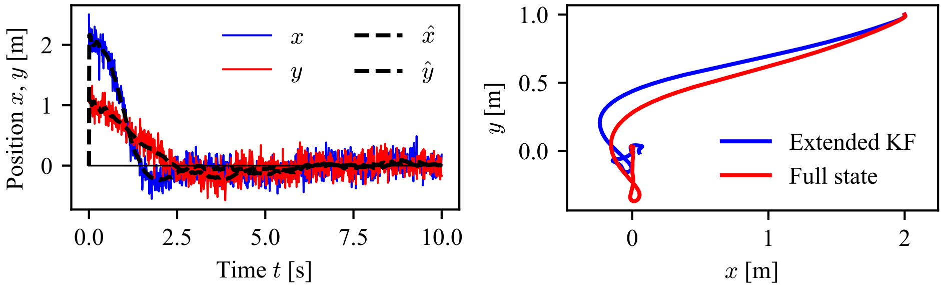
For the (extended) Kalman filter, we model the process disturbances and sensor noise as white noise processes with noise intensities

$$R_V = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}, \qquad R_W = \begin{bmatrix} 2\times 10^{-4} & 0 & 1\times 10^{-5} \\ 0 & 2\times 10^{-4} & 1\times 10^{-5} \\ 1\times 10^{-5} & 1\times 10^{-5} & 1\times 10^{-4} \end{bmatrix}$$

Figure 6.3 shows the response of the system starting from an initial position $(x_0, y_0) = (2, 1)$ and with disturbances and noise with intensity 10-100X smaller than the worst case for which we designed the system.

6.6 Further Reading

There is a vast literature on Kalman filtering and linear quadratic Gaussian (LQG) control theory. The treatment in this chapter follows fairly closely to that of Friedland [Fri04]. A compact treatment of LQG theory is given in the books by Anderson and Moore [AM90], Åström [Åst06b], and Lindquist and Picci [LP15].



(a) Estimated states (b) Full state versus EKF

Figure 6.3: LQR control of the VTOL system with an extended Kalman filter to estimate the state. (a) The x and y positions as a function of time, with dashed lines showing the estimated values from the extended Kalman filter. (b) The xy path of the system with full state feedback (and no noise) versus the controller using the extended Kalman filter.

Exercises

6.1. Show that if we define the estimated state of a random process X as the conditional mean

$$\hat{x}(t) = \mathbb{E}(X(t) \mid y(\tau),\, \tau \leq t)$$

that $\hat{x}$ minimizes

$$\mathbb{E}(\hat{x}(t) - X(t) \mid y(\tau),\, \tau \leq t).$$

6.2. Consider a scalar control system

$$\begin{aligned} \dot{X} &= \lambda X + u + \sigma_v V \\ Y &= X + \sigma_w W, \end{aligned}$$

where V and W are zero-mean, Gaussian white noise processes with covariance 1 and $\sigma_v, \sigma_w > 0$. Assume that the initial value of X is modeled as a Gaussian with mean X_0 and variance $\sigma^2_{X_0}$.

(a) Assume that we initialize a Kalman filter such that the initial covariance starts near a steady state value p^*. Given conditions on λ such that error covariance is locally stable about this solution.

(b) Suppose that V is no longer taken to be white noise and instead has a correlation function given by

$$\rho_V(\tau) = e^{-\alpha|\tau|}, \qquad \alpha > 0.$$

Write down an estimator that minimizes the mean square error of the output under these conditions. You do not need to explicitly solve the resulting equations, just write them down in a form that is similar to an appropriate Kalman filter equation.

6.3. Consider a discrete-time, scalar linear system with dynamics

$$x[k+1] = ax[k] + v[k], \qquad y[k] = x[k] + w[k],$$

where v and w are discrete-time, Gaussian random processes with mean zero and variances 1 and σ^2, respectively. Assume that the initial value of the state has zero mean and variance σ_0^2.

(a) Compute the optimal estimator for the system using y as a (noisy) measurement. Your solution should be in the form of an explicit, time-varying, discrete-time system.

(b) Assume that $a < 1$. Write down an explicit formula for the mean and covariance of the steady-state error of the optimal estimator.

(c) Suppose the mean value of the initial condition is $\mathbb{E}(x[0]) = 1$ and $a = 1$. Determine the optimal steady-state estimator for the system.

6.4. Consider the dynamics of an inverted pendulum whose dynamics are given by

$$\frac{dx}{dt} = \begin{bmatrix} x_2 \\ \sin x_1 - cx_2 + u\cos x_1 \end{bmatrix},$$

where $x = (\theta, \dot\theta)$ and $c > 0$ is the damping coefficient. We assume that we have a sensor that can measure the offset of a point along the inverted pendulum

$$y = r\sin\theta,$$

where $r \in [0, \ell]$ is a point along the pendulum and ℓ is the length of the pendulum. Assume that the pendulum is subject to input disturbances v that are modeled as white noise with intensity σ_v^2 and that the sensor is subject to additive Guassian white noise with noise intensity σ_w^2.

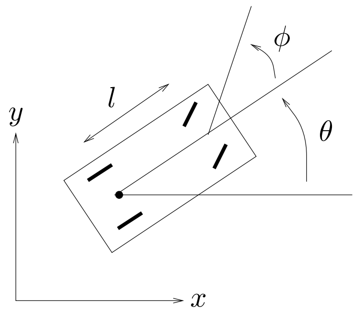
(a) Determine the optimal location of the sensor (r^*) that minimizes the steady state covariance of the error P for system linearized around $x_\mathrm{e} = 0$ and justify your answer.

(b) Show that the Kalman filter gain $L = PC^\mathsf{T} R_w^{-1}$ does not depend on the covariance of the error in $\dot\theta$.

(c) Take $c = 0$ and compute the steady state gain for the Kalman filter as a function of the sensor location r.

Note: for the first two parts it is not necessary to solve equations for the steady state covariance of the error. For the last part, your answer should not require a substantial amount of algebra if you organize your calculations a bit (and set $c = 0$).

6.5. Consider the problem of estimating the position of an autonomous mobile vehicle using a GPS receiver and an IMU (inertial measurement unit). The dynamics of the vehicle are given by



$$
\begin{aligned}
\dot{x} &= \cos\theta\, v \\
\dot{y} &= \sin\theta\, v \\
\dot{\theta} &= \frac{1}{\ell}\tan\delta\, v,
\end{aligned}
$$

We assume that the vehicle has disturbances in the inputs v and δ with standard deviation of up to 10% and noisy measurements from the GPS receiver and IMU.

We consider a trajectory in which the car is driving on a constant radius curve at $v = 10$ m/s forward speed with $\delta = 5°$ for a duration of 10 seconds.

(a) Suppose first that we only have the GPS measurements for the xy position of the vehicle. These measurements give the position of the vehicle with approximately 10 cm accuracy. Model the GPS error as Gaussian white noise with $\sigma = 0.1$ meter in each direction. Design a Kalman filter-based estimator for the system and plot the estimated states versus the actual states. What is the covariance of the estimate at the end of the trajectory?

(b) An IMU can be used to measure angular rates and linear acceleration. For simplicity, we assume that the IMU is able to directly measure the angle of the car with a standard deviation of 1 degree. Design an updated estimator for the system using the GPS and IMU measurements, and plot the estimated states versus the actual states. What is the covariance of the estimate at the end of the trajectory?

Chapter 7

Sensor Fusion

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

Prerequisites. The material in this chapter is designed to be reasonably self-contained, so that it can be used without covering Sections 5.3–5.4 or Chapter 6 of this supplement. We assume rudimentary familiarity with discrete-time linear systems, at the level of the brief descriptions in Chapters 3 and 7 of FBS2e, and discrete-time random processes as described in Section 5.2 of these notes.

7.1 Discrete-Time Stochastic Systems

We begin with a concise overview of stochastic system in discrete time, echoing our development of continuous-time random systems described in Chapter 5. We consider systems of the form

$$X[k+1] = AX[k] + Bu[k] + FV[k], \qquad Y[k] = CX[k] + W[k], \tag{7.1}$$

where $X \in \mathbb{R}^n$ represents the state, $u \in \mathbb{R}^m$ represents the (deterministic) input, $V \in \mathbb{R}^q$ represents process disturbances, $Y \in \mathbb{R}^p$ represents the system output and $V \in \mathbb{R}^p$ represents measurement noise.

As in the case of continuous-time systems, we are interested in the response of the system to the random input $V[k]$. We will assume that V is a Gaussian process with zero mean and correlation function $r_V(k, k+d)$ (or correlation matrix $R_V(k, k+d)$ if V is vector valued). As in the continuous case, we say that a random process is white noise if $r_V(k, k+d) = r_V \delta(d)$ with $\delta(d) = 1$ if $d = 0$ and 0 otherwise. (Note that in the discrete-time case, white noise has finite covariance.)

To compute the response $Y[k]$ of the system, we look at the properties of the state vector $X[k]$. For simplicity, we take $u = 0$ (since the system is linear, we can always add it back in by superposition). Note first that the state at time $k + d$ can be written as

$$\begin{aligned}
X[k+d] &= AX[k+d-1] + FV[x+l-1] \\
&= A(AX[k+d-2] + FV[x+l-2]) + FV[x+l-1] \\
&= A^d X[k] + \sum_{j=1}^{d} A^{j-1} FV[k+d-j].
\end{aligned}$$

The mean of the state at time k is given by

$$\mathbb{E}(X[k]) = A^k \mathbb{E}(E[0]) + \sum_{j=1}^{k} A^{j-1} F \mathbb{E}(V[k-j]) = A^k \mathbb{E}(X[0]).$$

To compute the covariance $R_X(k, k+d)$, we start by computing $R_X(k, k+1)$:

$$\begin{aligned}
R_X(k, k+1) &= \mathbb{E}(X[k] X^{\mathsf{T}}[k+1]) \\
&= \mathbb{E}((A^k x[0] + A^{k-1} F w[0] + \cdots + AFw[k-2] + F[k-1]) \cdot \\
&\qquad (A^{k+1} x[0] + A^k F w[0] + \cdots + Fw[k])^{\mathsf{T}})
\end{aligned}$$

Performing a similar calculation for $R_X(k, k+d)$, it can be shown that

$$\begin{aligned}
R_X(k, k+d) = \big(A^k P[0] (A^{\mathsf{T}})^k + A^{k-1} F R_V[0] F^{\mathsf{T}} (A^{\mathsf{T}})^{k-1} + \dots \\
+ F R_V[k] F^{\mathsf{T}}\big)(A^{\mathsf{T}})^d =: P[k] (A^{\mathsf{T}})^d, \quad (7.2)
\end{aligned}$$

where

$$P[k+1] = AP[k]A^{\mathsf{T}} + FR_V[k]F^{\mathsf{T}}. \tag{7.3}$$

The matrix $P[k]$ is the covariance of the state matrix and we see that its value can be computed *recursively* starting with $P[0] = \mathbb{E}(X[0]X^{\mathsf{T}}[0])$ and then applying equation (7.3). Equations (7.2) and (7.3) are the equivalent of Proposition 5.2 for continuous-time processes. If we additionally assume that V is stationary and focus on the steady state response, we obtain the following.

Proposition 7.1 (Steady state response to white noise)**.** *For a discrete-time, time-invariant, linear system driven by white noise, the correlation matrices for the state and output converge in steady state to*

$$R_X(d) = R_X(k, k+d) = PA^d, \qquad R_Y(d) = CR_X(d)C^{\mathsf{T}},$$

where P satisfies the algebraic equation

$$APA^{\mathsf{T}} + FR_V F^{\mathsf{T}} = 0, \qquad P > 0. \tag{7.4}$$

7.2 Kalman Filters in Discrete Time (FBS2e)

We now consider the optimal estimator in discrete time. This material is presented in FBS2e in slightly simplified (but consistent) form.

Consider a discrete time, linear system with input, having dynamics

$$\begin{aligned} X[k+1] &= AX[k] + Bu[k] + FV[k], \\ Y[k] &= CX[k] + W[k], \end{aligned} \tag{7.5}$$

where $V[k]$ and $W[k]$ are Gaussian, white noise processes satisfying

$$\begin{aligned} &\mathbb{E}(V[k]) = 0 && \mathbb{E}(W[k]) = 0 \\ &\mathbb{E}(V[k]V^\mathsf{T}[j]) = \begin{cases} 0 & k \neq j \\ R_V & k = j \end{cases} && \mathbb{E}(W[k]W^\mathsf{T}[j]) = \begin{cases} 0 & k \neq j \\ R_W & k = j \end{cases} \\ &\qquad\qquad \mathbb{E}(V[k]W^\mathsf{T}[j]) = 0. \end{aligned} \tag{7.6}$$

We assume that the initial condition is also modeled as a Gaussian random variable with

$$\mathbb{E}(X[0]) = x_0 \qquad \mathbb{E}(X[0]X^\mathsf{T}[0]) = P[0]. \tag{7.7}$$

We wish to find an estimate $\hat{X}[k]$ that gives the minimum mean square error (MMSE) for $\mathbb{E}((\hat{X}[k] - X[k])(\hat{X}[k] - X[k])^\mathsf{T})$ given the measurements $\{Y[l] : 0 \leq l \leq k\}$. We consider an observer of the form

$$\hat{X}[k+1] = A\hat{X}[k] + Bu[k] - L[k](C\hat{X}[k] - Y[k]). \tag{7.8}$$

The following theorem summarizes the main result.

Theorem 7.2. *Consider a random process $X[k]$ with dynamics* (7.5) *and noise processes and initial conditions described by equations* (7.6) *and* (7.7)*. The observer gain L that minimizes the mean square error is given by*

$$L[k] = AP[k]C^\mathsf{T}(R_W + CP[k]C^\mathsf{T})^{-1},$$

where

$$\begin{aligned} P[k+1] &= AP[k]A^\mathsf{T} + FR_VF^\mathsf{T} - AP[k]C^\mathsf{T}R_\epsilon^{-1}CP[k]A^\mathsf{T} \\ P[0] &= \mathbb{E}(X[0]X^\mathsf{T}[0]). \end{aligned} \tag{7.9}$$

Proof. We wish to minimize the mean square of the error, $\mathbb{E}((\hat{X}[k] - X[k])(\hat{X}[k] - X[k])^\mathsf{T})$. We will define this quantity as $P[k]$ and then show that it satisfies the recursion given in equation (7.9). Let $E[k] = C\hat{X}[k] - Y[k]$ be the residual between the measured output and the estimated output. By definition,

$$\begin{aligned} P[k+1] &= \mathbb{E}(E[k+1]E^\mathsf{T}[k+1]) \\ &= (A - LC)P[k](A - LC)^\mathsf{T} + FR_VF^\mathsf{T} + LR_WL^\mathsf{T} \\ &= AP[k]A^\mathsf{T} - AP[k]C^\mathsf{T}L^\mathsf{T} - LCP[k]A^\mathsf{T} + \\ &\qquad\qquad L(R_W + CP[k]C^\mathsf{T})L^\mathsf{T} + FR_VF^\mathsf{T}. \end{aligned}$$

Letting $R_\epsilon = (R_W + CP[k]C^\mathsf{T})$, we have

$$\begin{aligned}P[k+1] &= AP[k]A^\mathsf{T} - AP[k]C^\mathsf{T}L^\mathsf{T} - LCP[k]A^\mathsf{T} + LR_\epsilon L^\mathsf{T} + FR_VF^\mathsf{T} \\ &= AP[k]A^\mathsf{T} + \big(L - AP[k]C^\mathsf{T}R_\epsilon^{-1}\big)R_\epsilon\big(L - AP[k]C^\mathsf{T}R_\epsilon^{-1}\big)^\mathsf{T} \\ &\quad - AP[k]C^\mathsf{T}R_\epsilon^{-1}CP[k]A^\mathsf{T} + FR_VF^\mathsf{T}.\end{aligned}$$

In order to minimize this expression, we choose $L = AP[k]C^\mathsf{T}R_\epsilon^{-1}$ and the theorem is proven. □

Note that the Kalman filter has the form of a *recursive* filter: given $P[k] = \mathbb{E}(E[k]E[k]^\mathsf{T})$ at time k, can compute how the estimate and covariance *change*. Thus we do not need to keep track of old values of the output. Furthermore, the Kalman filter gives the estimate $\hat{X}[k]$ *and* the covariance $P[k]$, so we can see how reliable the estimate is. It can also be shown that the Kalman filter extracts the maximum possible information about output data: the correlation matrix for the estimation error of the filter is

$$R_E[j,k] = R\delta_{jk}.$$

In other words, the error is a white noise process, so there is no remaining dynamic information content in the error.

In the special case when the noise is stationary (R_V, R_W constant) and *if* $P[k]$ converges, then the observer gain is constant:

$$L = APC^\mathsf{T}(R_W + CPC^\mathsf{T})^{-1},$$

where P satisfies

$$P = APA^\mathsf{T} + FR_VF^\mathsf{T} - APC^\mathsf{T}\big(R_W + CPC^\mathsf{T}\big)^{-1}CPA^\mathsf{T}.$$

We see that the optimal gain depends on both the process noise and the measurement noise, but in a nontrivial way. Like the use of LQR to choose state feedback gains, the Kalman filter permits a systematic derivation of the observer gains given a description of the noise processes. The solution for the constant gain case is solved by the `dlqe` command in MATLAB and python-control.

7.3 Predictor-Corrector Form

The Kalman filter can be written in a two-step form by separating the correction step (where we make use of new measurements of the output) and the prediction step (where we compute the expected state and covariance at the next time instant).

We make use of the notation $\hat{X}[k|j]$ to represent the estimated state at time instant k given the information up to time j (where typically $j = k-1$). Using this notation, the filter can be solved using the following algorithm:

Step 0: Initialization.

$$\begin{aligned}k &= 1, \\ \hat{X}[0|0] &= \mathbb{E}(X[0]), \\ P[0|0] &= \mathbb{E}(X[0]X^\mathsf{T}[0]).\end{aligned}$$

Step 1: Prediction. Update the estimates and covariance matrix to account for all data taken up to time $k-1$:

$$\begin{aligned}\hat{X}[k|k{-}1] &= A\hat{X}[k{-}1|k{-}1] + Bu[k-1],\\ P[k|k{-}1] &= AP[k{-}1|k{-}1]A^{\mathsf{T}} + FR_V[k-1]F^{\mathsf{T}}.\end{aligned}$$

Step 2: Correction. Correct the estimates and covariance matrix to account for the data taken at time step k:

$$\begin{aligned}\tilde{L}[k] &= P[k|k{-}1]C^{\mathsf{T}}(R_W + CP[k|k{-}1]C^{\mathsf{T}})^{-1},\\ \hat{X}[k|k] &= \hat{X}[k|k{-}1] - \tilde{L}[k](C\hat{X}[k|k{-}1] - Y[k]),\\ P[k|k] &= P[k|k{-}1] - \tilde{L}[k]CP[k|k{-}1].\end{aligned}$$

(We use $\tilde{L}[k]$ to distinguish the optimal gain in this form from that given in Theorem 7.2, as discussed briefly at the end of this section.)

Step 3: Iterate. Set k to $k+1$ and repeat steps 1 and 2.

Note that the correction step reduces the covariance by an amount related to the relative accuracy of the measurement, while the prediction step increases the covariance by an amount related to the process disturbance.

This form of the discrete-time Kalman filter is convenient because we can reason about the estimate in the case when we do not obtain a measurement on every iteration of the algorithm. In this case, we simply update the prediction step (increasing the covariance) until we receive new sensor data, at which point we call the correction step (decreasing the covariance).

The following lemma will be useful in the sequel:

Lemma 7.3. *The optimal gain* $\tilde{L}[k]$ *satisfies*

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

Proof. $\tilde{L}[k]$ is defined as

$$\tilde{L}[k] = P[k|k{-}1]C^{\mathsf{T}}(R_W + CP[k|k{-}1]C^{\mathsf{T}})^{-1}.$$

Multiplying through by the inverse term on the right and expanding, we have

$$\begin{aligned}\tilde{L}[k](R_W + CP[k|k{-}1]C^{\mathsf{T}}) &= P[k|k{-}1]C^{\mathsf{T}},\\ \tilde{L}[k]R_W + \tilde{L}[k]CP[k|k{-}1]C^{\mathsf{T}} &= P[k|k{-}1]C^{\mathsf{T}},\end{aligned}$$

and hence

$$\begin{aligned}\tilde{L}[k]R_W &= P[k|k{-}1]C^{\mathsf{T}} - \tilde{L}[k]CP[k|k{-}1]C^{\mathsf{T}},\\ &= (I - \tilde{L}[k]C)P[k|k{-}1]C^{\mathsf{T}} = P[k|k]C^{\mathsf{T}}.\end{aligned}$$

The desired results follows by multiplying on the right by R_W^{-1}. □

It can be shown that the predictor-corrector form matches the form in Theorem 7.2 if we define $\hat{x}[k] = \hat{x}[k|k-1]$, $P[k] = P[k|k-1]$ and $\tilde{L}[k] = AL[k]$.

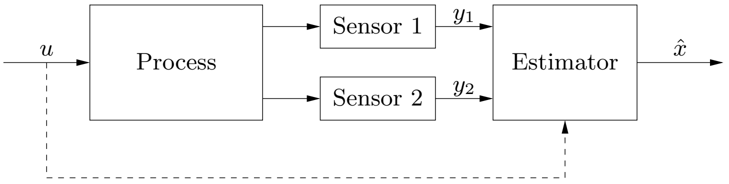


Figure 7.1: Sensor fusion. Multiple sensors report on data from a single process. The estimator fusions this information from the sensors to obtain an estimate of the state of the system. Depending on the use case, the input (dashed line) may not be available to the estimator.

7.4 Sensor Fusion

We now return to the main topic of the chapter: sensor fusion. Consider the situation described in Figure 7.1, where we have an input/output dynamical system with multiple sensors capable of taking measurements. The problem of sensor fusion involves deciding how to best combine the measurements from the individual sensors in order to accurately estimate the process state X. Since different sensors may have different noise characteristics, evidently we should combine the sensors in a way that places more weight on sensors with lower noise. In addition, in some situations we may have different sensors available at different times, so that not all information is available on each measurement update.

While sensor fusion can be used for estimation of the state of a system being controlled, another common application is to sense the state of a system in the environment. A difference for this use case is that the input to the system in the environment is often not available, requiring the estimator to use a model for the system in which the input is modeled by a random process.

Sensor weighting

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

$$Y^i = C^i X + W^i, \qquad i = 1, \dots, p,$$

where the superscript i corresponds to the specific sensor. Let W^i be a zero mean, white noise process with covariance $\sigma_i^2 = R_{W^i}(0)$. It follows from Lemma 7.3 that

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

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

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

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

Information filters

An alternative formulation of the Kalman filter is to make use of the inverse of the covariance matrix, called the *information matrix*, to represent the error of the estimate. It turns out that writing the state estimator in this form has several advantages both conceptually and when implementing distributed computations. This form of the Kalman filter is known as the *information filter*.

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

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

In this form, it can be shown that the correction step of the Kalman filter for the multi-sensor case can be written as

$$\begin{aligned} I[k|k] &= I[k|k{-}1] + \sum_{i=1}^{p} (C^i)^\mathsf{T} R_{W^i}^{-1}[k|k] C^i, \\ \hat{Z}[k|k] &= \hat{Z}[k|k{-}1] + \sum_{i=1}^{p} (C^i)^\mathsf{T} R_{W^i}^{-1}[k|k] Y^i. \end{aligned}$$

The advantage of using the information filter version of the equation is that it allows a simple addition operation for the correction step, corresponding to adding the "information" obtained through the acquisition of new data. We also see the clear relationship between the information content in each sensor channel and the inverse covariance of that sensor.

Another feature of the information filter formulation is that it allows some efficiencies when implementing distributed estimation across networks. In particular, the information carried in the individual sensors can be simply added together through the updates of $I[k|k{-}1]$. This is helpful especially when the sensors have variable sampling rate and the measurement packets arrive at different times. New information is incorporated whenever it arrives and then a global update of $I[k|k–1]$ at a centralized node is used to integrate all sensor measurements (which can the be rebroadcast out to the sensors). The information form also makes clear how to handle missing data: if no data arrives for a given sensor then no information is added and only the time update is applied, hence the measurement update is skipped.

7.5 Implementation in Python

Steady state gains for a discrete time Kalman filter can be computed using the `dlqe` command:

```
L, P, E = ct.dlqe(A, F, C, Qv, Qw, C=C_sensors),
```

where `A`, `F`, and `C` are the (discrete-time) system matrices and `Qv` and `Qw` are the disturbance and noise covariance matrices.

The `create_estimator_iosystem` function can be used to implement a discrete-time Kalman filter in the form given in Theorem 7.2. This function creates an input/output system that implements a discrete-time state estimator of the form

$$\begin{aligned}\hat{x}[k+1] &= Ax[k] + Bu[k] - L(C\hat{x}[k] - y[k]),\\ P[k+1] &= APA^T + FQ_NF^T - APC^TR_\epsilon^{-1}CPA,\\ L &= APC^TR_\epsilon^{-1}, \qquad R_\epsilon = R_N + CPC^T.\end{aligned}$$

To create the estimator system, using the command

```
estim = ct.create_estimator_iosystem(sys, Qv, Qw)
```

where `sys` is the process dynamics and `Qv` and `Qw` are the covariance of the disturbance noise and sensor noise. The function returns the estimator `estim` as I/O system with a parameter `correct` that can be used to turn off the correction term in the estimation (for forward predictions).

Sensor fusion can be implemented by passing `create_estimator_iosystem` a system with full state output and then using the `C` keyword to define the measured values to be used by the estimator:

```
estim = ct.create_estimator_iosystem(sys, Qv, Qw, C=C_sensors)
```

The matrix `C_sensors` can have as many rows as there are sensors, with the matrix `Qw` describing the noise covariance (and cross-correlation, if diagonal entries are non-zero).

As described in Section 6.4, the `create_statefbk_iosystem` function can be used to create an I/O system that includes an estimator and state feedback to form a dynamic compensator.

7.6 Additional Topics

Unscented Kalman filter[1]

Consider a *nonlinear* system

$$\begin{aligned}\dot{X} &= f(X,u,V), \qquad && X \in \mathbb{R}^n, u \in \mathbb{R}^m,\\ Y &= h(X) + W, && Y \in \mathbb{R}^p,\end{aligned} \tag{7.10}$$

where V and W are Gaussian white noise processes with covariance matrices R_V and R_W. We saw in Chapter 6 how to extend the Kalman filter to this case by using the local linearization of the dynamics. In some cases this approximation is not a very good one, especially if there are strong nonlinearities in the update function f or output function h.

ChatGPT prompt: Explain how an unscented Kalman filter works, in the form of a textbook.

---

[1] This section was written by ChatGPT on 20 Feb 2023 and (lightly) edited by Richard Murray.

Response: The unscented Kalman filter (UKF) is a variant of the Kalman filter that can be used for nonlinear systems. In this [subsection], I'll assume that you're already familiar with the Kalman filter and its notation.

The basic idea behind the UKF is to approximate the probability distribution of the state variables using a set of sample points, called sigma points. These sample points are carefully chosen to capture the mean and covariance of the distribution, and are propagated through the nonlinear system to obtain a predicted distribution for the next time step.

At each time step, the UKF goes through two main steps: the prediction step and the update step.

Prediction step. In the prediction step, the UKF generates a set of sigma points from the current estimate of the state distribution. These sigma points are chosen to capture the mean and covariance of the distribution. The number of sigma points required is typically twice the number of state variables plus one.

Once the sigma points are generated, they are propagated through the nonlinear system dynamics to obtain a predicted mean and covariance for the next time step. This is done by passing each sigma point through the nonlinear function, and then computing the mean and covariance of the resulting set of points.

The predicted mean and covariance are then used to generate a predicted distribution for the state variables at the next time step.

Update step. In the update step, the UKF uses new measurements of the system to correct the predicted distribution generated in the prediction step.

First, a set of sigma points is generated from the predicted distribution, in the same way as in the prediction step. These sigma points are then propagated through the measurement function to obtain the predicted measurement values corresponding to each sigma point.

The predicted measurement mean and covariance are then computed from the resulting set of predicted measurement values. This predicted measurement distribution is compared to the actual measured values, and the difference is used to correct the predicted state distribution.

Finally, the corrected state distribution is used as the new estimate for the state variables at the current time step.

In summary, the UKF approximates the distribution of the state variables using a set of carefully chosen sample points, and propagates these points through the nonlinear system dynamics and measurement function to generate predicted distributions. The predicted distributions are then updated with new measurements to obtain a corrected estimate of the state variables at each time step.

ChatGPT prompt: Show me the equations for an unscented Kalman filter.

1. Initialization: initialize state estimate and covariance matrix:
   - $\hat{x}_0$: initial state estimate
   - P_0 : initial covariance matrix
2. Prediction step:
   - Generate sigma points:

    - $X_0 = \hat{x}_k$,
    - $X_i = \hat{x}_k + \left(\sqrt{(n+\lambda)P_k}\right)_i, i = 1, ..., n$,
    - $X_i = \hat{x}_k - \left(\sqrt{(n+\lambda)P_k}\right), i = n+1, ..., 2n$.
  - Propagate sigma points through nonlinear function $f(x)$:
    - $Y_i = f(X_i), i = 0, 1, ..., 2n$.
  - Compute predicted state mean and covariance:
    - $\hat{x}_{k|k-1} = \sum_{i=0}^{2n} w_i^m Y_i$,
    - $P_{k|k-1} = \sum_{i=0}^{2n} w_i^c (Y_i - \hat{x}_{k|k-1})(Y_i - \hat{x}_{k|k-1})^T + Q_k$,

    where n is the dimensionality of the state, λ is a scaling parameter, P_k is the covariance matrix of the state at time k, w_i^m and w_i^c are weight coefficients that depend on n and λ, and Q_k is the process noise covariance matrix.

3. Update step:

  - Generate sigma points from predicted state distribution:
    - $X_0 = \hat{x}_{k|k-1}$,
    - $X_i = \hat{x}_{k|k-1} + \left(\sqrt{(n+\lambda)P_{k|k-1}}\right)_i, i = 1, ..., n$,
    - $X_i = \hat{x}k|k-1 - \left(\sqrt{(n+\lambda)P_{k|k-1}}\right)_{i-n}, i = n+1, ..., 2n$.
  - Propagate sigma points through measurement function $h(x)$:
    - $Z_i = h(X_i), i = 0, 1, ..., 2n$.
  - Compute predicted measurement mean and covariance:
    - $\hat{z}_{k|k-1} = \sum_{i=0}^{2n} w_i^m Z_i$,
    - $S_k = \sum_{i=0}^{2n} w_i^c (Z_i - \hat{z}k|k-1)(Z_i - \hat{z}k|k-1)^T + R_k$,

    where R_k is the measurement noise covariance matrix.
  - Compute cross-covariance matrix:
    - $P_{x,z,k} = \sum_{i=0}^{2n} w_i^c (X_i - \hat{x}k|k-1)(Z_i - \hat{z}k|k-1)^T$.
  - Compute Kalman gain:
    - $K_k = P_{x,z,k} S_k^{-1}$.

    Compute updated state estimate and covariance:
    - $\hat{x}k = \hat{x}k|k-1 + K_k(z_k - \hat{z}_{k|k-1})$,
    - $P_k = P_{k|k-1} - K_k S_k K_k^T$.

    where z_k is the measurement at time k, $\hat{z}_{k|k-1}$ is the predicted measurement, K_k is the Kalman gain, $P_{x,z,k}$ is the cross-covariance matrix, and P_k is the updated covariance matrix.

The weight coefficients are defined as follows:

- $w_0^m = \frac{\lambda}{n+\lambda}$,
- $w_0^c = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta)$,
- $w_i^m = w_i^c = \frac{1}{2(n+\lambda)}, i = 1, ..., 2n$,

where α, β, and κ are tuning parameters that determine the spread of the sigma points. Typically, $\alpha = 0.9$, $\beta = 2$, and $\kappa = 3 - n$ are used in practice.

Ensemble Kalman filtering[2]

ChatGPT prompt: Write a section for a textbook on the ensemble Kalman filter, with equations.

Introduction. The ensemble Kalman filter (EnKF) is a popular data assimilation algorithm that is widely used in numerical weather prediction, oceanography, and other fields. It is a variant of the Kalman filter that replaces the Gaussian assumptions about the prior and posterior probability distributions with a set of Monte Carlo samples, or "ensemble members". The EnKF is particularly useful in high-dimensional problems where the computational cost of a full Kalman filter update becomes prohibitive.

Algorithm. The EnKF algorithm proceeds as follows:

1. Initialization: Generate an initial ensemble of N state vectors x_0^i, $i = 1, \dots, N$, with mean $\hat{x}_0$ and covariance matrix P_0.
2. Forecast step: Perturb each ensemble member using a random draw from a distribution with mean zero and covariance matrix Q_k, the process noise:
   - $x_k^i = f(x_{k-1}^i) + \epsilon_k^i$

   Evaluate the forecast ensemble mean and covariance:
   - $\hat{x}_{k|k-1} = \frac{1}{N} \sum_{i=1}^N x_k^i$
   - $P_{k|k-1} = \frac{1}{N-1} \sum_{i=1}^N (x_k^i - \hat{x}_{k|k-1})(x_k^i - \hat{x}_{k|k-1})^T + Q_k$
3. Analysis step: Obtain a set of M observations y_k and the associated observation operator h.
   - Perturb each ensemble member using a random draw from a distribution with mean zero and covariance matrix R_k, the observation error:
     - $y_k^i = h(x_k^i) + \delta_k^i$

     Evaluate the forecast ensemble mean and covariance:
     - $\hat{y}_{k|k-1} = \frac{1}{M} \sum_{i=1}^M y_k^i$
     - $S_k = \frac{1}{M-1} \sum_{i=1}^M (y_k^i - \hat{y}_{k|k-1})(y_k^i - \hat{y}_{k|k-1})^T + R_k$
   - Compute the cross-covariance matrix between the forecast state and the observations:
     - $P_{k|k-1} = \frac{1}{N-1} \sum_{i=1}^N (x_k^i - \hat{x}_{k|k-1})(y_k^i - \hat{y}_{k|k-1})^T$
   - Compute the Kalman gain:
     - $K_k(y_k - \hat{y}_{k|k-1})$
   - Update the state ensemble using the Kalman gain:
     - $x_k^i = x_{k|k-1}^i + K_k(y_k - h(x_{k|k-1}^i))$
   - Evaluate the updated ensemble mean and covariance:
     - $\hat{x}_k = \frac{1}{N} \sum_{i=1}^N x_k^i$

---

[2] This section was written by ChatGPT on 20 Feb 2023 and (lightly) edited by Richard Murray.

- $P_k = \frac{1}{N-1}\sum_{i=1}^{N}(x_k^i - \hat{x}_k)(x_k^i - \hat{x}_k)^T$

  where ϵ_k^i and δ_k^i are random vectors drawn from the process and observation noise distributions, respectively.

The EnKF algorithm also requires specification of the process noise covariance matrix Q_k and the observation error covariance matrix R_k. These matrices can be estimated using statistical methods or by tuning them through trial and error.

One advantage of the EnKF over the traditional Kalman filter is that it is easily parallelizable. The forecast step and analysis step can be performed independently for each ensemble member, allowing the algorithm to take advantage of distributed computing architectures. However, the EnKF can suffer from sampling errors, particularly in the presence of nonlinearities or non-Gaussian distributions. Various modifications to the basic EnKF algorithm have been proposed to address these issues, such as the Local ensemble transform Kalman filter (LETKF) and the ensemble square root filter (ESRF).

Moving horizon estimation

The various extensions and variations of the Kalman filter equations all assume that the system disturbances and measurement noise can be modeled by Gaussian processes, in which the (potentially time-varying) mean and covariance of the signals are sufficient to model the underlying sources of uncertainty. In many cases, this is not a good assumption, for example when there are bounded, non-symmetric, or multi-modal sources of disturbances and noise.

Consider a nonlinear system with discrete time dynamics of the form

$$X[k+1] = f(X[k], u[k], V[k]), \qquad Y[k] = h(X[k]) + W[k], \tag{7.11}$$

where $X[k] \in \mathbb{R}^n$, $u[k] \in \mathbb{R}^m$, and $Y[k] \in \mathbb{R}^p$, and $V[k] \in \mathbb{R}^q$ and $W[k] \in \mathbb{R}^p$ represent random processes that are not necessarily Gaussian white noise processes. The estimation problem that we wish to solve is to find the estimate $\hat{x}[\cdot]$ that matches the measured outputs $y[\cdot]$ with "likely" disturbances and noise.

For a fixed horizon of length N, this problem can be formulated as an optimization problem where we define the likelihood of a given estimate (and the resulting noise and disturbances predicted by the model) as a cost function. Suppose we model the likelihood using a conditional probability density function $p\bigl(x[0], \dots, x[N] \mid y[0], \dots, y[N-1]\bigr)$. Then we can pose the state estimation problem as

$$\hat{x}[0], \dots, \hat{x}[N] = \arg \max_{\hat{x}[0], \dots, \hat{x}[N]} p\bigl(\hat{x}[0], \dots, \hat{x}[N] \mid y[0], \dots, y[N-1]\bigr) \tag{7.12}$$

subject to the constraints given by equation (7.11). The result of this optimization gives us the estimated state for the previous N steps in time, including the "current" time $x[N]$. The basic idea is thus to compute the state estimate that is most consistent with our model and penalize the noise and disturbances according to how likely the are (based on a some sort of stochastic system model for each).

Given a solution to this fixed horizon, optimal estimation problem, we can create an estimator for the state over all times by applying repeatedly applying the optimization problem (7.12) over a moving horizon. At each time k, we take the measurements for the last N time steps along with the previously estimated state at the start of the horizon, $x[k-N]$ and reapply the optimization in equation (7.12). This approach is known as a *moving horizon estimator* (MHE).

The formulation for the moving horizon estimation problem is very general and various situations can be captured using the conditional probability function $p(x[0], \dots, x[N] \mid y[0], \dots, y[N-1]$. We start by noting that if the disturbances are independent of the underlying states of the system, we can write the conditional probability as

$$p\bigl(x[0], \dots, x[N] \mid y[0], \dots, y[N-1]\bigr) = \\ p_{X[0]}(x[0]) \prod_{k=0}^{N-1} p_V\bigl(y[k] - h(x[k])\bigr)\, p\bigl(x[k+1] \mid x[k]\bigr).$$

This expression can be further simplified by taking the log of the expression and maximizing the function

$$\log p_{X[0]}(x[0]) + \sum_{k=0}^{N-1} \log p_W\bigl(y[k] - h(x[k])\bigr) + \log p_V(v[k]). \tag{7.13}$$

The first term represents the likelihood of the initial state, the second term captures the likelihood of the noise signal, and the final term captures the likelihood of the disturbances.

If we return to the case where V and W are modeled as Gaussian processes, then it can be shown that maximizing equation (7.13) is equivalent to solving the optimization problem given by

$$\min_{x[0], \{v[0], \dots, v[N-1]\}} \|x[0] - \bar{x}[0]\|_{P_0^{-1}} + \sum_{k=0}^{N-1} \|y[k] - h(x_k)\|^2_{R_W^{-1}} + \|v[k]\|^2_{R_V^{-1}}. \tag{7.14}$$

Note that here we only compute the estimated initial state $\hat{x}[0]$, but we can now reconstruct the entire history of estimated states using the system dynamics:

$$\hat{x}[k+1] = F(\hat{x}[k], u[k], v[k]), \quad k = 0, \dots, N-1,$$

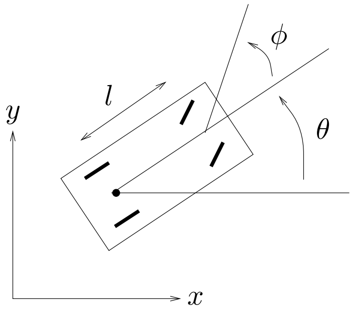
and we can implement the estimator in receding horizon fashion by repeatedly solving the optimization of a window of length N backwards in time.

One of the simpler cases where the moving horizon formulation is useful is when we have *a priori* knowledge that our disturbances are bounded. In this case, we simply add a constraint in the optimization in equation (7.14), for example requiring that $v[k] \in [v_{\min}, v_{\max}]$.

This functionality is implemented in python-control using the `solve_oep()` and `create_mhe_iosystem()` functions. An example demonstrating the implementation is available via the course website.

Exercises

7.1. Consider the problem of estimating the position of an autonomous mobile vehicle using a GPS receiver and an IMU (inertial measurement unit). The continuous time dynamics of the vehicle are given by



$$\dot{x} = \cos\theta\, v$$
$$\dot{y} = \sin\theta\, v$$
$$\dot{\theta} = \frac{1}{\ell}\tan\delta\, v,$$

We assume that the vehicle is disturbance free, but that we have noisy measurements from the GPS receiver and IMU and an initial condition error.

(a) Rewrite the equations of motion in discrete time, assuming that we update the dynamics at a sample time of $h = 0.005$ sec and that we can take $\dot{x}$ to be roughly constant over that period. Run a simulation of your discrete time model from initial condition $(0, 0, 0)$ with constant input $\delta = \pi/8$, $v = 5$ and compare your results with the continuous time model.

(b) Suppose that we have a GPS measurement that is taken every 0.1 seconds and an IMU measurement that is taken every 0.01 seconds. Write a MATLAB program that that computes the discrete time Kalman filter for this system, using the same disturbance, noise and initial conditions as Exercise 6.5.

7.2. Consider the problem of estimating the position of a car operating on a road whose dynamics are modeled as described in Example 2.3. We assume that the car is executing a lane change manuever and we wish to estimate its position using a set of available sensors:

- A stereo camera pair, which relatively poor longitudinal (x) accuracy but good lateral position (y) accuracy. We model the covaraiance of the sensor noise as $R_{\text{lat}} = \text{diag}(1, 0.1)$.
- An automotive grade radar, which has good longitudinal position (x) accuracy but poor lateral (y) accuracy, with $R_{\text{lon}} = \text{diag}(0.1, 1)$.
- We assume the radar can also measure the longitudinal velocity ($\dot{x}$) as an optional measurement, with $R_{\text{vel}} = 1$.

In this problem we assume that the detailed model of the system is not known and also that the inputs to the vehicle (throttle and steering) are not known. We use a variety of system models to explore how these different measurements can be fused to obtain estimates and predictions of the vehicle position.

(a) Consider a model of the vehicle consisting of a particle in 2D, with the velocity of particle in the x and y direction taken as the input:

$$\dot{x} = u_1, \qquad \dot{y} = u_2$$

A discrete-time version of the system dynamics is given by

$$x[k+1] = x[k] + u_1[k] * T_{\text{s}}, \qquad y[k+1] = y[k] + u_2[k] * T_{\text{s}},$$

where $T_s = 0.1$ s is the sampling time between sensor measurements.

Construct an estimator for the system using a combination of the stereo pair and the radar (position only). Estimate the state and covariance of the system during the lane change manuever from Example 2.3 and predict the state for the next 4 seconds.

(b) Assume now that we now add (noisy) measurement of the velocity from the radar as an approximation of the input u_1. Update your Kalman filter to utilize this measurement (with no filtering), and replot the estimate and prediction for the system.

(c) To provide a better prediction, we can increase the complexity of our model so that it includes the velocity of the vehicle as a state, allowing us to model the acceleration as the input. In continuous time, this model is given by

$$\ddot{x} = u_1, \qquad \dot{y} = u_2$$

(note that we are still modeling the lateral position using a single integrator).

Convert this model to discrete time and construct an estimator for the system using a combination of the stereo pair and the radar (position and velocity). Estimate the state and covariance of the system during the lane change manuever and predict the state for the next 4 seconds.

Note: in this problem you have quite a bit of freedom in how you model the disturbances, which should model the unknown inputs to the vehicle being observed. Make sure to provide some level of justification for how you chose these disturbances.

7.3. The form of the optimal feedback for a discrete time Kalman filter differes slightly depending on whether we use the form in Theorem 7.2 or the predictor-corrector form in Section 7.3.

(a) Show that the predictor-corrector form of the optimal estimator for a linear process driven by white noise matches the form in Theorem 7.2 if we define $\hat{x}[k] = \hat{x}[k|k-1]$, $P[k] = P[k|k=1]$ and $\tilde{L}[k] = AL[k]$.

(b) Alternatively, show that if we formulate the optimal estimate using an estimator of the form

$$\hat{X}[k+1] = A\hat{X}[k] + L[k](Y[k+1] - CA\hat{X}[k])$$

that we recover the update law in the predictor-corrector form.

7.4. The unscented Kalman filter (UKF) equations created by ChatGPT are convincing, but they are incorrect. Find and fix the errors.

7.5. Consider the discrete time linear system

$$X[k+1] = \begin{bmatrix} 0.5 & 1 & 0 \\ 0 & -0.8 & 1 \\ 0 & 0 & 0.5 \end{bmatrix} X[k] + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} V, \qquad Y[k] \quad = \sin(\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} X) + W,$$

where V is a discrete time, white noise process with covariance 0.01 and W is a discrete time, white noise process with covariance 10^{-4}. Let $u[k] = \sin(2\pi k/5)$ and assume that $P[0] = \mathbb{E}(X[0]X^{\mathsf{T}}[0]) = 0.5I$.

(a) Construct an optimal estimator (Kalman filter) for the system linearized about the origin and plot the state estimate and covariance of each state (using error bars) for a trajectory starting at the origin and for a duration of $K = 20$.

(b) Construct a moving horizon estimator for the system using time windows of length $N = 1$, $N = 3$, and $N = 6$ and compare the state estimate to the result from (a).

(c) Change the penalty on the initial state in the window from $P[0]$ to the value of P obtained from the steady state estimator for the linearized system and compare the performance for the three horizon lengths in (b).

(d) Suppose that the disturbance V is constrained to take values in the the range -0.1 to 0.1. Compare the steady state optimal estimator for the linearized system to a moving horizon estimator with appropriate horizon that takes the constraints into account. How does the moving horizon estimator compare to the linearized estimator?

Bibliography

[AF06] M. Athans and P. L. Falb. *Optimal Control: An Introduction to the Theory and Its Applications*. Dover, 2006. Originally published in 1963.

[AM90] B. D. O. Anderson and J. B. Moore. *Optimal Control Linear Quadratic Methods*. Prentice Hall, Englewood Cliffs, NJ, 1990. Republished by Dover Publications, 2007.

[ÅM08] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008. Available at `http://fbsbook.org`.

[ÅM21] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, second edition, 2021. Available at `http://fbsbook.org`.

[Åst06a] K. J. Åström. *Introduction to Stochastic Control Theory*. Dover, New York, 2006. Originally published by Academic Press, New York, 1970.

[Åst06b] K. J. Åström. *Introduction to Stochastic Control Theory*. Dover, New York, 2006. Originally published by Academic Press, New York, 1970.

[BBM17] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[BBvB$^+$01] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mallor, Ken Shwaber, and Jeff Sutherland. The Agile Manifesto. Technical report, The Agile Alliance, 2001. Available at `http://agilemanifesto.org`.

[BdTH$^+$07] J. W. Burdick, N. du Toit, A. Howard, C. Looman, J. Ma, R. M. Murray, and T. Wongpiromsarn. Sensing, navigation and reasoning technologies for the darpa urban challenge. Technical report, California Institute of Technology, 2007. Available from `https://apps.dtic.mil/sti/citations/ADA475619`.

[BH75] A. E. Bryson, Jr. and Y.-C. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Wiley, New York, 1975.

[Bro81] R. W. Brockett. Control theory and singular Riemannian geometry. In *New Directions in Applied Mathematics*, pages 11–27. Springer-Verlag, New York, 1981.

[CHHR22] N. Correll, B. Hayes, C. Heckman, and A. Roncone. *Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms*. MIT Press, Cambridge, MA, 1st edition, 2022.

[dB78] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[Dra55] C. S. Draper. Flight control. *Journal Royal Aeronautical Society*, 59(July):451–477, 1955. 45th Wilber Wright Memorial Lecture.

[FGM$^+$21] S. Fuller, B. Greiner, J. Moore, R. Murray, R. van Paassen, and R. Yorke. The python control systems library (python-control). In *Proc. IEEE Control and Decision Conference*, 2021.

[FLMR92] M. Fliess, J. Levine, P. Martin, and P. Rouchon. On differentially flat nonlinear systems. *Comptes Rendus des Séances de l'Académie des Sciences*, 315:619–624, 1992. Serie I.

[FLMR95] M. Fliess, J. Levine, P. Martin, and P. Rouchon. Flatness and defect of nonlinear systems: Introductory theory and examples. *International Journal of Control*, 61(6):1327–1361, 1995.

[Fri04] B. Friedland. *Control System Design: An Introduction to State Space Methods*. Dover, New York, 2004.

[GMSW] P. E. Gill, W. Murray, M. A. Saunders, and M. Wright. *User's Guide for NPSOL 5.0: A Fortran Package for Nonlinear Programming*. Systems Optimization Laboratory, Stanford University, Stanford, CA 94305.

[GS01] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes*. Oxford University Press, third edition, 2001.

[HO01] J. Hauser and H. Osinga. On the geometry of optimal control: The inverted pendulum example. In *American Control Conference*, 2001.

[Isi89] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag, 2nd edition, 1989.

[Jad01] A. Jadbabaie. *Nonlinear Receding Horizon Control: A Control Lyapunov Function Approach*. PhD thesis, California Institute of Technology, Control and Dynamical Systems, 2001.

[JSK99] M. Jankovic, R. Sepulchre, and P. V. Kokotović. CLF based designs with robustness to dynamic input uncertainties. *Systems Control Letters*, 37:45–54, 1999.

[JYH01] A. Jadbabaie, J. Yu, and J. Hauser. Unconstrained receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 46(5):776–783, 2001.

[Kal64] R. E. Kalman. When is a linear control system optimal? *J. Basic Engrg. Trans. ASME Ser. D*, 86:51–60, 1964.

[Kel17] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[KKK95] M. Krstić, I. Kanellakopoulos, and P. Kokotović. *Nonlinear and Adaptive Control Design*. Wiley, 1995.

[KKM91] I. Kanellakopoulos, P. V. Kokotovic, and A. S. Morse. Systematic design of adaptive controllers for feedback linearizable systems. *IEEE Transactions on Automatic Control*, 36(11):1241–1253, 1991.

[KV86] P. R. Kumar and P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice Hall, Inc., 1986.

[LAMK17] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2872–2879, 2017.

[Lév10] Jean Lévine. On necessary and sufficient conditions for differential flatness. *Applicable Algebra in Engineering, Communication and Computing*, 22(1):47–90, 2010.

[Lib10] D. Liberzon. Calculus of variations and optimal control theory: A concise introduction. Online notes, 2010. Retrieved, 16 Jan 2022.

[LM67] E. B. Lee and L. Markus. *Foundations of Optimal Control Theory.* Robert E. Krieger Publishing Company, 1967.

[LP15] A. Lindquist and G. Picci. *Linear Stochastic Systems: A Geometric Approach to Modeling, Estimation and Identification.* Springer, Berlin, Heidelberg, 2015.

[LS95] F. L. Lewis and V. L. Syrmos. *Optimal Control.* Wiley, second edition, 1995.

[Lue97] D. G. Luenberger. *Optimization by Vector Space Methods.* Wiley, New York, 1997.

[LVS12] F. L. Lewis, D. L. Vrabie, and V. L. Syrmos. *Optimal Control.* John Wiley & Sons, Ltd, 2012.

[MA73] P. J. Moylan and B. D. O. Anderson. Nonlinear regulator theory and an inverse optimal control problem. *IEEE Trans. on Automatic Control*, 18(5):460–454, 1973.

[MDP94] P. Martin, S. Devasia, and B. Paden. A different look at output tracking—Control of a VTOL aircraft. *Automatica*, 32(1):101–107, 1994.

[MFHM05] M. B. Milam, R. Franz, J. E. Hauser, and R. M. Murray. Receding horizon control of a vectored thrust flight experiment. *IEE Proceedings on Control Theory and Applications*, 152(3):340–348, 2005.

[MHJ$^+$03] R. M. Murray, J. Hauser, A. Jadbabaie, M. B. Milam, N. Petit, W. B. Dunbar, and R. Franz. Online control customization via optimization-based control. In T. Samad and G. Balas, editors, *Software-Enabled Control: Information Technology for Dynamical Systems.* IEEE Press, 2003.

[Mil03] M. B. Milam. *Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems.* PhD thesis, California Institute of Technology, 2003.

[MM99] M. B. Milam and R. M. Murray. A testbed for nonlinear flight control techniques: The Caltech ducted fan. In *Proc. IEEE International Conference on Control and Applications*, 1999.

[MM02] M. Milam and R. M. Murray et al. NTG: Nonlinear Trajectory Generation library. `http://github.com/murrayrm/ntg`, 2002. Retrieved, 28 Jan 2023.

[MRRS00] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[Mur96] R. M. Murray. Trajectory generation for a towed cable flight control system. In *Proc. IFAC World Congress*, 1996.

[Mur97] R. M. Murray. Nonlinear control of mechanical systems: A Lagrangian perspective. *Annual Reviews in Control*, 21:31–45, 1997.

[PBGM62] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes.* Wiley-Interscience, 1962. (translated from Russian).

[PND99] J. A. Primbs, V. Nevistić, and J. C. Doyle. Nonlinear optimal control: A control Lyapunov function and receding horizon perspective. *Asian Journal of Control*, 1(1):1–11, 1999.

[QB97] S. J. Qin and T. A. Badgwell. An overview of industrial model predictive control technology. In J.C. Kantor, C.E. Garcia, and B. Carnahan, editors, *Fifth International Conference on Chemical Process Control*, pages 232–256, 1997.

[RM98] M. Rathinam and R. Murray. Configuration flatness of Lagrangian systems underactuated by one control. *SIAM Journal of Control and Optimization*, 36(1):164–179, 1998.

[RMD17] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[Rug90] W. J. Rugh. Analytical framework for gain scheduling. In *Proc. American Control Conference*, pages 1688–1694, 1990.

[SC92] R. Shishko and R. G. Chamberlain. NASA systems engineering handbook. Technical report, National Aeronautics and Space Administration, 1992.

[Sha90] J. S. Shamma. Analysis of gain scheduled control for nonlinear plants. *IEEE Transactions on Automatic Control*, 35(12):898–907, 1990.

[SJK97] R. Sepulchre, M. Jankovic, and P. V. Kokotović. *Constructive Nonlinear Control*. Springer, London, 1997.

[Son83] E. D. Sontag. A Lyapunov-like characterization of asymptotic controllability. *SIAM Journal of Control and Optimization*, 21:462–471, 1983.

[vNM98] M. J. van Nieuwstadt and R. M. Murray. Rapid hover to forward flight transitions for a thrust vectored aircraft. *Journal of Guidance, Control, and Dynamics*, 21(1):93–100, 1998.

[vNRM98] M. van Nieuwstadt, M. Rathinam, and R. M. Murray. Differential flatness and absolute equivalence. *SIAM Journal of Control and Optimization*, 36(4):1225–1239, 1998.

Index

actuator
    saturation, 1-3
algebraic Riccati equation, 3-13

bang-bang control, 3-10
binomial distribution, 5-3

control Lyapunov function, 4-5
cost function, 3-1
costate variables, 3-6

decision-making layer, 1-9
defect, for a differentially flat system, 2-15
design V, 1-2
differential flatness, 2-8
    defect, 2-15
dynamic programming, 3-18

error system, 2-3
events, 5-1
expectation, 5-7
exponential distribution, 5-4
extended Kalman filter, 6-5
extremum, 3-4

feasible trajectory, 3-5
feedback regulation layer, 1-8
feedforward, 2-4
final cost, 3-5
finite horizon, 3-5
flat flag, 2-8

gain scheduling, 2-4
Gaussian distribution, 5-4

Hamilton-Jacobi-Bellman equation, 3-18
Hamiltonian, 3-6
hardware-in-the-loop simulation (HIL), 1-3
Harrier AV-8B aircraft, 3-16

infinite horizon, 3-5
information filter, 7-7
information matrix, 7-7
innovations process, 6-4
integral cost, 3-5

Kalman filter
    recursive form, 6-2

Lagrange multipliers, 3-3
linear quadratic, 3-5
linear quadratic Gaussian control problem, 6-6
linearization, 2-3
locally positive definite, 4-5

matrix differential equation, 3-12
mean, 5-4, 5-7
mechanical systems, 2-15
motion primitive, 2-18
moving horizon estimator, 7-13

noise intensity, 5-14
normal distribution, 5-4

operating envelope, 1-8
optimal control problem, 3-4
optimal value, 3-1
optimization, 3-1
Ornstein-Uhlenbeck process, 5-13

Poisson distribution, 5-3
positive definite function, 4-5
probability distribution, 5-1
probability mass function, 5-2
probability measure, 5-2

random process, 5-8
random variable, 5-2
receding horizon control, 2-3, 4-2
regression analysis, 1-3
residual random process, 6-3
Riccati ODE, 3-12

sample space, 5-1
smoothness, 1-3

stabilizable, 4-6
standard deviation, 5-4

terminal cost, 3-5
trajectory generation layer, 1-8
two point boundary value problem, 3-12

uniform distribution, 5-4