3 Programming Robots

Chapter Outline

Instructions for useful actions

Program design
- *Flowcharts*

Programming
- *Abstraction and interface*
- *Commenting*
- *NXT-G programming environment*

Conclusion

Outcomes

—Demonstrate the ability to program a basic robot controller.

Instructions for useful action

Without a set of instructions to follow, a robot wouldn't be very useful. This set of instructions given to the robot is called a program, and the process of creating the program is called programming, which is the topic of this chapter. Don't believe the stereotypical view that all programmers do is sit in front of a screen mindlessly typing on a keyboard. While that is one part of computer programming, you might be surprised to learn how much creativity and careful planning it takes to write a program. Just like designing a robot or some other machine, programming is a creative process that takes invention, ingenuity, planning, and hard work and there is usually more than one way to accomplish a task. Some tasks are complex enough that large teams of programmers work together so flowcharts are used as a standard way to map out the plan for what each part of a program will do, creating a plan for the program in the same way builders use blueprints to know how to construct a building. This allows each programmer to know how their assigned part of the program should behave and interact with the rest of the program.

To give instructions to a robot, they need to be in a form that the robot can understand. The instructions for the Jacquard loom we read about in Chapter 1 are physically stored on cards that can be chained together to program the machine to create complex patterns in the fabric it weaves. Modern robots are normally given their instructions electronically—interpreted by a computer so that the robot knows what actions to take—which is why we'll focus on computer programming here.

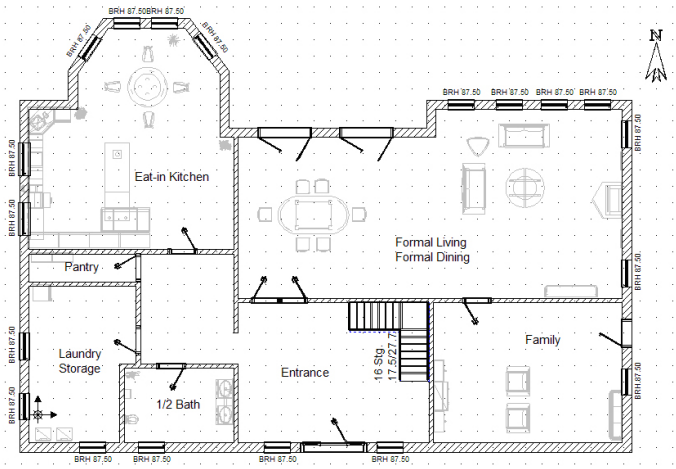

Figure 1. A floor plan gives builders an overall picture of the way a building should turn out.

Program design

If you've ever watched a building being constructed, you've likely seen that it is a complicated process. Without a detailed plan to follow, each person that works on the building might have a slightly different view of what the end goal is, and it would be practically impossible to have the finished product end up looking the way the designer had in mind. In an analogous way, the programs people create to do useful things for us are often so complicated that they wouldn't work as designed unless they are mapped out in what's called a **flowchart**, which is a graphical representation of a program's behavior.

Flowcharts

flowchart
a standardized visual way of representing a program's behavior

To make these plans even more useful, the way the programs are represented in flowcharts is standardized, which lets the people involved understand each other and the plan. The goal is to avoid the kind of thing that happened to the builders of the tower of Babel when their language was confounded! In this section, we'll look at some of the ways instruction steps are represented in standard flowcharts.

Figure 2 shows a simple flowchart for a robot programmed to follow a dark line using a light sensor and watch out for a wall up ahead using a distance sensor. The flowchart starts with a **terminator block** called "Start," which is represented by a rounded rectangle. Arrows show the flow of control through the flowchart. Just below the start block, the first thing the robot should actually do is contained in an **input/output block**, represented by a parallelogram that says "Find distance." The diamond-shaped block below it is called a **decision block**, which has two possible branches. The block uses some input to make a decision, then passes control to the correct block based on the decision. In this case, "yes" corresponds to seeing a close object, so this branch sends the flow of control to the terminator block called "Stop,"

terminator block
rounded rectangle: the beginning or end of a program

input/output block
parallelogram: get or give some information

decision block
rhombus (diamond): make a decision

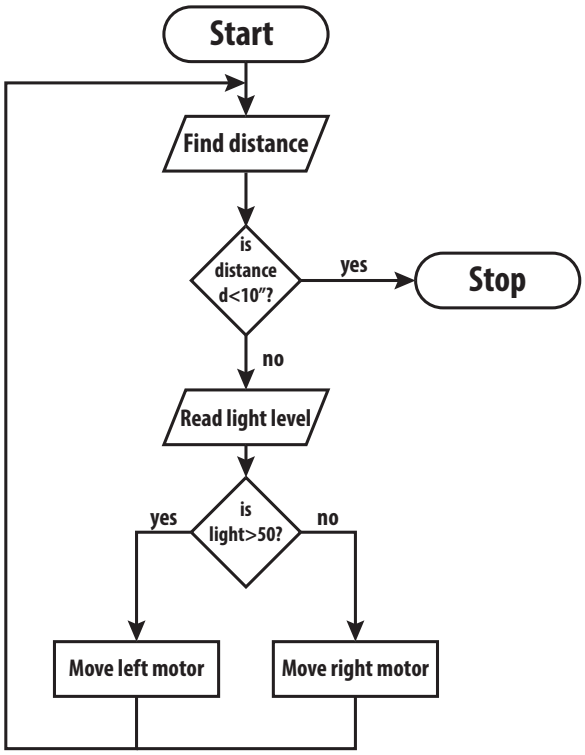

Figure 2. Like a floor plan, a flowchart gives a programmer an overall picture of the way a program should turn out. This flowchart is for a line following robot that stops when it detects an object in front of the robot.

stopping the program if an object is found. On the other hand, if no object is detected with the distance sensor, the decision block sends flow to another input/output block that tells the robot to check what the light sensor is reading. The information from the sensor is used in the second decision block, which decides whether the robot should turn left or turn right. Based on that decision, one of the two lower rectangular **process blocks** is run before passing control back to the original distance sensor check. If the wall is never found, the robot will continue to make the light/dark decision forever.

process block
rectangle: perform some action, such as a calculation

One more important type of block is the **subroutine block** (or **subprocess block**), which can be used to take the place of many instructions. It is drawn as a rectangle with two lines down each side, and Figure 3 shows this same program line follower program with the line following subroutine replaced by a subroutine block, whose contents will then be defined somewhere else on the flowchart in case anyone needs to know what it contains. Replacing the subroutine in this simple flowchart may be overkill, but using the subroutine block can cut down on clutter in complicated flowcharts.

subroutine block
rectangle with double sides: follow a set of instructions

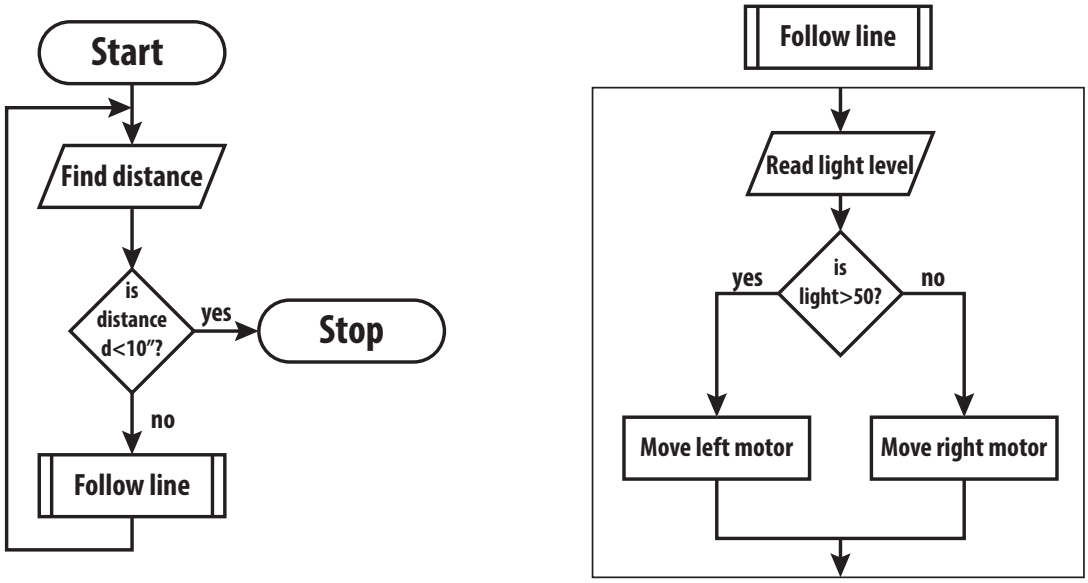

Figure 3. This flowchart accomplishes the same task as the one in Figure 2, but incorporates a subroutine called "Follow line," which is defined on the right and contains several steps in one compact block.

Programming

As mentioned before, programming involves creating a set of instructions that will accomplish a task. There are many computer programming languages, each more or less specialized for certain types of tasks in certain types of environments. Table 1 lists a few commonly-used languages and a short description for each of them.

Language	Description
FORTRAN	*Used for numeric calculations.*
C++	*General-purpose language.*
C	*Like C++ but slightly lower-level.*
Java	*Platform-independent language.*
Python	*High-level, general purpose language.*
PHP	*Used to create dynamic web pages.*
JavaScript	*Used for web applications.*

Table 1. Some programming languages and typical uses.

syntax
rules for the way a program is written

A programming language contains a set of rules for how to give instructions to a computer in a way that can be understood, called the language's **syntax**. The syntax of each language must be followed while programming to avoid ambiguity, since any ambiguity can lead to errors. Computers are very good at doing exactly what you tell them to do even if what you tell them to do isn't what you intended. Consider the difference between commands sent to the robot in Figure 4. As you spell the instructions out more and more clearly, the possibility for misunderstanding disappears.

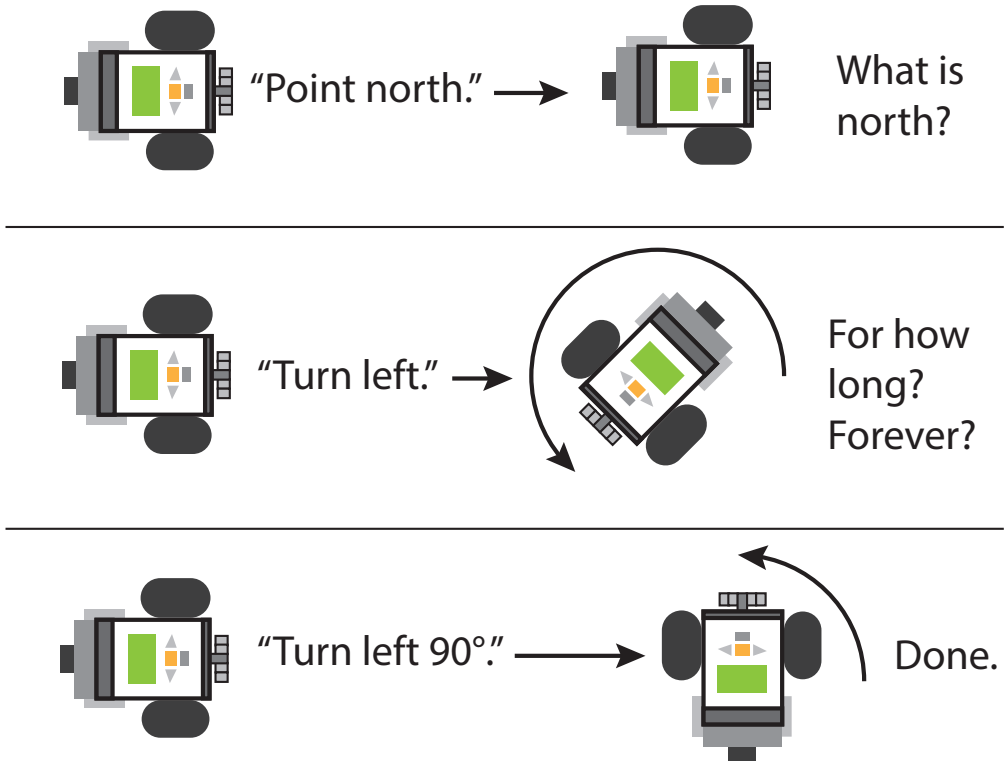

Figure 4. This shows three instructions and the robot's response to each one slightly more specific as you move down the figure.

Abstraction and interface

abstraction
hiding details of something to simplify its use

A powerful idea in computing is **abstraction**, which paradoxically involves hiding things to make the computer simpler to use. How the processor of a computer works with information hasn't changed much from the holes and cards of the Jacquard loom. The holes or lack of holes have changed into electrical signals but the math is still done one "hole" at a time using a "one" or a "zero." Everything else the computer deals with—for example, numbers larger than one, letters, symbols, images, audio, or video—can be represented by groups of ones and zeroes. The **operating system** provides a layer of abstraction by taking instructions from programs and translating them into commands written in the ones and zeroes the processor and memory are based on. Programs run by the operating system provide another layer of abstraction for the user of the computer by making it easy to give large sets of commands to the computer all at once. Most programs have what is called a **graphical user interface** (GUI), which is a set of visible inputs and outputs to let the user interact more intuitively with the program.

operating system
acts as a translator between software (programs) and hardware

GUI
visual interface to facilitate software interaction with the user

You can think of the actual processing by the computer as the inner core of an onion, with each successive layer of abstraction wrapped around the one before like a layer of the onion in order to hide distracting details. In the example from Figure 4 the motor commands would be fairly low-level commands, but you could imagine a program that took higher-level commands like "turn left 90 degrees" and translated it into the actual movements of the motors. An example of higher-level abstraction would be a program that kept track of the robot's orientation and could respond to the command "point north" by retrieving the current orientation from memory, determining how far to turn, then sending the correct rotation amounts to each motor.

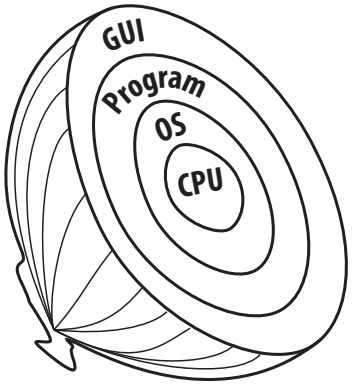

Figure 5. As we move to layers further from the core, we hide more and more details of what the computer is actually doing in a process called abstraction. The outer levels are referred to as high level, and the inner levels are low level.

Commenting

Good programming involves more than just writing the instructions in a way the computer can understand. Besides instructions for the computer, a good programmer will also add comments for other programmers that the computer ignores while running the program. These comments are for other programmers trying to understand the program or for the original programmer if they ever need to revisit the program, for example to look for errors while **debugging**. These comments should make the purpose of each part of the program clear to someone who understands the programming language it's written in, so the comments should include the same kind of things you would see in a flowchart without simply interpreting what each line of code does. It's fair to think about these comments as an effort to wrap parts of the program in a layer of abstraction. Instead of reading through a long block of code, a person can read the comments to see what information each of the program's subroutines takes in as input, what it does with the information, and what it gives as output.

debugging
identifying errors in a program and removing them

NXT-G programming environment

The software we'll be using to program our robots is called NXT-G, an example of a class of languages called visual programming languages. In NXT-G, the program itself looks something like a flowchart. When you open a new file and begin programming, there is a symbol that represents the start block of the flowchart, and there are three Lego beams extending upward, downward, and to the right. In NXT-G, the instructions are contained in programming blocks that are dragged onto the workspace and connected by the Lego-shaped sequence beams that show the flow of control. Highlighting a block by clicking it will allow access to its **configuration panel**, which shows up in the lower left part of the program window. Here you can manually change settings for the block to adjust the instructions that it gives to the robot. One other important feature of the programming blocks is the **data hub** that will fly out if you click on the lower-left hand part of the block. This panel has inputs (wired into it from the left) and outputs (wired on the right) that can communicate with other programming blocks using wires that transmit the information from outputs to inputs. While the other course materials for this week teach more about NXT-G, a few example programs are shown below.

configuration panel
allows access to a block's options

data hub
allows blocks to be wired together to communicate settings

As an example of a simple program that could be given to a robot, consider navigating the part of the obstacle course shown by a dotted line in Figure 6a. Assuming "motor B" controls the left wheel and "motor C" controls the right wheel, the program is shown as a flowchart and in NXT-G in Figure 6b. The motors spin together during the straight part of the course, then to turn the robot to the left, only motor C moves forward. Finally, the motors again spin together to move down the last section of this part of the obstacle course.

A slightly more advanced program moves the robot through the next part of the obstacle course, which is shaped like a semicircle. For this part, instead of the higher-level Move block we used in the previous example, we'll use the lower-level Motor block found on the "Complete" palette. This block controls only one motor but gives a few more options in its configuration panel. By sending less power to the Motor block that controls motor C, the robot will veer right as it moves forward. To create the program shown in Figure 6b, only one of the Motor blocks is placed on the sequence beam, and the other is dropped below the first block. An extra sequence beam can be added by holding down the shift key and clicking on an existing beam, then clicking on the beam extending from the one you want to connect. This way of arranging the blocks passes control to both Motor blocks simultaneously. Passing control to two subroutines so they can run simultaneously is called **parallel processing**. Incidentally, this turn could also have been accomplished using the Move block because the steering can be adjusted in the block's configuration panel, but we used the Motor block to see an example of parallel processing.

parallel processing
running multiple processes simultaneously

We'll go back to the Move block for this next example, Figure 6c, in which the robot needs to spiral inward while turning to the left. Here, a Variable block is used to store a number that becomes the value of the steering when it is passed to the Motor block's steering input.

If you are familiar with text-based programming, there is a more traditional (and free) way to program the NXT robots called NXC (Not eXactly C), which is based on C and NBC—a low-level language designed to run on the lego robots. There is even an Integrated Development Environment that helps program in NXC called Bricx Command Center. This class will focus on using the NXT-G (visual) environment, but you're welcome to experiment with this more traditional programming language if you're feeling adventurous.

Conclusion

Learning to program computers can be difficult but since we live in a world where so many useful things—including robots—depend on programming to receive instructions, it's worth the effort. Even if you find yourself in a profession where you're using software at the highest levels of abstraction, the more you understand what's going on "behind the curtain," the better off you'll be. You may be able to better describe and troubleshoot problems you encounter, find ways to automate something, or possibly program the robot that eventually makes your job redundant; at least you'll have an argument for why the company should keep you around for a while afterwards.

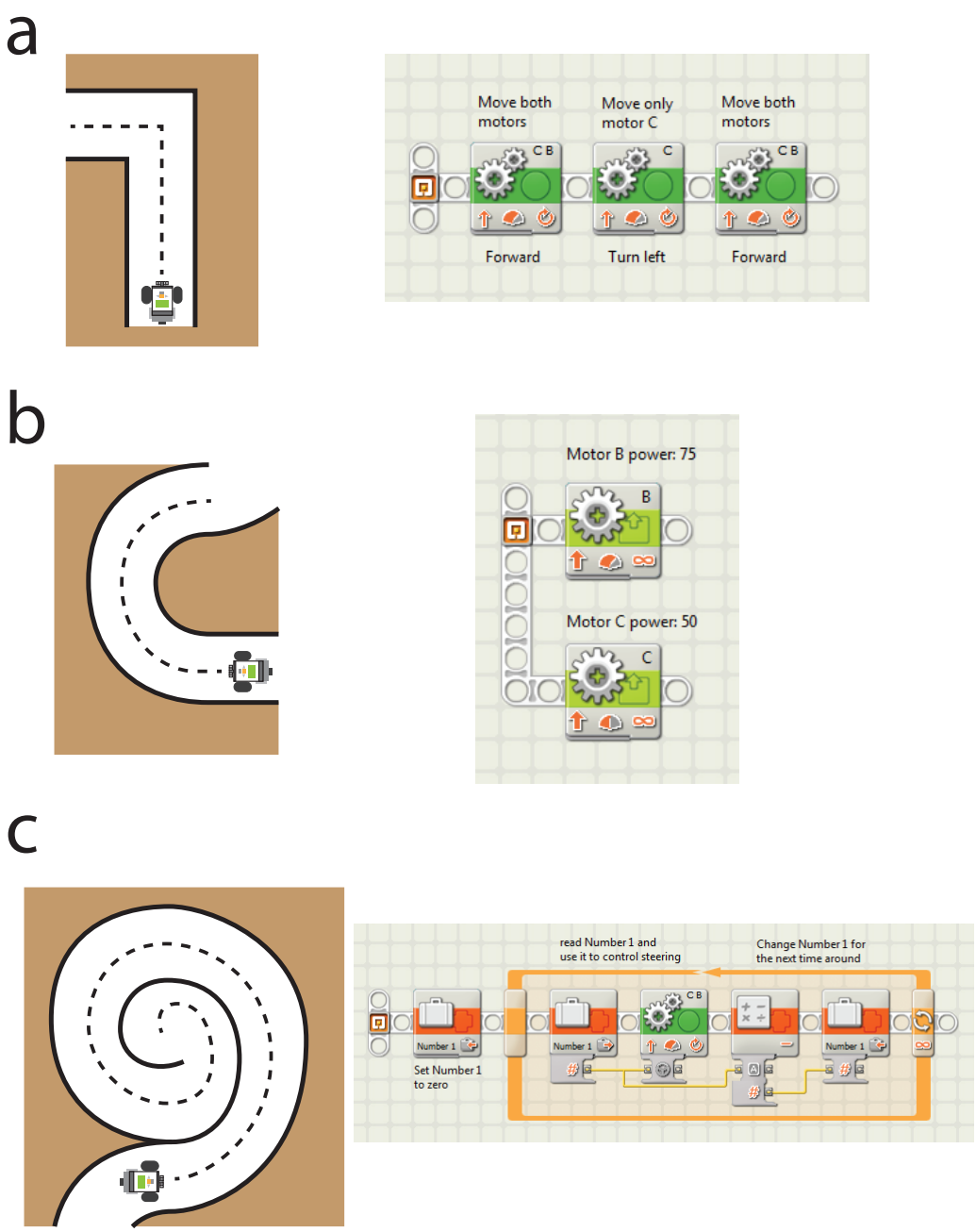

Figure 6. NXT-G programs (right) that could be used to navigate each section of an obstacle course (left).