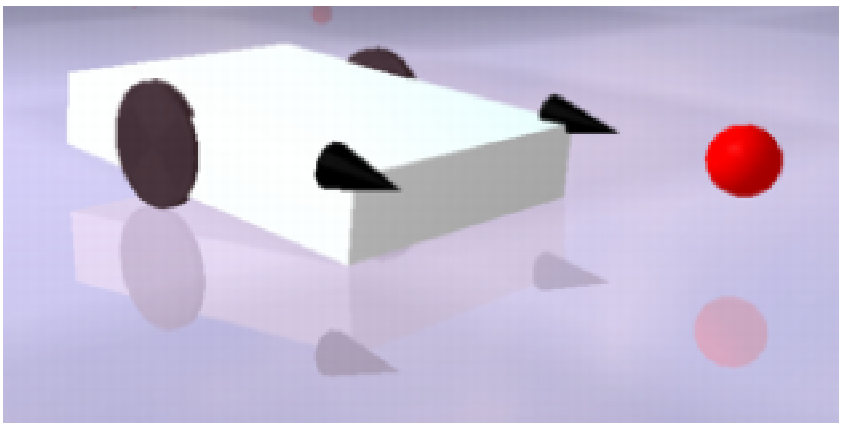
Building Braitenberg Vehicles in breve

22nd July 2004



a tutorial

The breve Simulation Environment

http://www.spiderland.org

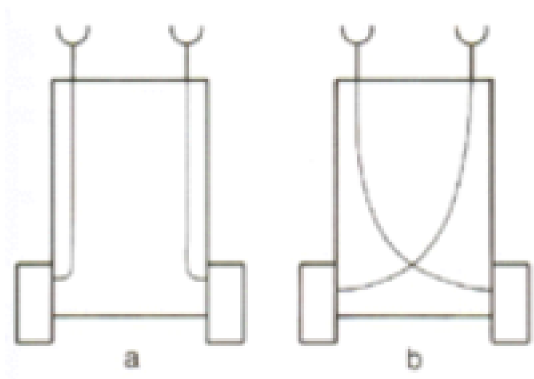


Figure 1: two simple Braitenberg vehicles

Introduction

Braitenberg vehicles are small robots that can exhibit complex behaviors with very simple circuitry. The vehicles typically have small box-shaped bodies with one wheel on each side. At the front of the vehicle are sensors, which detect different types of stimuli (for example, light) from the environment. These sensors are connected directly to the vehicle's wheels so that the wheels turn when the sensors are activated. Two simple Braitenberg vehicles are shown in figure 1.

The concept is simple, but Braitenberg showed that these vehicles could appear to exhibit interesting and complex behaviors such as "love" or "hate", depending on how the sensors were connected to the wheels. This tutorial describes how to create simulated Braitenberg vehicles in breve. Follow the directions outlined in the following steps in order to create a basic Braitenberg vehicle. You should read through each step in its entirety before making changes to the code. You should also be careful to type in each line exactly as it appears in this tutorial—punctuation and capitalization are especially important.

Step 1: Open a Braitenberg template file

The first step in creating a Braitenberg Vehicle in breve is to launch the breve application (version 1.5 or later) and open a template file for Braitenberg vehicles. You can find this template by selecting the menu item "Demos -> Braitenberg -> BraitenbergTemplate.tz". This should open a window containing the following text:

```
1   @use Braitenberg.
2
3   Controller myBraitenbergControl.
4
5   BraitenbergControl : myBraitenbergControl {
6      + variables:
7          vehicle (object).
8          leftSensor, rightSensor (object).
9          leftWheel, rightWheel (object).
10         light (object).
11
12     + to init:
13         light = new BraitenbergLight.
14         light move to (10, 1, 0).
15
16         vehicle = new BraitenbergVehicle.
17         self watch item vehicle.
18   }
```

This is a template for a Braitenberg vehicle simulation. What does the code do so far? All breve simulations require a *controller* object, which will specify how the simulation is set up. We tell breve the name of the controller at line 3. At line 5, we start to define the object, called myBraitenbergControl. Inside this controller object, we create a method called *init* (starting at line 12). This method will be run automatically when the object is created—so anything inside the method will be automatically run when the simulation starts. Inside this *init* method, we do two things: we create a BraitenbergLight object (line 13), and we create a BraitenbergVehicle object (line 16). In addition, we associate names with these objects by assigning them to some of the variables we've specified. Later on, we can refer to these objects by their names—the BraitenbergVehicle object will be known as *vehicle* and the BraitenbergLight object will be known as *light*.

If you click on the "Play" button, you should see the simulation start, and that the light and the vehicle are added to the simulated world (though the light is off-camera to the right slightly). Stop the simulation and return to your code to begin the next step.

Step 2: Add wheels to the vehicle

You probably noticed that the vehicle you created had no wheels. You might be saying that vehicles without wheels are, in fact, not vehicles. But instead of arguing about words, let's do something about it. Let's add wheels to our vehicle.

In order to do this, we'll need to "tell" the vehicle to add some new wheels. Remember that we've attached the name *vehicle* to our vehicle, so here's how we tell it to create a new wheel:

```
vehicle add-wheel at (X, Y, Z).
```

X, Y, and Z will specify the location of the new wheel on the vehicle—remember that we're working in 3-dimensional space.

A typical Braitenberg vehicle might have wheels halfway between the front of the vehicle and the back, halfway between the top and the bottom, with one on each side. In breve, Braitenberg vehicles are 4.0 units long, .75 units high and 3.0 units wide. The X, Y and Z values we pick have to be relative to the center of the vehicle, which has the coordinates (0, 0, 0). So relative to the center of the vehicle, what are the X, Y and Z coordinates of our wheels?

The first two measurements are easy, since we want them to be halfway between the front and back, and halfway between the top and bottom, they should already be lined up with the middle of the vehicle. This means that the X and Y values should be 0. As for the Z values, we'll want one wheel on the left side, and one wheel on the right. Since the vehicle is 3.0 units wide, our Z values will be 1.5 and -1.5.

So we'll want to add two lines to our program, after all the other commands in the init method (but before the "}" line):

```
vehicle add-wheel at (0, 0, -1.5).
vehicle add-wheel at (0, 0, 1.5).
```

As with the vehicle and the light, we'll need to attach names to them, so we'll modify these lines slightly:

```
leftWheel = (vehicle add-wheel at (0, 0, -1.5)).
rightWheel = (vehicle add-wheel at (0, 0, 1.5)).
```

Run the simulation again. The vehicle should now have wheels. Stop the simulation and proceed to step 3.

Step 3: Add sensors to your vehicle

Adding sensors to your vehicle is a lot like adding wheels, so we won't go into too much detail. Sensors are added with the command:

```
vehicle add-sensor at (X, Y, Z).
```

This time, however, we want the new objects placed at the front of the vehicle, with one on each side. So what values should be used for X, Y and Z this time? You'll probably end up with commands something like this:

```
vehicle add-sensor at (2.0, 0, -1.5).
vehicle add-sensor at (2.0, 0, 1.5).
```

As with the code we used to create the wheels, we'll put these commands after all of the others in the init method. And once again, we'll want to attach names to the sensors so that we can use them later. So modify the sensor lines to look like this:

```
leftSensor = (vehicle add-sensor at (2.0, 0, -1.5)).
rightSensor = (vehicle add-sensor at (2.0, 0, 1.5)).
```

Run the simulation again, and you should see the new sensors attached to the vehicle. Stop the simulation and proceed to step 4.

Step 4: Link the sensors to the wheels

At this point, we have a fully assembled Braitenberg vehicle, but it doesn't move! In order to make the vehicle react to stimuli in the environment, we'll need to link the sensors to the wheels. We have the names of the sensors and the names of the wheels, so we just tell each sensor to link itself to a wheel. We can link the sensors to either wheel, linking either to the wheels on the same side, or on opposite sides. If we wanted to simulate Braitenberg's *Aggressor*, for example, we would link the sensors to the opposite wheels:

```
leftSensor link to rightWheel.
rightSensor link to leftWheel.
```

Run the simulation again and you'll see that the vehicle now moves. The vehicle rolls toward (and then through) the light. You can even use the arrow tool to select the light and move it away from the vehicle. You'll see that the vehicle will chase after it, as long as it's visible to the vehicle's sensors. Stop the simulation and proceed to step 5.

Step 5: Customize the sensors & wheels

Now we've got a functioning Braitenberg vehicle! By customizing the configurations of the sensors and the wheels, we can get all sorts of interesting behaviors. We'll customize the vehicle by changing the starting velocity of its wheels and by changing the strength of the connections between the sensors and wheels.

On the vehicle we've created so far, the wheels start with a speed of 0, and then accelerate in response to sensor input. The speed of a wheel in the absence of light is called the *natural velocity* of the wheel in breve. We can tell each wheel to set its natural velocity like this:

```
leftWheel set-natural-velocity to .5.
rightWheel set-natural-velocity to .5.
```

This means that the wheels will start with a speed of .5 even when there is no sensor input. Of course, the natural velocities we give the wheels do not need to be equal, nor do they need to be positive. Choosing asymmetrical or negative natural velocities can lead to more interesting vehicle behaviors.

As for the strength of the connection between a sensor and a wheel, this value will determine how strongly the wheel will react in response to sensor input—the higher the value, the stronger the reaction. In breve, the strength of this connections is called the *sensor bias*. We can change the bias associated with a sensor like this:

```
leftSensor set-bias to 3.0.
rightSensor set-bias to 3.0.
```

In this example, we set the bias to be three times as strong as the starting value (which defaults to 1.0). This means that the vehicle will react more strongly to the light, making it faster and making its behavior more dramatic.

As with the natural velocity of the wheels, we can choose any bias values we'd like. By using negative bias values, we cause the wheels to slow down (and even turn backwards) in response to light. Picking different bias values for the left and right wheels will cause "asymmetrical" vehicle behaviors.

Step 6: Things to try...

- ... add additional lights to make the vehicle move in a specific pattern, like a figure-eight or a circle.
- ... modify the weights, wheel positions and velocities to create new behaviors, like PARANOID, WHIMSICAL or CONFUSED.
- ... program the lights to move around by themselves (see the documentation for the class “Mobile”).
- ... continually modify weights and wheel velocities over the course of the simulation (add an “iterate” method to the myBraitenbergControl class).
- ... add a second vehicle with a different behavior.