10 *Stay in Control*
Feedback Control

So far, we have talked about robot bodies, including their sensors and effectors. Now it's time to turn to robot brains, the controllers that make decisions and command the robot's actions.

From Chapter 2 you remember that control theory was one of the founding areas of robotics, and that feedback control is a key component of every real robot. In this chapter we will learn the principles of feedback control and a bit of math that is used to implement feedback controllers on any system, from a steam engine to a modern robot.

10.1 Feedback or Closed Loop Control

FEEDBACK CONTROL
SET POINT

Feedback control is a means of getting a system (a robot) to achieve and maintain a desired state, usually called the *set point*, by continuously comparing its current state with its desired state.

FEEDBACK

Feedback refers to the information that is sent back, literally "fed back," into the system's controller.

The most popular example of a control system is the thermostat. It is a challenge to describe concepts in control theory without resorting to thermostats, but let's try, since we already know that thermostats are not robots.

DESIRED STATE
GOAL STATE

The *desired state* of the system, also called the *goal state*, is where the system wants to be. Not surprisingly, the notion of goal state is quite fundamental to goal-driven systems, and so it is used both in control theory and in AI, two very different fields, as we saw in Chapter 2. In AI, goals are divided into two types: achievement and maintenance.

ACHIEVEMENT GOAL

Achievement goals are states the system tries to reach, such as a particular location, perhaps the end of a maze. Once the system is there, it is done,

and need not do any more work. AI has traditionally (but not exclusively) concerned itself with achievement goals.

MAINTENANCE GOAL

Maintenance goals, on the other hand, require ongoing active work on the part of the system. Keeping a biped robot balanced and walking, for example, is a maintenance goal. If the robot stops, it falls over and is no longer maintaining its goal of walking while balanced. Similarly, following a wall is a maintenance goal. If the robot stops, it is no longer maintaining its goal of following a wall. Control theory has traditionally (but not exclusively) concerned itself with maintenance goals.

The goal state of a system may be related to internal or external state, or a combination of both. For example, the robot's internal goal may be to keep its battery power level in some desired range of values, and to recharge the battery whenever it starts to get too low. In contrast, the robot's external goal state may be to get to a particular destination, such as the kitchen. Some goal states are combinations of both, such as the goal state that requires the robot to keep its arm extended and balance a pole. The arm state is internal (although externally observable), and the state of the pole is external. The goal state can be arbitrarily complex and consist of a variety of requirements and constraints. Anything the robot is capable of achieving and maintaining is fair game as a goal. Even what is not doable for a robot can be used as a goal, albeit an unreachable one. The robot may just keep trying and never get there. It's good to strive.

So, if the system's current and desired states are the same, it does not need to do anything. But if they are not, which is the case most of the time, how does the system decide what to do? That is what the design of the controller is all about.

10.2 The Many Faces of Error

The difference between the current and desired states of a system is called the *error*, and a goal of any control system is to minimize that error. Feedback control explicitly computes and tells the system the error in order to help it reach the goal. When the error is zero (or small enough), the goal state is reached. Figure 10.1 shows a typical schematic of a feedback control system.

ERROR

As a real world example of error and feedback, let's consider the game that's sometimes called "hot and cold," in which you have to find or guess some hidden object, and your friends help you by saying things like "You're getting warmer, hotter, colder, freezing" and so on. (Ok, we are talking about

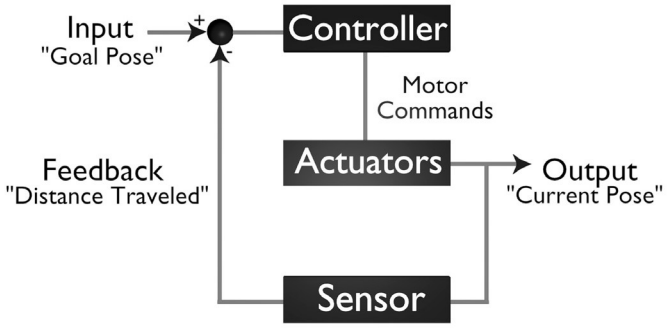

Figure 10.1 A diagram of a typical feedback controller.

temperature now, but we are still not talking about thermostats.) What they are really doing is computing the error and giving you feedback about it.

Imagine a cruel version of the same game, in which your friends tell you only "You are there, you win!" or "Nope, you are not there." In that case, what they are telling you is only if the error is zero or non-zero, if you are at the goal state or not. This is not very much information, since it does not help you figure out which way to go in order to get closer to the goal, to minimize the error.

Knowing if the error is zero or non-zero is better than knowing nothing, but not by much. In the normal version of the game, when you are told "warm" or "cool," you are being given the *direction of the error*, which allows for minimizing the error and getting closer to the goal.

DIRECTION OF ERROR

By the way, in Chapter 21 we will learn how the notion of feedback is related to the basic principles of reinforcement learning. Replace "warm" with "good robot" and "cool" with "bad robot," and you will start to get the idea of how some forms of learning work. But we have a few chapters to go before we get there.

MAGNITUDE OF ERROR

When the system knows how far off it is from the goal, it knows the *mag-*

nitude of error, the distance to the goal state. In the "hot and cold" game, the gradations of freezing, chilled, cool, warm, and so on are used to indicate the distance from (or closeness to) the goal object.

We can do better still by telling more than just the magnitude of the error. If the system is given the precise direction and magnitude of the error, it is told exactly what to do to get to the goal. In the game above, if you told the person who was guessing: "Ok, turn toward the door, go through to the kitchen, open the freezer, and look under the ice tray," it would not be much of a game. For robots, on the other hand, getting that much information still does not make control or learning trivial, but it does make it easier.

As you can see, control is made easier if the robot is given plenty of feedback information about its error and if that information is provided accurately and frequently. Let's work this out through an example of a wall-following robot.

10.3 An Example of a Feedback Control Robot

How would you write a controller for a wall-following robot using feedback control?

The first step is to consider the goal of the task. In wall-following, the goal state is a particular distance, or range of distances, from a wall. This is a maintenance goal, since wall-following involves keeping that distance over time.

Given the goal, it is simple to work out the error. In the case of wall-following, the error is the difference between the desired distance from the wall and the actual distance at any point in time. Whenever the robot is at the desired distance (or range of distances), it is in the goal state. Otherwise, it is not.

We are now ready to write the controller, but before we do that, we should consider sensors, since they will have to provide the information from which both state and error will be computed.

What sensor(s) would you use for a wall-following robot and what information would they provide?

Would they provide magnitude and direction of the error, or just magnitude, or neither? For example, a contact bump sensor would provide the least information (it's a simple sensor, after all). Such a sensor would tell

the robot only whether it has hit a wall; the robot could sense the wall only through contact, not at a distance. An infra red sensor would provide information about a possible wall, but not the exact distance to it. A sonar would provide distance, as would a laser. A stereo vision system could also provide distance and even allow for reconstructing more about the wall, but that would most definitely be overkill for the task. As you can see, the sensors determine what type of feedback can be available to the robot.

Whatever sensor is used, assume that it provides the information to compute *distance-to-wall*. The robot's goal is to keep distance-to-wall at a particular value or, more realistically, in a particular range of values (let's say between 2 and 3 feet). Now we can write the robot's feedback controller in the form of standard if-then-else conditional statements used in programming, like this one:

```
If distance-to-wall is in the right range,
     then keep going.
If distance-to-wall is larger than desired,
     then turn toward the wall,
     else turn away from the wall.
```

Given the above controller algorithm, what will the robot's behavior look like? It will keep moving and switch/wiggle back and forth as it moves along. How much switching back and forth will it do? That depends on two parameters: how often the error is computed and how much of a correction (turn) is made each time.

Consider the following controller:

```
If distance-to-wall is exactly as desired,
     then keep going.
If distance-to-wall is larger than desired,
     then turn by 45 degrees toward the wall,
     else turn by 45 degrees away from the wall.
```

The above controller is not very smart. Why? To visualize it, draw a wall and a robot and follow the rules of the above controller through a few iterations to see what the robot's path looks like. It oscillates a great deal and rarely if ever reaches the desired distance before getting too close to or too far from the wall.

In general, the behavior of any simple feedback system oscillates around

the desired state. (Yes, even in the case of thermostats.) Therefore, the robot oscillates around the desired distance from the wall; most of the time it is either too close or too far away.

How can we decrease this oscillation?

There are a few things we can do. The first is to compute the error often, so the robot can turn often rather than rarely. Another is to adjust the turning angle so the robot turns by small rather than large angles. Still another is to find just the right range of distances that defines the robot's goal. Deciding how often to compute the error, how large a turning angle to use, and how to define the range of distances all depend on the specific parameters of the robot system: the robot's speed of movement, the range of the sensor(s), and the rate with which new distance-to-wall is sensed and computed, called the *sampling rate*.

SAMPLING RATE

The calibration of the control parameters is a necessary, very important, and time-consuming part of designing robot controllers.

Control theory provides a formal framework for how to properly use parameters in order to make controllers effective. Let's learn how.

10.4 Types of Feedback Control

The three most used types feedback control are proportional control (P), proportional derivative control (PD), and proportional integral derivative control (PID). These are commonly referred to as P, PD, and PID control. Let's learn about each of them and use our wall-following robot as an example system.

10.4.1 Proportional Control

PROPORTIONAL CONTROL

The basic idea of *proportional control* is to have the system respond in proportion to the error, using both the direction and the magnitude of the error. A proportional controller produces an output o proportional to its input i, and is formally written as:

$$o = K_p i$$

K_p is a proportionality constant, usually a constant that makes things work, and is specific to a particular control system. Such parameters are abundant

in control; typically you have to figure them out by trial and error and calibration.

What would a proportional controller for our wall fallowing robot look like?

It would use distance-to-wall as a parameter to determine the angle and distance and/or speed with which the robot would turn. The larger the error, the larger the turning angle and speed and/or distance; the smaller the error, the smaller the turning angle and speed and/or distance.

GAIN

In control theory, the parameters that determine the magnitude of the system's response are called *gains*. Determining the right gains is typically very difficult, and requires trial and error, testing and calibrating the system repeatedly. In some cases, if the system is very well understood, gains can be computed mathematically, but that is rare.

PROPORTIONAL GAIN

If the value of a particular gain is proportional to that of the error, it is called *proportional gain*. As we saw in the case of our wall-following robot, incorrect gain values cause the system to undershoot or overshoot the desired state. The gain values determine whether the robot will keep oscillating or will eventually settle on the desired state.

DAMPING

Damping refers to the process of systematically decreasing oscillations. A system is properly *damped* if it does not oscillate out of control, meaning its oscillations are either completely avoided (which is very rare) or, more practically, the oscillations gradually decrease toward the desired state within a reasonable time period. Gains have to be adjusted in order to make a system properly damped. This is a tuning process that is specific to the particular control system (robot or otherwise).

When adjusting gains, you have to keep in mind both the physical and the computational properties of the system. For example, how the motor responds to speed commands plays a key part in control, as do the backlash and friction in the gears (remember Chapter 4?), and so on. Physical properties of the robot influence the exact values of the gains, because they constrain what the system actually does in response to a command. *Actuator uncertainty* makes it impossible for a robot (or a human, for that matter) to know the exact outcome of an action ahead of time, even for a simple action such as "Go forward three feet." While actuator uncertainty keeps us from predicting the exact outcome of actions, we can use probability to estimate and make a pretty good guess, assuming we know enough about the system to set up the probabilities correctly. See the end of the chapter for more reading on probabilistic robotics.

ACTUATOR UNCERTAINTY

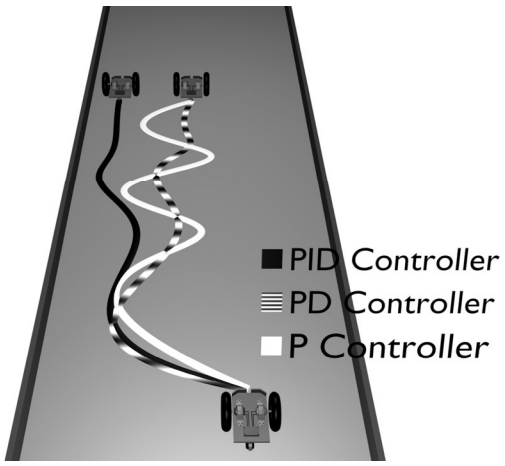

Figure 10.2 Different trajectories produced by P, PD, and PID feedback controllers.

10.4.2 Derivative Control

As we have seen, setting gains is difficult, and simply increasing the proportional gain does not remove oscillatory problems from a control system. While this may work for small gains (called low gains; gain is referred to as high or low), as the gain increases, the system's oscillations increase along with it. The basic problem has to do with the distance from the set point/desired state: *When the system is close to the desired state, it needs to be controlled differently than when it is far from it.* Otherwise, the momentum generated by the controller's response to the error, its own correction, carries the system beyond the desired state and causes oscillations. One solution to this problem is to correct the momentum as the system approaches the desired state.

First, let's remember what momentum is:

$$Momentum = mass * velocity$$

Since momentum and velocity are directly proportional (the faster you move and/or the bigger you are, the more momentum you have), we can control momentum by controlling the velocity of the system. As the system nears the desired state, we subtract an amount proportional to the velocity:

$$-(gain * velocity)$$

DERIVATIVE TERM

This is called the *derivative term*, because velocity is the derivative (the rate of change) of position. Thus, a controller that has a derivative term is called a D controller.

A derivative controller produces an output o proportional to the derivative of its input i:

$$o = K_d \frac{di}{dt}$$

K_d is a proportionality constant, as before, but this time with a different name, so don't assume you can use the same number in both equations.

The intuition behind derivative control is that the controller corrects for the momentum of the system as it approaches the desired state. Let's apply this idea to our wall-following robot. A derivative controller would slow the robot down and decrease the angle of its turning as its distance from the wall gets closer to the desired state, the optimal distance to the wall.

10.4.3 Integral Control

INTEGRAL TERM

STEADY STATE ERROR

Yet another level of improvement can be made to a control system by introducing the so-called *integral term*, or I. The idea is that the system keeps track of its own errors, in particular of the repeatable, fixed errors that are called *steady state errors*. The system integrates (sums up) these incremental errors over time, and once they reach some predetermined threshold (once the cumulative error gets large enough), the system does something to compensate/correct.

An integral controller produces an output o proportional to the integral of its input i:

$$o = K_f \int i(t)dt$$

K_f is a proportionality constant. You get the pattern.

How do we apply integral control to our wall-following robot?

Actually, it's not too easy, because there is no obvious place for steady state error to build up in the simple controller we came up with. That's a good thing about our controller, so we can pat ourselves on the back and take another example. Consider a lawn-mowing robot which carefully covers the complete lawn by going from one side of the yard to the other, and moving

over by a little bit every time to cover the next strip of grass in the yard. Now suppose that the robot has some consistent error in its turning mechanism, so whenever it tries to turn by a 90-degree angle to move over to the next strip on the grass, it actually turns by a smaller angle. This makes it fail to cover the yard completely, and the longer the robot runs, the worse its coverage of the yard gets. But if it has a way of measuring its error, even only once it gets large enough (for example, by being able to detect that it is mowing already mowed grass areas a lot), it can apply integral control to recalibrate.

Now you know about P, I, and D, the basic types of feedback control. Most real-world systems actually use combinations of those three basic types; PD and PID controllers are especially prevalent, and are most commonly used in industrial applications. Let's see what they are.

10.4.4 PD and PID Control

PD control is a combination, actually simply a sum, of proportional (P) and derivative (D) control terms:

$$o = K_p i + K_d \frac{di}{dt}$$

PD control is extremely useful and applied in most industrial plants for process control.

PID control is a combination (yes, again a sum) of proportional P, integral I, and derivative D control terms:

$$o = K_p i + K_f \int i(t)dt + K_d \frac{di}{dt}$$

Figure 10.2 shows example trajectories that would be produced by our wall-following robot if it were controlled by P, PD, and PID feedback controllers.

You can learn a great deal more about feedback control and the larger topic of control theory from the sources given at the end of this chapter. It's a large field studied in electrical engineering and mechanical engineering.

Before we move on, we should put control theory into perspective relative to robotics. As you have been reading and will read much more in the chapter to come, getting robots to do something useful requires many components. Feedback control plays a role at the low level, for example, for controlling the wheels or other continuously moving actuators. But for other aspects of robot control, in particular achieving higher-level goals (navigation, coordination, interaction, collaboration, human-robot interaction),

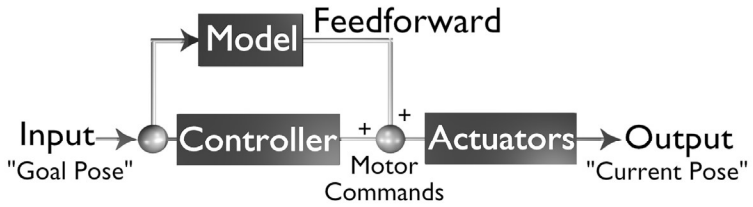

Figure 10.3 A diagram of a typical feedforward controller.

other approaches must come into play that are better suited for representing and handling those challenges. Those levels of robot control use techniques that come from the field of artificial intelligence, but that have come a long way from what AI used to be about in its early days, as we learned in Chapter 2. That's coming up, so stick around.

10.5 Feedforward or Open Loop Control

CLOSED LOOP CONTROL

Feedback control is also called *closed loop control* because it closes the loop between the input and the output, and provides the system with the error as feedback.

What is an alternative to closed loop control?

OPEN LOOP CONTROL
FEEDFORWARD CONTROL

You guessed it (or you read the section title): the alternative to feedback or closed loop control is called feedforward or open loop control. As the name implies, *open loop control* or *feedforward control*, does not use sensory feedback, and state is not fed back into the system. Thus the loop between the input and output is open, and not really a loop at all. In open loop control, the system executes the command that is given, based on what was predicted, instead of looking at the state of the system and updating it as it goes along, as in feedback control. In order to decide how to act in advance, the controller determines set points or sub-goals for itself ahead of time. This requires looking ahead (or looking forward) and predicting the state of the system, which is why the approach is called feedforward. Figure 10.3 shows a schematic of a feedforward open loop controller.

Open loop or feedforward control systems can operate effectively if they are well calibrated and their environment is predictable and does not change

in a way that affects their performance. Therefore they are well suited for repetitive, state-independent tasks. As you have probably guessed, those are not very common in robotics.

To Summarize

- Feedback/closed loop control and feedforward/open loop control are important aspects of robotics.
- Feedback control aims to minimize system error, the difference between the current state and the desired state.
- The desired state or goal state is a concept used in AI as well as control theory, and can come in the form of achievement or maintenance.
- Error is a complex concept with direction and magnitude, and cumulative properties.
- Proportional, derivative, and integral control are the basic forms of feedback control, which are usually used in combination in real-world control systems (robotic and otherwise).

Food for Thought

- What happens when you have sensor error in your system? What if your sensor incorrectly tells you that the robot is far from a wall but in fact it is not? What about vice versa? How might you address these issues?
- What can you do with open loop control in your robot? When might it be useful?

Looking for More?

- The Robotics Primer Workbook exercises for this chapter are found here: http://roboticsprimer.sourceforge.net/workbook/Feedback_Control
- Here are some textbooks on control theory, if you want to learn (much) more:
  - *Signals and Systems* by Simon Haykin and Barry Van Veen, John Wiley.

- *Intelligent Control Systems, Theory and Applications* edited by Madan M. Gupta and Naresh K. Sinha.
- *Linear Control System Analysis and Design: Conventional and Modern* by J. J. D'Azzo and C. Houpis.
- *Automatic Control Systems* by B. C. Kuo.