CHAPTER 15

Object Detection

TOPICS IN THIS CHAPTER

- Object Detection
- Feature Detectors
- Feature Listeners

The leJOS developers wanted a standard set of classes to detect objects and obstacles. We came up with the package lejos.robotics.objectdetection in order to generalize the task of object detection. We want to allow our navigation classes to deal with objects that might be detected in the environment. But since there are multiple sensors and methods for detecting objects, it is important for our API to be able to deal with not only standard object detection sensors (such as ultrasonic sensor) but also custom systems users might create on their own. With a generic object detection API, a user can hack together as many object detection strategies they like and use them in our standard navigation classes.

Object Detection

Object detection allows a robot to detect objects in its path and take some action, such as avoiding the object. We chose the word feature to describe an object rather than obstacle because sometimes the robot is seeking an object out, rather than avoiding it (such as a soccer ball or robot combatant). The word object is not used because it was a potentially confusing class name with object oriented Java programming.

Feature Detectors

A FeatureDetector reports on objects it detects using a sensor, such as a touch sensor or ultrasonic sensor. It is the main interface in the object detection package from which data originates. There are many benefits of using a FeatureDetector:

- Automatic scanning and reporting of data
- A listener interface to segregate the action-response code
- Allows a single block of code to respond to data from

Two implementations of FeatureDetector in our API are:

- RangeFeatureDetector - uses RangeFinder classes, such as the LEGO ultrasonic sensor.
- TouchFeatureDetector - uses Touch classes, such as the LEGO touch sensor.

The RangeFeatureDetector allows you to choose some parameters for the sensor, such as the maximum range you want it to report findings for, and the time between performing scans. You can construct a simple RangeFeatureDetector as follows:

```
int MAX_DISTANCE = 50; // In centimeters
int PERIOD = 500; // In milliseconds
UltrasonicSensor us = new
UltrasonicSensor(SensorPort.S4);
FeatureDetector fd = new
RangeFeatureDetector(us, MAX_DISTANCE, PERIOD);
```

Once you have a FeatureDetector instantiated, such as fd above, you can perform a scan on it to retrieve data:

```
Feature result = fd.scan();
if(result != null)
  System.out.println("Range: " + result.
getRangeReading().getRange());
```

NOTE: Make sure to check for a null object before trying to read data from the returned Feature object, otherwise your code will throw a null pointer exception.

The main benefit of a FeatureDeteector is the ability to automatically notify other classes when an object is detected via a listener interface. The next section discusses this in more detail.

Feature Listeners

Once you have a FeatureDetector instantiated, you can add a FeatureListener to it. The FeatureListener code is notified when an object is detected via FeatureListener.featureDetectreact to the detected object by performing some sort of action. The following code shows how to use a FeatureListener:

```
RangeFeatureDetector fd = new
RangeFeatureDetector(us, MAX_DETECT, 500);
fd.addListener(listener);
```

A more complete example of using a FeatureListener is included in the full code sample below.

Feature data

As you saw above, data is returned from FeatureDetectors in a Feature object. This is actually an interface which defines the basic requirements of a Feature object. The most basic class to implement the Feature interface is the RangeFeature class. This class is a data container, and the data is retrieved by these methods:

`getRangeReading()`

- Returns a RangeReading object

`getRangeReadings()`

- Returns a RangeReadings collection

`getTimeStamp()`

- Returns the system time (in ms) when this data was collected

Some scanners are capable of detecting multiple objects, such as the LEGO ultrasonic sensor. Other sensors are only capable of detecting a single object, such as a touch sensor. In either case, they are both capable of producing data to fulfill the first two methods listed above.

How? When the getRangeReading() method is called, it returns the closest object that was detected by a scanner, even if it is capable of returning more than one hit.

When the getRangeReadings() method is called, it returns turning one hit, the RangeReadings object will contain only one RangeReading.

Handheld Range Meter

Let's create a simple but fun project to demonstrate the object detector and listener interface. The device is simply an ultrasonic sensor attached to the NXT brick (see Figure 15-1). Using a short cable, plug the sensor into port 4.

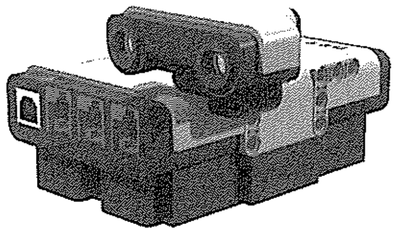
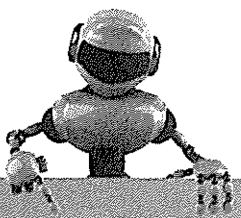


Figure 15-1: Building the simplest project in the book

The code below uses the RangeFeatureDetector constructor to limit the distance objects are reported. The code uses 150 cm, but you can change this as you wish. By default, it reports readings every 500 ms, but you can change this too.

```
import lejos.nxt.*;
import lejos.robotics.objectdetection.*;

public class ObjectDetect implements
FeatureListener {

  public static int MAX_DETECT = 150;

  public static void main(String[] args)
throws Exception {

    // Instructions:
    System.out.println("Autodetect ON");
    System.out.println("Max dist: " + MAX_DETECT);
    System.out.println("ENTER = do scan");
```

```
   // Initialize the detection objects:
   ObjectDetect listener = new ObjectDetect();
   UltrasonicSensor us = new
UltrasonicSensor(SensorPort.S4);
   RangeFeatureDetector fd = new
RangeFeatureDetector(us, MAX_DETECT, 500);
   fd.addListener(listener);

   // Disable default button sound:
   Button.setKeyClickVolume(0);

   // Button inputs:
   while(!Button.ESCAPE.isPressed()) {

    // Perform a single scan:
    if(Button.ENTER.isPressed()) {
     Feature res = fd.scan();
     if(res == null) System.out.
println("Nothing detected");
     else {
      listener.featureDetected(res, fd);
     }
     Thread.sleep(500);
    }

    // Enable/disable detection using buttons:
    if(Button.RIGHT.isPressed()) {
     if(fd.isEnabled()) {
      Sound.beepSequence();
      System.out.println("Autodetect OFF");
     } else {
      Sound.beepSequenceUp();
      System.out.println("Autodetect ON");
     }
     fd.enableDetection(!fd.isEnabled());
     Thread.sleep(500);
    }
    Thread.yield();
   }
  }

 public void featureDetected(Feature feature,
FeatureDetector detector) {
   int range = (int)feature.
getRangeReading().getRange();
   Sound.playTone(1600 - (range * 10), 100);
   System.out.println("Range:" + range);
```



Just for fun!

The handheld range meter is an interesting project for testing the performance of the ultrasonic sensor. Try pointing it at a wall and then changing the angle (see Figure 15-2). Notice that if you go over a certain angle it no longer returns a reading? That is because the sound is being deflected away from the sensor instead of reflecting back towards it. Also, try pointing it at a soft or furry object, like a dog. Dogs are almost 'radar invisible' to the ultrasonic sensor, much like a stealth plane.

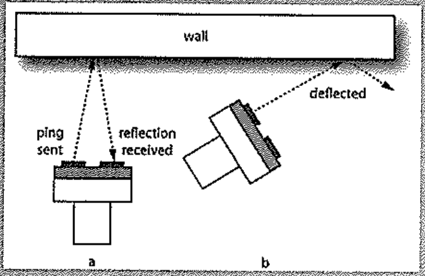


Figure 15-2: Ultrasonic ping deflecting away from the sensor

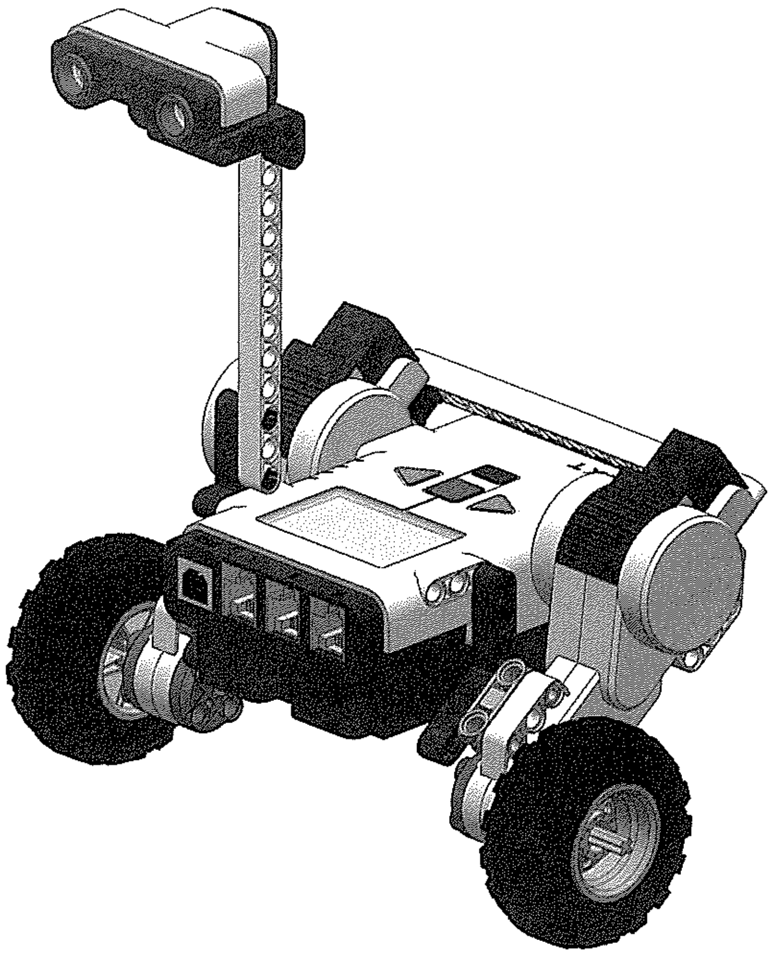
Once you run the program you can walk around your house like Spock with a tricorder, pointing it at different objects and watching the distance readout on the LCD. You can also disable the auto-detect listener by pressing the right arrow button. In this mode, simply press enter to take a single reading.

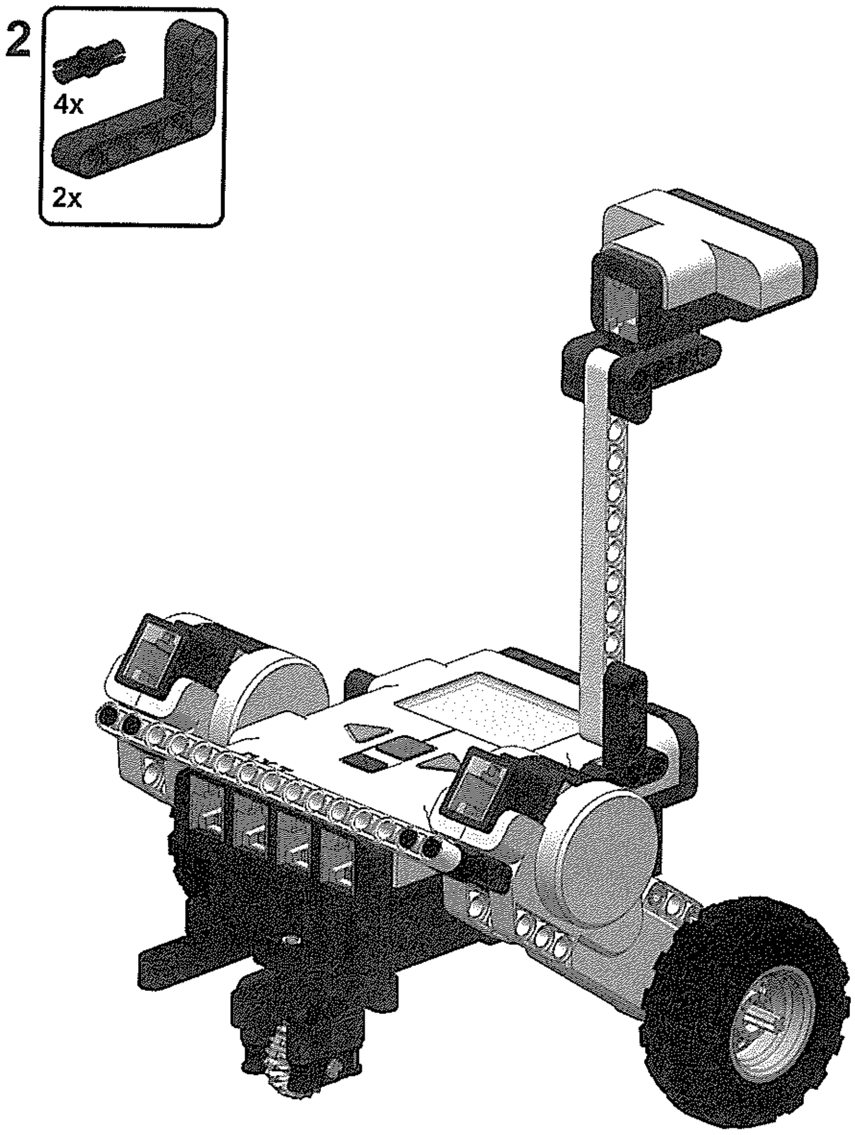
Rear Bumper Robot

It is also possible to combine several different FeatureDetectors into one FeatureDetector using the FusorDetector class. This is useful for robots that have a number of sensors located around the robot, such as several bumpers at different locations, plus several range sensors. By doing this, it allows one FeatureListener to respond to all of the sensors of the robot.

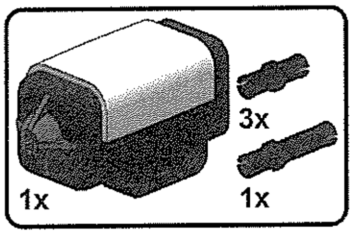
To test this out, we'll need to mount at least two sensors to a robot. This example uses the Carpet Rover with an ultrasonic sensor and a rear bumper. Start with the Carpet Rover equipped with an ultrasonic sensor (see Figure 4-6). Then continue with the instructions below:

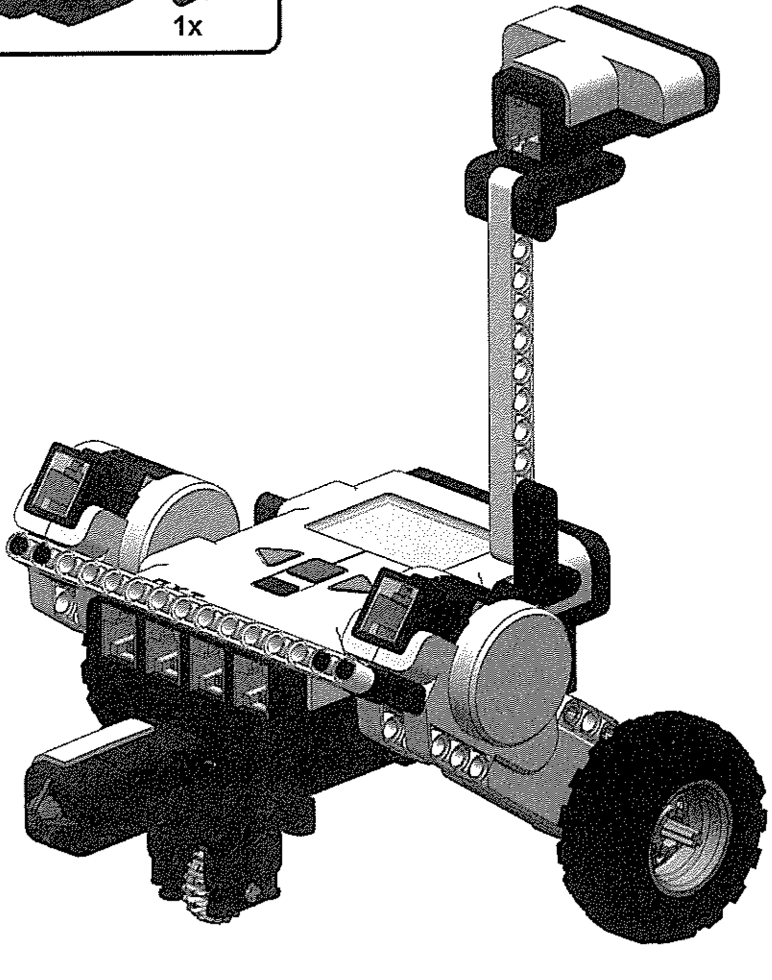
1



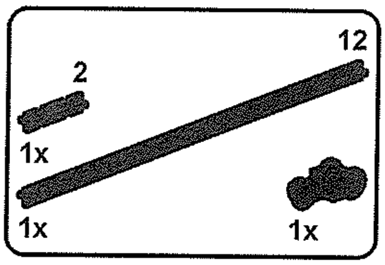

2
4x
2x

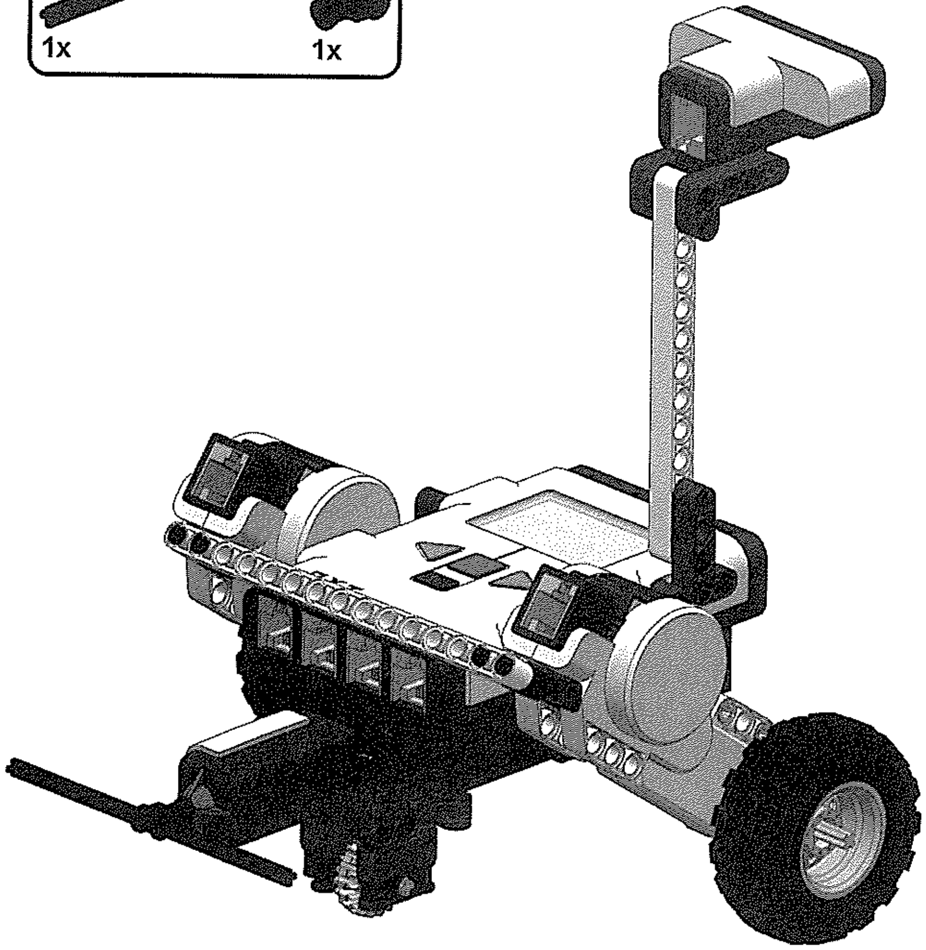
3



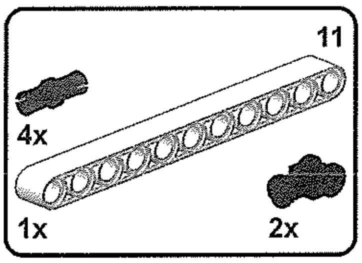


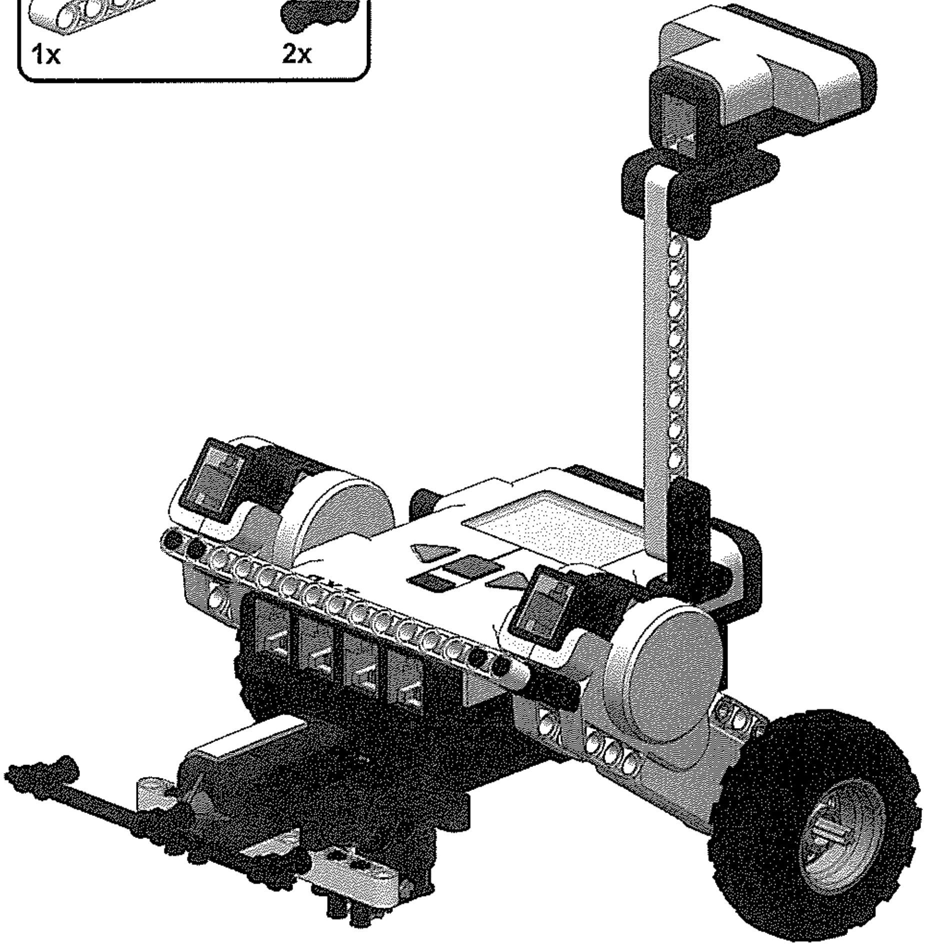
4



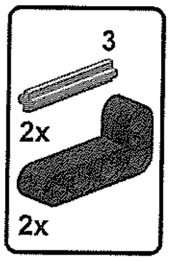


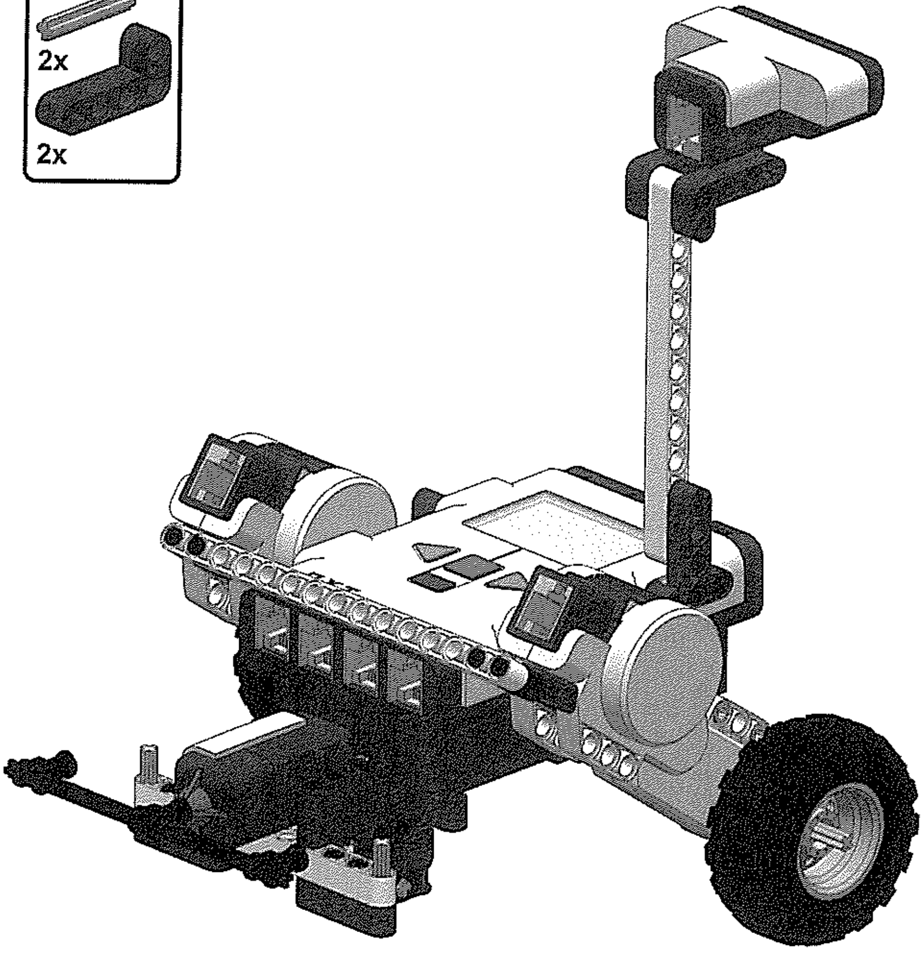
5



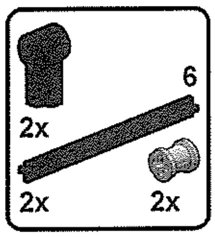


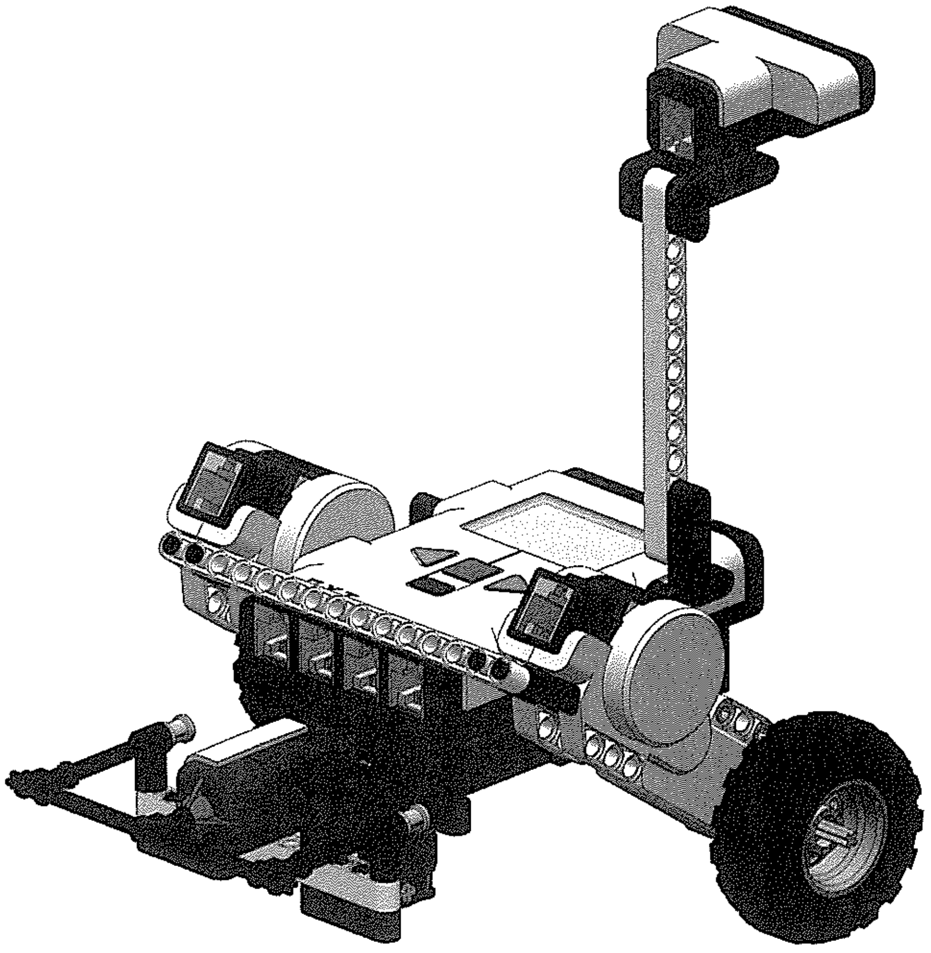
6





7





Now enter the following code and upload it to the robot.

```
import lejos.nxt.*;
import lejos.robotics.Touch;
import lejos.robotics.navigation.
DifferentialPilot;
import lejos.robotics.objectdetection.*;

public class BumperBot implements
FeatureListener {

  private static final int MAX_DETECT = 50;
  private static final int RANGE_READING_
DELAY = 500;
  private static final int TOUCH_X_OFFSET = 0;
  private static final int TOUCH_Y_OFFSET = -14;

  private DifferentialPilot robot;

  public BumperBot() {
    robot = new DifferentialPilot(4.32, 16.35,
Motor.B, Motor.C, false);
    robot.forward();
  }

  public static void main(String[] args )
throws Exception {
    UltrasonicSensor us = new
UltrasonicSensor(SensorPort.S4);
    FeatureDetector usdetector = new
RangeFeatureDetector(us, MAX_DETECT,RANGE_
READING_DELAY, 90);

    Touch ts = new TouchSensor(SensorPort.S1);
    FeatureDetector tsdetector = new
TouchFeatureDetector(ts, TOUCH_X_OFFSET,
TOUCH_Y_OFFSET);

    FusorDetector fusion = new
FusorDetector();
    fusion.addDetector(tsdetector);
    fusion.addDetector(usdetector);

    fusion.addListener(new BumperBot());

    Button.waitForAnyPress();
  public void featureDetected(Feature feature,
FeatureDetector detector) {
    System.out.println("R:" + feature.
getRangeReading().getAngle());
    if(feature.getRangeReading().getAngle() >=
0) {
      detector.enableDetection(false);
      robot.rotate(90 * Math.random());
      detector.enableDetection(true);
      robot.backward();
    } else {
      robot.forward();
    }
  }
}
```

Plug the touch sensor into port 1 and the ultrasonic sensor into port 4. This code makes the robot drive forward until it detects an obstacle. Then it rotates to a new direction and reverses until the rear bumper detects an object.

Watch the bumper car carefully to see how long it survives before something goes wrong. Try to note what it is that goes wrong. Alter the code to make the robot more robust as it navigates. Or, maybe you need to physically modify the bumpers? The goal is to create consistency, repeatability, longevity and robustness in your robots.