Chapter 4
Braitenberg’s Vehicle 2

4.0 CHAPTER OVERVIEW

Three general themes have been developed in the preceding chapters. The first has been the general nature of the classical approach in cognitive science. Classical cognitive science views thinking as the rule-governed manipulation of symbols, inspired by the workings of the digital computer. As a result, its general characteristics include logicism, the manipulation of content-laden formal structures, the disembodiment of thought, and the emphasis on central control mechanisms. A second theme has been that alternative approaches are arising; these new views are reactions against classical cognitive science. In embodied cognitive science, an agent’s world is seen as an important contributor to its intelligence. The mind is said to have leaked into a world that scaffolds intelligence or thinking. Thought is not disembodied rationalism, but is instead *bricolage* in which different processes—some scaffolded, others not—are selected to solve problems at hand. A third theme has been that theories in embodied cognitive science might be best developed by using a synthetic approach. In this approach, systems are first constructed from interesting components, and then observed in action to see what kinds of interesting behaviours they can produce, and how these behaviours are affected by changing the environment in which the system is embedded.

At the end of Chapter 3, it was suggested that not only were *bricolage* and the synthetic approach important to the theories of embodied cognitive science, but these notions were also central to teaching students about the embodied approach. The purpose of Chapter 4 is to provide a
concrete example of this. It provides detailed instructions about building, programming, and observing a simple robot constructed from LEGO Mindstorms components. This robot is used to provide some hands-on experience with the themes that have been introduced in the first three chapters. This robot also sets the stage for slightly more advanced machines that are discussed in later chapters. Of particular note is the antiSLAM robot that is presented in Chapter 9, which demonstrates the navigational capabilities of a Vehicle 2 that has “evolved” to have additional sensory mechanisms.

4.1 A ROBOT’S PARABLE

4.1.1 Path of a Robot

In the parable of the ant (Simon, 1969), a researcher’s task was to explain the complex path taken by an ant along a beach. Figure 4-0 illustrates another path; one traced by a pen attached to a robot that wandered along a sheet of paper. What mechanisms are responsible for the shape of the robot’s path? This chapter explains these mechanisms.

4.1.2 Analysis and Synthesis

Braitenberg has argued that “when we analyze a mechanism, we tend to overestimate its complexity” (Braitenberg, 1984, p. 20). There is an overwhelming tendency to explain complex behaviour by attributing complex mechanisms to a behaving agent. Contributions of the agent’s environment are ignored. Noting this, Braitenberg proposed the law of uphill analysis and downhill synthesis. According to this law, theories produced by analyzing agent behaviours will be more complicated than theories created by building a situated system and observing what surprising and complex behaviours it can produce. The goal of building a robot to produce the path in Figure 4-0 can be used to illustrate Braitenberg’s point.

For example, one could analyze Figure 4-0 with the goal of writing a LOGO program that would cause the LOGO turtle (Papert, 1980) to reproduce it. The program would tell the turtle when to move forward, when to turn, when to put the pen down, and so on. Comparisons would be made between the drawing made by the turtle and Figure 4-0. Any discrepancies between the two would result in the program being modified until the LOGO turtle produced a satisfactory rendering. At this point, this program would likely be long, complex, and would make the LOGO turtle completely responsible for the drawing. The turtle’s environment would play no role.
An alternative synthetic approach would be to ignore Figure 4-0 altogether and instead build a simple system that was attracted or repelled by stimuli in the environment. In exploring the behaviour of this system, it might be discovered that a complex environment would cause the simple robot to follow the path illustrated in the figure below. In the following pages, we will show that a robot that uses two light sensors to control the speeds of two motors will follow a moving light around, and can produce the path below—in a complex environment. The path of Figure 4-0 is the result of a simple robot following the more complex path of a moving light.

4.2 BRAITENBERG’S THOUGHT EXPERIMENTS

4.2.1 A Thought Experiment

In his classic book Vehicles, neuroscientist Valentino Braitenberg explores synthetic psychology by describing, as thought experiments, a number of different robots (Braitenberg, 1984). One, called Vehicle 2, propelled itself underwater. It had two separate engines, one on each side, and two separate sensors (e.g., for measuring temperature), again on each side of the agent. The output of one sensor was used to control one motor, and the output of the other sensor was used to control the other motor, as follows: motor speed was directly proportional to the value detected by its sensor, so that when this value increased, the motor sped up, and when this value decreased, the motor slowed down.

This chapter provides instructions for building Vehicle 2 out of LEGO NXT Mindstorms components. This robot is a land-based agent that has been inspired by Braitenberg’s (1984) thought experiment.
Braitenberg (1984) argued that Vehicle 2 would generate complicated behaviour if it were embodied and situated in an interesting world. Furthermore, the kind of behaviour generated would depend upon whether a sensor was attached to the motor on the same side of the robot or to the motor on the other side.

Our incarnation is a “tractor-like” robot (Figure 4-1) that is intended to move around fairly flat surfaces.

4.2.2 Goals
Vehicle 2 is a simple robot that is fairly easy to build, to program, and to observe. It provides an ideal platform to introduce some of the concepts and skills that are central to this book.

With respect to skills, this robot provides hands-on experience with construction, including basic principles of sensors and motors. It requires the builder to also learn some simple programming skills in order to bring Vehicle 2 to life. Finally, it introduces the builder to the process of observing the agent’s behaviour, and as well as the manipulation of this behaviour by varying both the robot’s environment and some basic aspects of its design.

With respect to concepts, this robot begins to reveal the complexities of behaviour that can emerge when a simple, embodied agent is situated in an interesting, dynamic environment. Might human intelligence be derived from similar principles?
4.3 FORAGING FOR PARTS

4.3.1 Parts and Foraging

Figure 4-2 depicts the parts required to construct the version of Vehicle 2 that was illustrated in Figure 4-1. For our students, one approach to creating the robot might be to gather all of these parts prior to construction, foraging amongst the bins of available LEGO pieces.

4.3.2 Robot Bricolage

However, it is important to remember that some of these parts are not as plentiful as others, and that other robot builders are foraging for them as well. In some instances a desired part might be unavailable. In that case, the robot builder's — the bricoleur's — ingenuity must take over, and other (less desirable?) parts must be used instead. Slight deviations from the instructions might be required. As well, consistent with Braitenberg’s (1984) recognition of natural selection as a robot design principle, these deviations might result in the construction of a better robot than the one that was originally used to create this chapter of instructions.

The pages that follow provide instructions for constructing the LEGO Vehicle 2. If the reader would prefer to use wordless, LEGO-style instructions, they are available as a pdf file from the website that supports this book (http://www.bcp.psych.ualberta.ca/~mike/BricksToBrains/).
4.4 CHASSIS DESIGN (STEPS 1 THROUGH 4)

4.4.1 General Design

Our NXT version of Braitenberg’s Vehicle 2 will employ two light sensors, which in turn will control two motors, which in turn will rotate two rear wheels. The chassis of this robot is a rigid, central “spine” to which all of the other robot parts will be attached. The chassis is essentially constructed from a set of different liftarms that are held together by pins.

4.4.2 Initial Chassis Construction

The first four steps for building the chassis are illustrated in Figure 4-3, and are labelled on the left of the figure. In Step 1, a black pin (with friction) is inserted into a bent 9-hole liftarm.

In Step 2, a second bent 9-hole liftarm is attached to the pin inserted in Step 1. A length-4 axle is inserted through the axle hole on each end of the joined liftarms so that an equal amount of axle protrudes from each side. Then two pins are attached as shown.

For Step 3, slide a 2 × 4 L-shaped liftarm on the axle on the long end of the bent liftarm. At this point it will hang loosely and easily fall off, but will be attached more firmly soon. Insert a length-4 axle into the axle hole of the L-shaped liftarm as shown.

Step 4, shown at the bottom of Figure 4-3, involves sliding two more L-shaped liftarms onto the axle that was added in Step 3, and inserting two pins into each as shown in the image.
4.5 CONSTRUCTING THE CHASSIS (STEPS 5 THROUGH 7)

4.5.1 General Design

Figure 4.4 illustrates the next three steps involved in creating the chassis. In Step 5, attach a 2 × 4 L-shaped liftarm as illustrated, and insert two pins into it. Now the pieces cannot slip off the axle but will still swing freely. Complete this step by attaching two more pins in the same position in the liftarm on the other side.

To begin Step 6, slide two perpendicular axle joiners onto a length-4 axle side by side and centred on the axle. Slide the axle into the axle hole of a 2 × 4 L-shaped liftarm so that the axle joiners protrude in the opposite direction of the L and attach the L-shaped liftarm to the centerpiece as shown in Figure 4.4. Secure it with a second L-shaped liftarm and then insert a pin into each of the axle joiners.

For Step 7, connect two perpendicular axle joiners with 2-holes with a length-4 axle and attach them to the pins added in Step 6.

4.6 THE NXT INTERACTIVE SERVO MOTOR

4.6.1 The Evolution of LEGO Motors

Vehicle 2 will move on its own by activating two NXT servo motors that will rotate wheeled axles. These are the latest generation of motors provided for LEGO robots, and they have distinct advantages over their ancestors, the 9V Technic mini-motor.

One problem with this older LEGO motor is that it rotates very quickly (about 340 rpm when there is no load), but it supplies very little rotational force. As a result, even a moderate load on an axle will stall the motor. In order for the motor to supply sufficient torque to move Vehicle 2, this problem must be overcome, usually by using a gang of gears to increase the torque that is required to use the wheels to drive the robot forward.

A second problem with this older motor is that it is designed to be attached to other LEGO pieces using studs. However, half of the bottom of the motor is smooth, stud-free, and extends below the pips on the other part of the motor’s bottom. As a result, it is impossible to securely attach the mini-motor to a chassis on its own. Several additional parts are required to reinforce the motor; because of this, the motor itself does not contribute to the structural integrity of the robot.

4.6.2 The NXT Servo Motor

The NXT servo motor (Figure 4.5) that we will use in this chapter has been explicitly designed to solve both of these problems (Astolfo, Ferrari, & Ferrari, 2007). In terms of its internal structure, this motor has a
built-in gear train of eight gears that produces a substantial gear reduction. As a result, while the maximum rotational speed of this motor is half of the older mini-motor (170 rpm versus 340–360 rpm, as reported on http://www.philohome.com/motors/motorcomp.htm), the NXT servo motor delivers 3–4 times the mechanical power. This reduces the need for additional gears to be added to the robot.

In terms of its external structure, an NXT servo motor was designed not only to be incorporated into a studless design (i.e., a design that uses studless liftarms or beams instead of studded bricks), but also to provide structural support when used (Astolfo et al., 2007). As can be seen in Figure 4-5, the motor has one built-in 3-hole beam near its narrow end, and two such built-in beams at the opposite end of the motor. As well, there are three holes perpendicular to the pair of built-in beams that accept pins.

In addition to the built-in gear reduction and studless connectivity, the NXT servo motor has an internal rotation sensor. It is an optical encoder that counts the rotations of the motor shaft, and is accurate to 1° of rotation (Astolfo et al., 2007). The motor is interactive in the sense that while the NXT brick can send commands to turn the motor on, it can also receive signals from this rotation sensor, and use these signals to offer precise motor control. For example, one could use this sensor to determine when the motor is in a particular rotational position, or to impose relational properties, such as synchronization, on two or more motors. Our Braitenberg Vehicle 2 will not require exploiting this internal rotation sensor, but we will take advantage of its existence for more complex robots that we will discuss later in this book.

4.7 ADDING MOTORS TO THE CHASSIS (STEPS 8 AND 9)

Construction Step 8, shown in Figure 4-6 below, completes the chassis of our NXT Braitenberg Vehicle 2. It involves attaching a perpendicular axle joiner with double holes on to each end of the axle added in Step 7. Then insert pins into the holes as shown in the figure. Later, these pins will be used to attach the NXT brick.

The chassis is now ready to have two NXT servo motors attached to it, as shown in Step 9 below. Push the two neighbouring pins on one side of the chassis into the holes in the motor near the orange motor output. Then insert the pin and adjacent axle into the holes on the rear. With the motor attached the centrepiece will no longer swing freely. The

100 FROM BRICKS TO BRAINS
second motor is then added to the other side of the chassis, mirroring the first. Note how the structural integrity of the chassis is due to the rigidity of the motors—the motors are part of the chassis too!

4.8 **ADDING A FRONT SLIDER (STEP 10)**

4.8.1 Passive Front Support

The front slider attaches to the rear of the motors, which will be the front of the robot. It further secures the two motors, and will also be used to help attach the NXT brick. The front slider supports the robot as it moves. Importantly, using a slider (instead of a front wheel) means that all of the robot’s turning will be due to differences in the speeds of the two motors.

4.8.2 Constructing the Front Slider

Step 10, the construction of the front slider, is illustrated in Steps 1 and 2 of the subassembly shown in Figure 4-7. To begin, slide three perpendicular axle joiners onto a length-5 axle, as shown. Then attach a double-hole perpendicular axle joiner on to each end of the axle so that the double holes are oriented perpendicular to the holes of the three single-hole axle joiners (see image). In the centre single-hole axle joiner insert a pin with bush so that the bush is on the same side as the double-hole axle joiners’ holes. Insert a long pin into the other two axle joiners and two long pins into each of the double-hole axle joiners as in the image.
Next, attach a 7-hole beam on to the long pins; it should have two holes protruding on either side. Insert a pin into each of the holes on the end. The pins protruding from the beam will serve as the second attachment point for the NXT brick. The last step in the centrepiece is to insert a length-6 axle into the stop bush, to add a large wheel centre to the end of the axle, and to attach the entire slider to the chassis as illustrated.

4.9 CONSTRUCTING REAR AXLES (STEP 11)

4.9.1 Wheel Axle Design

Step 11 of our robot construction involves building the wheel axles. The orange motor output on the NXT motors has an axle hole in the centre and four pin holes around it. We can take advantage of the pin holes to add some extra strength to our wheel axles.

4.9.2 Constructing the Wheel Axles

There are three steps to constructing wheel axles, as shown in the sub-assembly illustrated in Figure 4-8. In Step 1 slide a 3-hole beam on to a
length-8 axle. In Step 2, slide two bushes onto the axle and insert two long pins with stop bushes into the empty holes of the beam as shown in the Figure. For Step 3, insert a length-3 axle into each of the stop bushes and add a second beam of the same length. Remember to use these steps to build two axle assemblies, one for each motor. Once constructed, they can be attached to the robot as shown in Figure 4-9.

4.10 ATTACHING THE NXT BRICK (STEP 12)

4.10.1 The NXT Brick

Our Braitenberg Vehicle 2 requires that light sensor inputs be converted into motor speed outputs. In our robot, such sense–act connections are mediated by the NXT brick. This brick is a small computer that has four different input ports and three different output ports; it also has a USB port to connect it to a computer in order to download programs onto it. The heart of this device is a 32-bit ARM7 microprocessor. The NXT brick can be powered by six AA batteries, or by a rechargeable lithium battery pack. It is possible to connect three different NXT bricks together, and to have communication between them, in order to develop a more complicated robot. However, our Vehicle 2 is simple enough that only one of these bricks is required.

4.10.2 Attaching the Brick

In Step 12, shown in Figure 4-9, the NXT brick is connecting to the chassis by attaching the holes on its back to the pins on top of the chassis. Make sure that the screen is on the same side as the front slider! Add two pins to each side of the NXT brick as illustrated. They will serve as attachment points for the sensor mounts.

4.11 ATTACHING LIGHT SENSOR SUPPORTS (STEP 13)

4.11.1 Sensor Mount Design

For Vehicle 2 to be situated in its environment it requires sensors. Braitenberg’s (1984) description of Vehicle 2 is general enough to permit a variety of different sensors to be employed. For our robot, we will use light sensors because they are very responsive to changes in light and
give a simple output that is easily used to control motor output. The light sensors are mounted on axles and can be slid back and forth and angled inward. They are easily changeable to allow for experimentation on how the embodiment of the robot affects its behaviour. Step 13, shown in Figure 4-10, illustrates how to construct the basic structure to which light sensors will be attached. A 1 × 11.5 double bent liftarm is attached to each side of the NXT brick using the pins that were inserted in Step 12. Then a length-12 axle is inserted into the end of each liftarm as shown in the figure.

4.12 ADDING LIGHT SENSORS (STEP 14)

4.12.1 Mounting Light Sensors

In Step 14, shown in Figure 4-11 below, a mount for each light sensor is constructed and then is mounted to the axles added in Step 13.

There are four steps involved in building a light sensor mount; these steps are illustrated in the subassembly part of Figure 4-11. Start by connecting a stack of two 1 × 1 plates to the end stud of a 1 × 4 brick with holes (Step 1). Duplicate this piece for the second mount.

Proceeding to Step 2, attach a 1 × 3 locking joint to the each brick and add a second 1 × 4 brick on top. In Step 3, insert 2 pins into each of the bricks as shown in Figure 4-11. Note that the position of the pins in the right sensor mount mirror the position of the pins in the left sensor mount.

Finally, attach a light sensor and a perpendicular axle joiner with two holes as shown in the image. The light sensors have two diodes protruding from the front. The pale pink one emits red light when the sensors are in reflected light mode but does not when in ambient light mode. For this robot the sensors will be set to measure the ambient light.
4.13 WHEELS AND CABLE CONSIDERATIONS

4.13.1 Completing the Robot

Step 15 (shown in Figure 4-12) is the final stage of constructing Vehicle 2, in which we add wheels onto the axles and connect the cables. The right motor should be connected to port C and the left motor should be connected to port B.

The options for connecting the light sensor cables are more interesting. One can make the relationship between sensors and motors *contra-lateral* — the sensor on the robot’s left controls the speed of the motor on the robot’s right, and the right sensor controls the left motor’s speed. This is accomplished by connecting the left sensor to input port 4 and the right sensor to input port 1 (Figure 4-13). Note the crossing of the cables in Figure 4-13.
A second approach to connectivity would be to enforce an *ipsilateral* relationship between sensors and motors, where the left sensor controls the left motor, and the right sensor controls the right motor. To accomplish this, connect the left sensor to input port 1 and the right sensor to input port 4 (Figure 4-14 or Figure 4-12). Note that in these two figures the sensor cables do not cross.

We will see that whether the sensor-motor relation is contralateral or ipsilateral has a huge impact on the behaviour of our Vehicle 2. However, in order to see this in our assembled robot, we must first create a simple program that mediates the relationship between sensed light and motor speed.

### 4.14 Sensing, Acting, and the NXT Brick

**4.14.1 The NXT Brick**

What is the point of building Vehicle 2? It is to start to explore the kinds of surprising or complex behaviours that might emerge from a simple embodied agent that is situated in an interesting environment. The notion of agent at this point is intended to mean a constructed robot that is capable of sensing some properties of its environment, and to directly convert these properties into certain actions. That is, the agent is defined by its sense–act cycles (Pfeifer & Scheier, 1999) — essentially, by its reflexes, which directly link sensing to acting and do not require any intermediary thinking.
The implementation of these reflexes in Vehicle 2 is the job of the NXT brick that is mounted on top of it. That is, the NXT serves as the interface between LEGO sensors and motors (Figure 4-15). While, as described below, the NXT is a small digital computer, we will avoid considering it as a “thinking” component of a sense–think–act cycle. This is because the simple code that we develop to link sensing and acting could be implemented by replacing the NXT with hardware components that perform the same function as the NXT and our code.

At the heart of the NXT is a small computer, which employs a 32-bit central processing unit (CPU)—Atmel's ARM7 microprocessor. This system has access to four input ports and three output ports, is capable of analog-to-digital conversion using an ARV coprocessor. This computer is small—it has available only 64 kb of random-access memory (RAM). The NXT’s operating system (its firmware) is held in a 256 kb FLASH memory.

In Vehicle 2, analog-to-digital converters built into the NXT convert the sensor readings into a usable form. That is, the sensor readings will be converted into a form that the program we write for the robot can translate into an output signal to be delivered from the NXT’s output ports to the motors mounted on the rear of the robot.

The NXT uses pulse-width modulation (PWM) to control motor speed. When a signal is sent from the output port, it is a pulse of constant amplitude. Motor speed is varied by altering the duration of this pulse. For example, to run the motor at 35% speed, a pulse is sent that is on for 35% of 128µs cycle that is standard for the brick, and then the output is switched off for the remaining 65% of the cycle (Gasperi, Hurbain, & Hurbain, 2007). In order to increase the speed of the motor, there is a decrease in the proportion of the cycle during which a pulse is sent. Because the speed of the motor is determined by the average voltage that it receives, there is a linear relationship between speed and applied voltage.

Pulse-width modulation works quite nicely to achieve turning behaviour in our Vehicle 2 when the two motors are running at different speeds because of differences between the two light sensors. As noted earlier, the servo characteristics of the motor provide more sophisticated means of control that will be exploited in later chapters, but which are not required for this particular robot.

4.15 NXT LIGHT SENSOR PROPERTIES

4.15.1 The LEGO Light Sensor

Programming the desired sense–act cycles into Vehicle 2 is helped by having a reasonable understanding of the workings of the device that situates the robot, the NXT light sensor shown in Figure 4-16. This device
is an analog sensor that measures the intensity of light that it receives. Its sensitivity covers a fairly broad range, from 0.5 lux (where a lux is the measure of lumens per square meter) to 500 lux (Prochnow, 2007).

While this device is treated as a light sensor when used in Vehicle 2, it actually is more sophisticated. It can be used as an active sensor. As an active sensor, it consists of two functional components. One is a phototransistor that measures incoming light. The other is a light-emitting diode that generates light. In active mode, the light sensor activates the LED for a short period of time, and then it measures returning light with the phototransistor. It is continually oscillating between sending light and receiving light.

The light sensor can also be configured as a passive sensor, which is its typical usage in Vehicle 2. As a passive sensor, the LED is never turned on, and the light sensor uses its phototransistor to measure the intensity of the ambient light in the environment.

The LEGO light sensor has several properties that can make it tricky to work with. It has peak sensitivity to light in the infrared range of the spectrum, and is less sensitive to shorter wavelengths of light. This means, for example, that it will see incandescent light bulbs as being brighter than would be experienced by a human observer. As well, as battery power changes, the behaviour of the light sensor is affected. As a result, light sensors are affected by changes in ambient light, by the reflective properties of environmental objects, and by decreasing battery power. Some of these sensing nuances might be sources of an interesting set of robot behaviours. However, it might be desirable to control some of these properties to some extent to reduce light sensor fluctuations.

One approach to controlling the light sensor involves deciding how it will be read by the NXT brick. The analog-to-digital conversion that the NXT brick performs on its input ports can be processed in a number of different ways.

For instance, one mode for taking light sensor readings is RAW. When this is done, the digital representation of sensor output is used directly. When the light sensor is detecting very bright light, readings will be values in the order of 300, while very dark conditions will produce readings around 1023.

Alternatively, one can set the NXT brick to deliver readings in PERCENT mode. When this mode is used, the brightest light produces a value of 100, while darkness produces a value of 0. Another interesting
mode for the light sensor is BOOLEAN; in this mode the light sensor only provides a signal when a change in light is detected!

While the default mode for light sensors is PERCENT, other modes might provide greater control over the variability of light sensor behaviour, and their effects on robot behaviour are worthy of exploration.

Of course, in addition to exploring the modes in which a sensor is read, Vehicle 2’s behaviour can be affected by whether the light sensors are active or passive. For instance, in a highly reflective environment, active sensors will provide additional light sources, and will produce different behaviour than will passive sensors!

4.16 PROGRAMMING THE NXT BRICK

4.16.1 Programming Steps

With the robot constructed, and armed with some understanding of the brick and the light sensor, we are now in a position to create a program that will mediate light sensor readings and motor speeds. This requires choosing a programming language, and creating some code in that language. This merely involves writing a text file on a desktop computer, where the contents of the file are the lines of the program. This text file is then processed by a compiler, which converts the text into a form that can be executed by the brick. Finally, the executable code is downloaded from the computer to the NXT brick using a cable that LEGO provides.
4.16.2 Programming Environment

Because of the popularity of LEGO Mindstorms robots, there are a number of different programming environments that integrate code creation, downloading, and other activities involving the brick. LEGO provides one environment with a Mindstorms kit, and many others are available on the internet as freeware.

We have elected to program the NXT brick using a language called Not eXactly C (NXC), and to do so in a programming environment called BricxCC, which is available from http://bricxcc.sourceforge.net/. This environment permits NXC code to be typed and saved, provides aids for debugging code, permits the code to be compiled and downloaded, and provides a number of other useful tools for programming and examining the brick. It also comes with a comprehensive set of help files that provide instruction in using the various menu items for BricxCC, as well as a complete manual describing the NXC language. Figure 4-17 illustrates BricxCC loaded with the Vehicle 2 program to be described in the following pages.

4.17 A SIMPLE MAIN TASK

4.17.1 The Main Task

Any NXC program requires that one or more tasks are defined. A task defines a standalone operation, in the sense that it is assumed that more than one task can be running at the same time.

One of the tasks in the program must be called “main.” For Vehicle 2, the main task initializes the light sensors, controls what information is displayed on the LCD screen of the NXT brick, and starts two other tasks.

The listing at the bottom of this page provides the main task written for our robot, and begins with the declaration “task main() {”. The first two lines of the task tell the NXT that two of its input ports are going to be connected to light sensors, and that these sensors are passive. The next two lines request the NXT brick to process the signals from these two sensors in PERCENT mode. The next two lines start two additional tasks, called “DriveLeft” and “DriveRight,” which will be discussed shortly. The only point to note here is that they are initiated by the main task.

4.17.2 Defining Variable Names

The main task that is listed below uses variable names like “LeftEye” and “RightEye” that make the program easier to read. These “plain English” terms are established using a set of #define statements that are also part of the program. Our Vehicle 2 program uses four of these statements, which are given below. For instance, the first one lets us use the term
LeftEye in place of S1, which is how NXC typically names input port 1. The third line lets us use the term LeftMotor instead of OUT_B, which is what NXC usually uses to represent output port B.

//Plain English definitions.
#define LeftEye S1
#define RightEye S4
#define LeftMotor OUT_B
#define RightMotor OUT_C

Note that these variable definitions will occur outside of any task, and are usually the first bit of code in any program.

### 4.17.3 Miscellaneous Syntax

The listing below provides many examples of NXC syntax; the semicolons and the use of parentheses are critical, and more information about syntax can be found in the NXC documentation. BricxCC also provides utilities to help keep proper NXC syntax. The indenting and commenting of the code is helpful for understanding it, and is recommended practice, but is not required for the code to function properly.

//Main task. Turn on the eyes (but not their LEDs) and start the tasks.
task main(){
    SetSensorType(LeftEye, SENSOR_TYPE_LIGHT_INACTIVE);
    SetSensorType(RightEye, SENSOR_TYPE_LIGHT_INACTIVE);
    SetSensorMode(LeftEye, SENSOR_MODE_PERCENT);
    SetSensorMode(RightEye, SENSOR_MODE_PERCENT);
    start DriveLeft;
    start DriveRight;
} // end task

### 4.18 Linking Light Sensors to Motors

#### 4.18.1 Two More Tasks

The main task described in Section 4.17.1 started two additional tasks, two variables DriveLeft and DriveRight. These two tasks are used to convert a light sensor reading into a motor speed; one task links one sensor-motor pair, the second sensor-motor pair is linked by the other. The listing of each of these tasks is provided below.

The two tasks are identical, with the exception that DriveLeft processes LeftEye and LeftMotor, while DriveRight processes RightEye and RightMotor.
When they are started by the main task, they will both be working at the same time. These tasks are written assuming ipsilateral connections between sensors and motors; however, if one changes the cabling to produce contralateral connections, the code does not have to be altered to keep the robot working.

To understand these two tasks, let us consider \texttt{DriveLeft} alone. The first line of this task states a \texttt{while (true)} loop. This construction initiates an infinite loop, so that the task repeatedly carries out any commands that are in between the \{ and \} of the loop’s syntax.

The operations in the infinite loop of \texttt{DriveLeft} work as follows: First, \texttt{OnFwd} is a command that turns a motor on, rotating in a forward direction. This command needs to specify which motor, and which speed. The motor that this command affects is the \texttt{LeftMotor}. The speed of this motor is going to be sent as a percentage (where 100\% would be full speed, and 0\% would be full stop). This percentage is determined by reading the light sensor. The \texttt{Sensor(LeftEye)} command reads the light sensed by the \texttt{LeftEye} as a percentage; it is this percentage that is used as the speed of the motor. The brighter the light detected by this sensor, the faster the motor; the motor will slow down as less light is detected by this sensor.

The \texttt{DriveRight} task proceeds in exactly the same manner, using the other input port to determine the percentage speed associated with the motor attached to the other output port.

```c
//The next two tasks run in parallel and constantly feed the values of each eye into the respective motor as a speed.
task DriveLeft(){
    while(true){ // Run the LeftMotor at the LeftEye's speed
        OnFwd(LeftMotor, Sensor(LeftEye));
    } // end while loop
} // end task

// end task

task DriveRight(){
    while(true){ // Run the RightMotor at the RightEye's speed
        OnFwd(RightMotor, Sensor(RightEye));
    } // end while loop
} // end task

4.19 A COMPLETE PROGRAM

The listing below is a complete example program for our Vehicle 2, and is available from the website that supports this book (http://www.bcp.psych.ualberta.ca/~mike/BricksToBrains/). The complete program is simply a concatenation of all of the components that we have been describing.
in the preceding sections. This program would exist on a computer as a text file, and could then be downloaded to the NXT brick by a utility such as BricxCC (available at http://bricxcc.sourceforge.net/). Once the utility compiles and downloads the program into the robot that we assembled, the sense–act cycles that define Vehicle 2 will come to life, and the robot should be capable of demonstrating the behaviours that Braitenberg imagined in his 1984 thought experiment.

```c
//Lego NXT Braitenberg Vehicle 2 code - Brian Dupuis 2008
//Plain English definitions.
//Note that these are arbitrary; LeftMotor may
//actually be wired to the right input port.
#define LeftEye S1
#define RightEye S4
#define LeftMotor OUT_B
#define RightMotor OUT_C
//The next two tasks run in parallel and constantly feed the values of each
//eye into the respective motor as a speed.
task DriveLeft(){
    while(true){ // Run the LeftMotor at the LeftEye's speed
        OnFwd(LeftMotor, Sensor(LeftEye));
    } // end while loop
} // end task
task DriveRight(){
    while(true){ // Run the RightMotor at the RightEye's speed
        OnFwd(RightMotor, Sensor(RightEye));
    } // end while loop
} // end task
//Main task. Turn on the eyes (but not their LEDs) and start the tasks.
task main(){
    SetSensorType(LeftEye, SENSOR_TYPE_LIGHT_INACTIVE);
    SetSensorType(RightEye, SENSOR_TYPE_LIGHT_INACTIVE);
    SetSensorMode(LeftEye, SENSOR_MODE_PERCENT);
    SetSensorMode(RightEye, SENSOR_MODE_PERCENT);
    start DriveLeft;
    start DriveRight;
} // end task
```

Chapter 4  Braitenberg's Vehicle 2  113
4.20 EXPLORING VEHICLE 2 BEHAVIOUR

4.20.1 Three Test Environments

With Vehicle 2 constructed and programmed, all that remains to do is to explore the behaviour of this robot. Vehicle 2 is most apt for moving about a flat world with few obstacles, because it has no sensors to detect obstacles, or reflexes to avoid them. Vehicle 2’s only sensors are light detectors; it is ideally suited for environments in which light sources are present.

4.20.2 A Simple World

From the preceding instructional pages, it is obvious that Vehicle 2 is a very simple robot. When it is placed in a simple environment, its behaviour is also quite simple. This is demonstrated in the early segments of Video4-1.mpg, available from the website for this book (http://www.bcp.psych.ualberta.ca/~mike/BricksToBrains/). When the robot’s world consists of a single light, ipsilateral motor connections cause the light to be avoided, and contralateral connections cause the robot to move toward the light. A slight change of embodiment in this simple world produces an interesting array of behaviours. For instance, the light sensors can be angled so that their receptive fields overlap. With this embodiment—and contralateral connections—the robot avoids lights when they are far away, but attacks them when they are nearby. Overlapping receptive fields also result in Vehicle 2 spiralling toward a light over a period of time.

4.20.3 A More Complex World

The parable of the ant (Simon, 1969), and the law of uphill analysis and downhill synthesis (Braitenberg, 1984), claim that simple devices can generate complex behaviour. Their ability to do so is contingent upon being situated in a world, and also depends upon the complexity of that world. One can increase the complexity of a robot’s behaviour by increasing the complexity of its environment, without manipulating the robot at all.

The final segments of Video4-1.mpg illustrate this principle. The video illustrates a number of ways in which the environment was made more complicated, producing complex behaviours that did not require a different program to be created for the robot. Vehicle 2 demonstrates colour preferences and the ability to follow another machine—provided that the second machine has a light source mounted on it. In a dark room, the robots move slowly, and come to a stop underneath a hanging light source. They are “awakened” when the room is illuminated, and
actively explore their world. If overhanging lights begin to swing, then the robots move away. Multiple robots appear to compete for resources. Imagine having to explain all of these behaviours analytically, without having direct knowledge of the robots’ embodiment or programming. Would the theory that resulted be as simple as the synthetic theory represented by the instructions on the previous pages?

4.20.4 Complexities via Embodiment

A number of computer simulations of Braitenberg vehicles are available on the internet, such as Thornton’s POPBUGS package, available at http://www.cogs.susx.ac.uk/users/christ/popbugs/braitenbergs.html. Why, then, would we go to the trouble of embodying Vehicle 2 as a LEGO artifact? The physical structure of the robot itself is another source of complexity. Computer simulations of Braitenberg vehicles are idealizations in which all motors and sensors work perfectly. This is impossible in a physically realized robot. Slight manufacturing differences will mean that one motor may not be as powerful as another, or that one sensor may be less sensitive than another. Such differences will affect robot behaviour. These imperfections are another important source of behavioural complexity, but are absent when such vehicles are created in simulated and idealized worlds.

4.21 FURTHER AVENUES FOR BRICOLEURS

4.21.1 Exploring Embodiment

The preceding instructions define one possible Vehicle 2. Of course, many alternative versions of this robot can be explored. For instance, alternative Vehicle 2 robots can be created by exploring alternative robot embodiments. One could start with minor changes of the existing robot: how does it behave when light sensors are slid to different positions? What happens when the light sensors are angled in different directions?

More elaborate *bricolage* involves redesigning some of the robot’s structure. What occurs when the front slider is replaced with a balanced wheel, or a wheel that isn’t completely balanced, or with a wheel that is not able to rotate a full 360°? What is the result of using different sensors, such as temperature sensors?

4.21.2 Manipulating Environments

One of the lessons of Vehicle 2 is that changing the robot is but one avenue to changing its behaviour. One can also manipulate behaviour by modifying the environment, while leaving the robot alone. The robot’s
environment can be explored by changing the number and location of lights, or by adding mirrors. The type of light can also be manipulated: the light sensors used in our version of Vehicle 2 are highly sensitive to infrared light, and therefore can process signals from remote controls used for televisions! Recognizing that an embodied robot is part of its world, light sources could be attached to the robot chassis, and more than one robot run at the same time.

4.21.3 Modifying Code

Yet another avenue for robot development would be to modify the robot’s program. The program reads the light sensors in PERCENT mode. What might be the effect of reading the sensors in RAW mode, and then doing some sort of processing of these readings that does not involve percentages? For example, one could compare raw sensor readings to each other, or to some standard value, or to an average light reading that is updated by the robot. As well, the robot’s light sensors could be initialized to be active; they would then become additional sources of light that could affect behaviour, particularly if the environment contained surfaces that reflected the LED emissions.

4.21.4 Bricolage, Not Design

All of the avenues mentioned above consider robot exploration from a synthetic perspective. That is, robot bricoleurs take some available components of interest, use them to modify or elaborate a machine, and then observe the result in order to understand what the robot can or cannot do within an environment that is also being manipulated. The robot is created first, and produces data of interest. This is in contrast to the more analytic approach, where a theory (e.g., a robot) is constructed on the basis of existing data (e.g., the robot path introduced at the start of the chapter in Figure 4.0).

The synthetic approach could be criticized in the sense that it is not goal directed. However, the success of the synthetic approach is derived from the components that are explored. As the minimalist composers found with their music, if the effort is made to begin with an interesting set of component mechanisms, then the resulting product should provide interesting or surprising results. The next few chapters will attempt to illustrate this by providing accounts of a variety of different robots constructed from LEGO components.