Chapter 6
Control

Robotics algorithms make decisions. The robot is given a task, but in order to perform the task it must take actions and these actions depend on the environment as detected by the sensors. For example, if a robot is to bring an object from a shelf in a warehouse to a delivery track, it must use sensors to navigate to the proper shelf, to detect and grasp the object, and then to navigate back to the truck and load the object. Only robots that act in extremely well-defined environments can carry out such tasks without information from sensors. An example is a robotic arm assembling a device in a factory; if the parts are precisely positioned on the work surface, the robot can manipulate the parts without sensing them. But in most environments sensors must be used. In a warehouse there may be obstacles on the way to the shelf, the object will not be precisely positioned on the shelf and the truck is never parked in exactly the same place. The robot needs to adapt to these small variations using *control algorithms* to make decisions: based upon data from the sensors what actions does the robot need to perform in order to accomplish the task? There is a sophisticated mathematical theory of control that is fundamental in robotics. In this chapter, we present the basic concepts of control algorithms.

Section 6.1 explains the difference between two control models: open loop control where the parameters of the algorithm are set in advance and closed loop control where data from sensors influences the behavior of the algorithm. Sections 6.2–6.5 present four increasingly sophisticated closed loop control algorithms. The designer of a robot must choose among these and similar algorithms to select the one that gives adequate performance for the least computational cost.

M. Ben-Ari and F. Mondada, *Elements of Robotics*,
https://doi.org/10.1007/978-3-319-62533-1_6

6.1 Control Models

There are two ways that a control algorithm can decide upon an action. In an open loop system, the parameters of the control algorithm are preset and do not change while the system runs. In a closed loop system, sensors measure the error between the desired state of the system and its actual state, and this error is used to decide what action to take.

6.1.1 Open Loop Control

A toaster is a machine that performs actions semi-autonomously. You place slices of bread in the toaster, set the timer and push the lever down to start the toasting action. As we all know, the results are not guaranteed: if the duration of the timer is too short, we have to toast the bread again; if the duration of the timer is too long, the scent of burnt toast floats through the kitchen. The outcome is uncertain because a toaster is an *open loop control system*. It does not check the outcome of the toasting action to see if the required result has been achieved. Open loop systems are very familiar: on a washing machine you can set the temperature of the water, the duration of the cycle and the amount of detergent used, but the machine does not measure the "cleanness" of the clothes (whatever that means) and modify its actions accordingly.

A mobile robot that moves to a target position based on odometry alone (Sect. 5.4) is also using open loop control. By keeping track of the motor power and the duration that the motors run, the robot can compute the distance it has moved. However, variations in the speed of the wheels and in the surface on which the robot moves will cause uncertainty in the final position of the robot. In most applications, odometry can be used to move the robot to the vicinity of the goal position, at which point sensors are used to move the robot to the precise goal position, for example, by using sensors to measure the distance to an object.

6.1.2 Closed Loop Control

To achieve autonomous behavior, robots use *closed loop control systems*. We have already encountered closed loop systems in the Braintenberg vehicles (Activity 3.5):

> **Specification (Attractive and repulsive)**: When an object approaches the robot from behind, it runs away until it is out of range.

The robot must *measure* the distance to the object and stop when this distance is sufficiently large. The power setting of the motors depends on the measurement of the distance, but the robot moves at a speed that depends on the power of the motors,

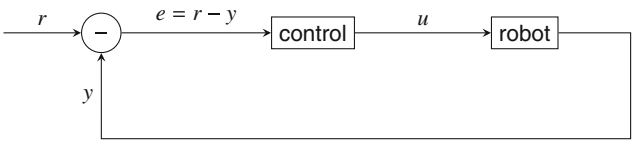

Fig. 6.1 A closed loop control system

which changes the distance to the object, which again modifies the power setting, which …. This circular behavior is the origin of the term "closed loop."

We now formalize the specification of a closed loop control system for a robot (Fig. 6.1). The variable r represents the *reference value*, the specification of the robot's task. In a warehouse robot, reference values include the position of the robot relative to a stack of shelves and the distance of the gripper arm from the object to be picked up. A reference value cannot be used directly by the robot; instead, it must be transformed into a *control value* u. For example, if the reference value is the position of the robot relative to a shelf, the control value will be power settings of the motors and the duration that the motors are running. The variable y represents the output, that is, the actual state of the robot, for example, the distance to an object.

The model in Fig. 6.1 is also called a *feedback control system* because the output value y is fed back to the control algorithm and used to compute the control value. The output is compared with the reference value to compute $e = r - y$, the *error*. The control algorithm uses the error to generate the control signal u that is the input to the robot.

6.1.3 The Period of a Control Algorithm

Control algorithms are run periodically (Algorithm 6.1). The robot's software initializes a *timer variable* to the required duration of the period to run the algorithm, for example, every 20 ms. The embedded computer has a *hardware clock* that "ticks" at fixed intervals causing an interrupt. The interrupt is handled by the operating system which decrements the value of the timer variable. When the value of this variable goes to zero, the timer has expired, and an event is raised in the software causing the control algorithm to be run.

The period of the algorithm is an important parameter in the design of a control system. If the period is too short, valuable computing resources will be wasted and the computer can become overloaded to the point that commands to the robot arrive too late. If the period is too long, the robot will not respond in time to correct errors in its motion.

Algorithm 6.1: Control algorithm outline

```
        integer period                    // Duration of timer period
        integer timer                     // Timer variable
 1: period ← ···                          // Period in milliseconds
 2: timer ← period                        // Initialize the timer
 3: loop
 4:     when timer-expired-event occurs
 5:         control algorithm             // Run the algorithm
 6:         timer ← period                // Reset the timer

    // Operating system
 7: when hardware-clock-interrupt occurs
 8:     timer ← timer − 1                 // Decrement the timer
 9:     if timer = 0                      // If the timer expires
10:         raise timer-expired-event     //     raise an event
```

Example Consider a robot approaching an object that is 10 cm away at 2 cm/s. A control period of 1 ms would waste computing resources because the robot will move only 0.002 cm (0.02 mm) during *each* 1 ms cycle of the control algorithm. Changes in motor power over such small distances won't affect the ability of the robot to fulfill its task. At the opposite extreme, a control period of 2 s is even worse: the robot will move 4 cm during this period and will likely to crash into the object. A control period of roughly 0.25 s (250 ms) during which the robot moves 0.5 cm seems a reasonable value to start with since 0.5 cm is a distance that is meaningful in terms of approaching an object. You can experiment with periods around this value to determine the optimum period: one that achieves satisfactory behavior with a period that is as long as possible to reduce the cost of computation.

Activity 6.1: Setting the control period

- In the example, we came to the conclusion that the optimum period of the control algorithm was of the order of magnitude of tenths of a second. In this activity we ask you consider what the optimum period should be for other control algorithms.
- A home heating system contains a thermostat for controlling the temperature. What would be an optimum period for the control algorithm? The period depends on the engineering parameters of the heating system and on the physical properties of how the heat is transferred to the rooms. Explain how you would measure these factors and how they affect the control period.
- Consider a self-driving car trying to park. What assumptions do you need to make to design a control period? What would be a reasonable period?

- How do the properties of the sensors affect the control period? For the example of a robot approaching an object, how would the period change if the sensor can detect the object at 2 cm, 5 cm, 10 cm, 20 cm, 40 cm?

We now define a sequence of four control algorithms, each one building on the previous one and providing more accurate control, at the price of more computational complexity. In practice, the system designer should choose the simplest algorithm that enables the robot to fulfill its task.

The algorithms are presented in the context of a robot which must approach an object and stop at a distance s in front of it. The distance is measured by a proximity sensor and the speed of the robot is controlled by setting the power of the motors.

6.2 On-Off Control

The first control algorithm is called the *on-off* or *bang-bang* algorithm (Algorithm 6.2). We define a constant reference that is the distance in front of the object at which the robot is to stop. The variable measured is the actual distance measured by the proximity sensor. The error is the difference between the two:

$$\text{error} \leftarrow \text{reference} - \text{measured},$$

Algorithm 6.2: On-off controller

```
integer reference ← ···     // Reference distance
integer measured            // Measured distance
integer error               // Distance error
error ← reference − measured
if error < 0
    left-motor-power ← 100      // Move forwards
    right-motor-power ← 100
if error = 0
    left-motor-power ← 0        // Turn off motors
    right-motor-power ← 0
if error > 0
    left-motor-power ← −100     // Move backwards
    right-motor-power ← −100
```

which is negative if the robot is too far away from the object and positive if it is too close to the object. The motor powers are turned to full forwards or full backwards depending on the sign of the error. For example, if the reference distance is 10 cm and the measured distance is 20 cm, the robot is too far away and the error is −10 cm. Therefore, the motors must be set to move forwards.

The robot approaches the object at full speed. When the robot reaches the reference distance from the object, it takes time for the sensor to be read and the error to be computed. Even if the robot measures a distance exactly equal to the reference distance (which is unlikely), the robot will not be able to stop immediately and will overrun the reference distance. The algorithm will then cause the robot to back up at full speed, again passing the reference distance. When the timer causes the control algorithm to be run again, the robot will reverse direction and go forwards at full speed. The resulting behavior of the robot is shown in Fig. 6.2: the robot will oscillate around the reference distance to the object. It is highly unlikely that the robot will actually stop at or near the reference distance.

A further disadvantage of the on-off algorithm is that the frequent and abrupt reversal of direction results in high accelerations. If we are trying to control a gripper arm, the objects that it is carrying may be damaged. The algorithm generates high levels of wear and tear on the motors and on other mechanical moving parts.

Activity 6.2: On-off controller

- Implement the on-off algorithm on your robot for the task of stopping at a reference distance from an object.
- Run it several times starting at differences distances from the object.

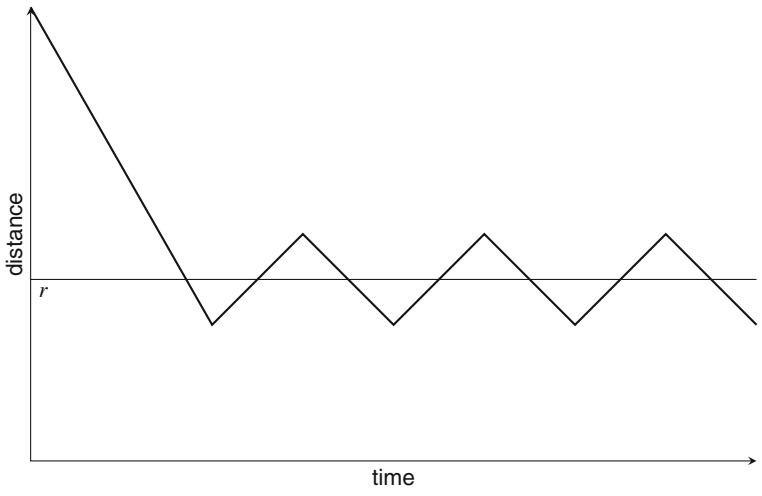

Fig. 6.2 Behavior of the on-off algorithm

- Algorithm 6.2 stops the robot when the error is exactly zero. Modify the implementation so that the robot stops if the error is within a small range around zero. Experiment with different ranges and see how they affect the behavior of the robot.

6.3 Proportional (P) Controller

To develop a better algorithm, we take inspiration from riding a bicycle. Suppose that you are riding your bicycle and see that the traffic light ahead has turned red. You don't wait until the last moment when you are at the stop line and then squeeze hard on the brake lever; if you do so, you might be thrown from the bicycle! What you do is to slow your speed gradually: first, you stop pedaling; then, you squeeze the brake gently to slow down a bit more; finally, when you are at the stop line and going slowly, you squeeze harder to fully stop the bicycle. The algorithm used by a bicycle rider can be expressed as:

> Reduce your speed more as you get closer to the reference distance.

The decrease in speed is (inversely) *proportional* to how close you are to the traffic light: the closer you are, the more you slow down. The factor of proportionality is called the *gain* of the control algorithm. An alternate way of expressing this algorithm is:

> Reduce your speed more as the error between the reference distance and the measured distance gets smaller.

Algorithm 6.3 is the *proportional control algorithm* or a *P-controller*.

Algorithm 6.3: Proportional controller

```
integer reference ← ···   // Reference distance
integer measured          // Measured distance
integer error             // Error
float gain ← ···          // Proportional gain
integer power             // Motor power
1: error ← reference − measured // Distances
2: power ← gain * error         // Control value
3: left-motor-power ← power
4: right-motor-power ← power
```

Example Suppose that the reference distance is 100 cm and the gain is −0.8. When the robot is 150 cm away from the object, the error is $100-150 = -50$ and the control algorithm will set the power to $-0.8 \cdot -50 = 40$. Table 6.1 shows the errors and power settings for three distances. If the robot overruns the reference distance of 100 cm and a distance of 60 cm is measured, the power will be set to −32 causing the robot to move backwards.

Figure 6.3 plots the distance of the robot to the object as a function of time when the robot is controlled by a P controller. The line labeled r is the reference distance. The change in the motor power is smooth so the robot doesn't experience rapid accelerations and decelerations. The response is somewhat slow, but the robot does approach the target distance.

Unfortunately, the robot does not actually reach the reference distance. To understand why this happens, consider what happens when the robot is very close to the reference distance. The error will be very small and consequently the power setting will be very low. In theory, the low power setting should cause the robot to move slowly, eventually reaching the reference distance. In practice, the motor power may become so low that it is not able to overcome the internal friction in the motors and their connection to the wheels, so the robot stops moving.

It might seem that increasing the gain of the P controller could overcome this problem, but a high gain suffers from a serious disadvantage. Figure 6.4 shows the effect of the gain on the P controller. Higher gain (dashed red line) causes the robot to

Table 6.1 Proportional controller for gain of −0.8

Distance	Error	Power
150	−50	40
125	−25	20
60	40	−32

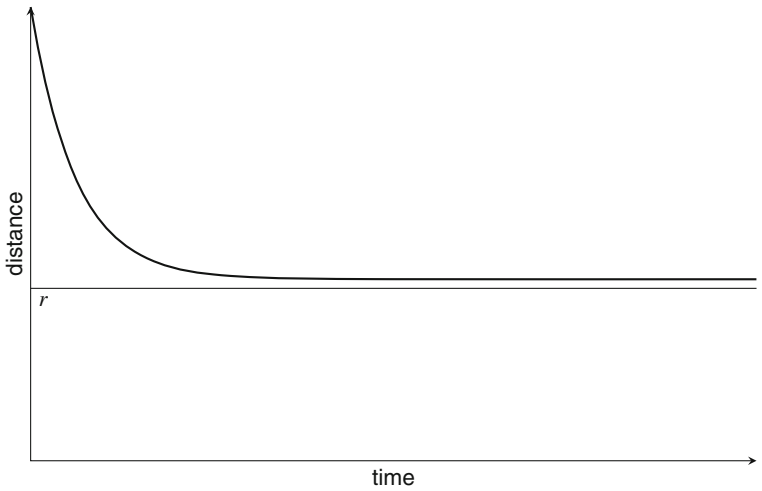

Fig. 6.3 Behavior of the P controller

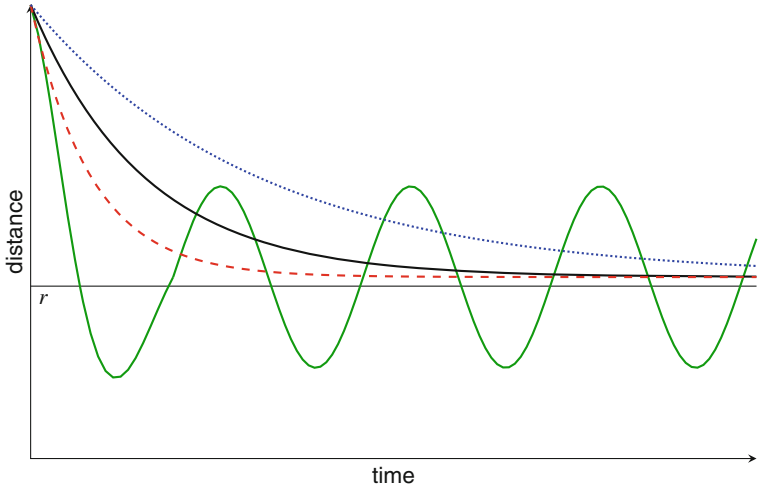

Fig. 6.4 The effect of the gain on the P controller: lower gain (*dotted blue line*), higher gain (*dashed red line*), excessive gain (oscillating *green line*)

approach the reference distance faster, while lower gain (dotted blue line) causes the robot to approach the reference distance slower. However, if the gain is too high, the P controller functions like an on-off controller with an oscillating response (green line). We say that the controller is *unstable*.

There are situations where the P controller cannot reach the reference distance even in a ideal system. Suppose that the object itself is moving at constant speed away from the robot. The P controller will set maximum motor power to cause the robot to move rapidly towards the object. Eventually, however, as the robot approaches the object, the measured distance will become small and the P controller will set the power so low that the speed of the robot is lower than the speed of the object. The result is that the robot will never reach the reference distance. If the robot could actually reach the reference distance, the error would be zero and therefore the speed of the robot would also be zero. The object, however, is still moving away from the robot, so somewhat later the robot will start moving again and the cycle repeats. This start-and-stop motion is not the intended goal of maintaining the reference distance.

Example We use the same data as in the previous example except that the object moves at 20 cm/s. Table 6.2 shows the errors and power settings for three distances. Initially, the robot is going faster then the object so it will catch up. At 125 cm from the object, however, the robot is moving at the same speed as the object. It maintains this fixed distance and will not approach the reference distance of 100 cm. If somehow the robot gets closer to the object, say, 110 cm, the power is reduced to 8 causing the robot to back away from the object.

Table 6.2 Proportional controller for a moving object and a gain of −0.8

Distance	Error	Power
150	−50	40
125	−25	20
110	−10	8

In general, the robot will stabilize at a fixed distance from the reference distance. You can reduce this error by increasing the gain, but the reference distance will never be reached and the only result is that the controller becomes unstable.

Activity 6.3: Proportional controller

- Implement the proportional control algorithm to cause the robot to stop at a specified distance from an object. How accurately can you achieve the goal when the object does not move?
- What happens if the object moves? For the object you can use a second robot programmed to move at a fixed speed.
- Experiment with the gain and the period to see how they affect the performance of the algorithm.

6.4 Proportional-Integral (PI) Controller

A *proportional-integral controller* can achieve the reference distance even in the presence of friction or a moving object by taking into account the accumulated error over time. Whereas the P controller only takes into account the current error:

$$u(t) = k_p e(t)\,,$$

the PI controller adds the integral of the error from the time when the algorithm starts to run until the present time:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\, d\tau\,.$$

Separate gain factors are used for the proportional and integral terms to allow flexibility in the design of the controller.

When implementing a PI controller, a discrete approximation to the continuous integral is performed (Algorithm 6.4).

Algorithm 6.4: Proportional-integral controller

```
integer reference ← ···                    // Reference distance
integer measured                           // Measured distance
integer error                              // Error
integer error-sum ← 0                      // Cumulative error
float gain-p ← ···                         // Proportional gain
float gain-i ← ···                         // Integral gain
integer power                              // Motor power
1: error ← reference − measured            // Distances
2: error-sum ← error-sum + error           // Integral term
3: power ← gain-p * error + gain-i * error-sum // Control value
4: left-motor-power ← power
5: right-motor-power ← power
```

In the presence of friction or a moving object, the error will be integrated and cause a higher motor power to be set; this will cause the robot to converge to the reference distance. A problem with a PI controller is that the integration of the error starts from the initial state when robot is far from the object. As the robot approaches the reference distance, the integral term of the controller will have already a large value; to decrease this value the robot must move past the reference distance so that there are errors of opposite sign. This can generate oscillations (Fig. 6.5).

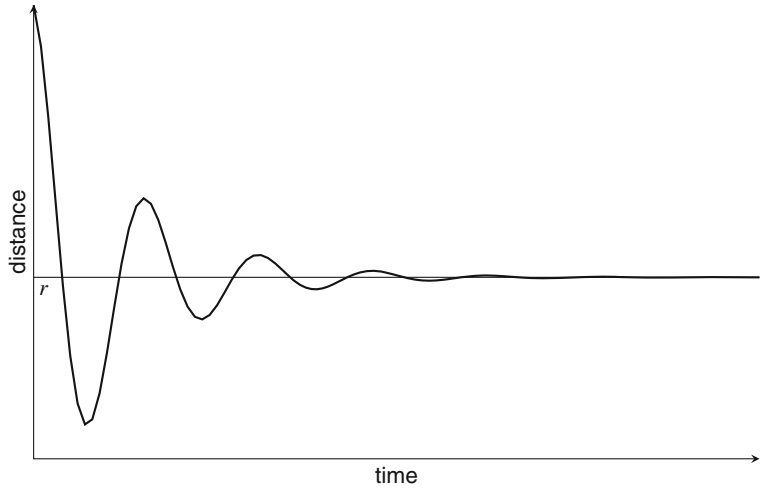

Fig. 6.5 Behavior of the PI controller

Activity 6.4: PI controller

- Implement a PI controller that causes the robot to stop at a specified distance from an object.
- Compare the behavior of the PI controller with a P controller for the same task by monitoring the variables of the control algorithms over time.
- What happens if you manually prevent the robot from moving for a short time and then let it go? This demonstrates a concept called *integrator windup*. Explore the concept through online sources and find a method to fix the problem.

6.5 Proportional-Integral-Derivative (PID) Controller

When you throw or kick a ball to another player who is moving, you do not throw it to his current position. By the time the ball reaches him, he will have moved to a new position. Instead, you estimate where the new position will be and aim the ball there. Similarly, a robot whose task is to push a parcel onto a moving trolley must time its push to the estimated future position of the trolley when the parcel reaches it. The control algorithm of this robot cannot be an on-off, P or PI controller, because they only take into account the current value of the error (and for the PI controller the previous values).

To estimate the future error, the rate of change of the error can be taken into account. If the rate of change of the error is small, the robot can push the parcel just before the trolley approaches it, while if the rate of change of the error is large, the parcel should be pushed much earlier.

Mathematically, rate of change is expressed as a derivative. A *proportional-integral-derivative (PID)* controller adds an additional term to P and I terms:

$$u(t) = k_p e(t) + k_i \int_{\tau=0}^{t} e(\tau)\, d\tau + k_d \frac{de(t)}{dt}\,. \tag{6.1}$$

In the implementation of a PID controller, the differential is approximated by the difference between the previous error and current error (Algorithm 6.5).

Algorithm 6.5: Proportional-integral-differential controller

```
integer reference ← ···                    // Reference distance
integer measured                           // Measured distance
integer error                              // Error
integer error-sum ← 0                      // Cumulative error
integer previous-error ← 0                 // Previous error
integer error-diff                         // Error difference
float gain-p ← ···                         // Proportional gain
float gain-i ← ···                         // Integral gain
float gain-d ← ···                         // Derivative gain
integer power                              // Motor power
```

```
1: error ← reference − measured            // Distances
2: error-sum ← error-sum + error           // Integral term
3: error-diff ← error − previous-error     // Differential term
4: previous-error ← error                  // Save current error
5: power ← gain-p * error +                // Control value
       gain-i * error-sum + gain-d * error-diff
6: left-motor-power ← power
7: right-motor-power ← power
```

The behavior of the PID controller is shown in Fig. 6.6. The robot smoothly and rapidly converges to the reference distance.

The gains of a PID controller must be carefully balanced. If the gains for the P and I terms are too high, oscillations can occur. If the gain for the D term is too high, the controller will react to short bursts of noise.

Activity 6.5: PID controller

- Implement a PID controller for the task of a robot approaching an object.
- Experiment with different gains until the robot smoothly approaches the reference distance.
- Repeat the experiments with a moving object.

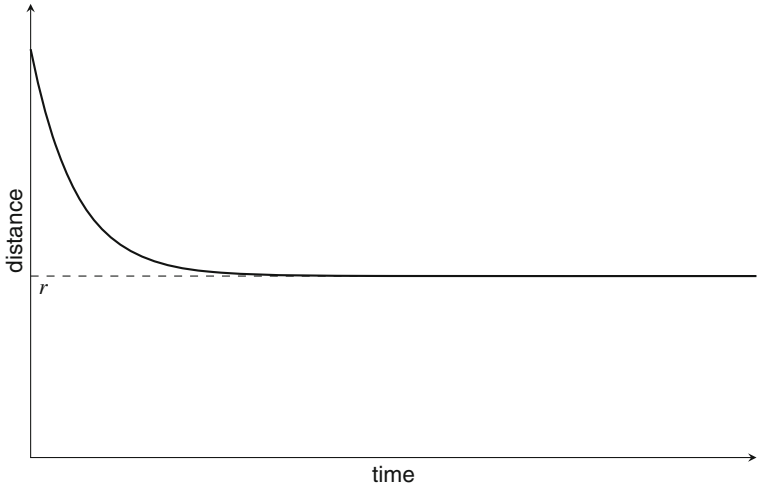

Fig. 6.6 Behavior of the PID controller

6.6 Summary

A good control algorithm should converge rapidly to the desired result while avoiding abrupt motion. It must be computationally efficient, but not require constant tuning. The control algorithm has to be adapted to the specific requirements of the system and the task, and to function correctly in different environmental conditions. We have described four algorithms, from the impractical on-off algorithm through algorithms that combine proportional, integral and derivative terms. The proportional term ensures that large errors cause rapid convergence to the reference, the integral term ensures that the reference can actually be attained, while the derivative term makes the algorithm more responsive.

6.7 Further Reading

A modern textbook on control algorithms is [1].

Reference

1. Åström, K.J., Murray, R.M.: Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press (2008). The draft of a second edition is available online at http://www.cds.caltech.edu/~murray/amwiki/index.php/Second_Edition