A Braitenberg vehicle simulator with pseudo-physics

late project — Embodied Intelligence

Eddie Kohler*

This paper describes a Braitenberg vehicle simulator called xbraitenberg. *The simulator can implement arbitrary Braitenberg vehicles in a configurable, pseudo-physical world.*

1 Braitenberg vehicles

Valentino Braitenberg introduced his vehicles in a little book published by the MIT Press in 1984 [1]. He describes these vehicles as thought experiments, the result of a "process of purification" that occurred as he considered the problem of mind. From a collection of simple parts—motors, sensors, and some wired-up logic—he creates phenomena that observers would identify as intelligence, and makes a clear case that intelligence is an observed condition, not an innate one.

Braitenberg situates his vehicles in an in-between world, half physical and half simulation. He relies on friction, for example, but introduces wires (Mnemotrix and Ergotrix) that have no analogue in the real world. The first vehicles he builds "conjure up images of vehicles swimming around in the water, while later what comes to mind are little carts moving on hard surfaces" [1, page 2]; yet these are only images, for the thought experiment does not require perfect correspondence with reality.

This makes Braitenberg vehicles perfectly suited for computer simulations. Modeling the real world in a computer simulation is impossible, but Braitenberg's vehicles live in a simpler space—one that Braitenberg controls, or that we control. We can add analogues of physical laws and properties as we like, until the simulation becomes too slow or the world is interesting enough to satisfy us.

Several Braitenberg vehicle simulators are already available on the Web. However, these simulators involve very simple worlds. They often don't bother modelling anything but the most simple physical properties. They all model the inverse-square dissipation of light energy, but few of them even model acceleration—vehicle motor output is directly proportional to vehicle velocity. Mark Ziler's Shockwave simulator [8] models one vehicle in a world with four lights; there is no friction and no acceleration. Chris Gerken's Java simulator [3] models many vehicles, but in a similar world (no friction, no acceleration, no collisions between vehicles). John Wiseman's simulator [7] is written in Lisp; while the code is not available, from his demonstrations, it seems that the same constraints apply (no friction, acceleration, or collisions). They also seem to apply to Torsten Will's Java simulator [6], Chris Thornton's POPBUGS simulator [5], and Robin Edwards's C simulator [2], although I couldn't get Edwards's simulator to run. These simulators differ in implementation language, the number of vehicles they support, the kinds of sensors the vehicles can have, and the way the vehicles' brains are built, but not in the complexity of the world—although Braitenberg himself writes early in his book that friction, among other real-world physical properties, should have a role in his vehicles' construction [1, page 19].

*This paper was originally written in spring 2000 for Rodney Brooks's 6.836 class at MIT.

This paper presents a simulator that models friction, acceleration, and collisions, as well as dissipation of light and other physical properties. *Xbraitenberg*'s vehicles are swimming around on an infinite body of water. They can have an arbitrary number of sensors and motors, pointed in arbitrary directions, and arbitrary logic connecting them. They can have headlights. When two vehicles collide, they bounce off one another like billiard balls. The world's friction and angular friction constants are under user control.

Of course, simulating the real world is an impossible task, and *xbraitenberg* does not correctly model many physical properties. For example, there are no waves or wakes in the "water", there is no air resistance, collisions between vehicles are elastic, vehicles cannot be broken, sensors have a uniform sensing profile, and motors can fire arbitrarily fast (although the user can control this). Nevertheless, it implements enough real behavior that the vehicles feel appreciably more real than the simulators mentioned above; and furthermore, we can investigate how the world's parameters change the ways vehicles behave.

2 Program overview

Xbraitenberg runs on Unix under the X Window System. A version of it was written for the first problem set in 1999; this year, a later version was downloaded and used by many students for the first problem set. *Xbraitenberg* creates an X window that displays the vehicles and any other objects in the world. Figure 1 shows a sample *xbraitenberg* screenshot. The screen displays two lights (the two standalone circles) and six vehicles. (Two of the eight original vehicles have already left the screen.) The line behind each vehicle is its *trail*, which shows where the vehicle has been.

The user can use keystrokes to pan through the world; to zoom in and out; to pause or resume motion; to start the simulation again with the vehicles in different, random positions and orientations; to toggle the display of trails; and to output the current display as an Encapsulated PostScript file. The right half of Figure 1 shows such a PostScript file. These displays, rather than screenshots, are used for the remainder of this paper.

The user can control the contents of the world with command line options and by hacking the code. Unfortunately, interactive world manipulation is not supported—there are no dialog boxes, and the vehicles and other objects cannot be clicked or dragged. Through the command line, the user can select vehicles from a number of types, determine whether they should have headlights, set up other fixed objects such as lights, set the vehicles' motor logic, and set characteristics of the world like friction coefficients, among other things. More detailed changes, such as adding motors or sensors to a vehicle or creating new fixed lights, must be done in source code.

The source code itself consists of around 7000 lines of C++. (Several thousand of these are from reusable libraries.) It roughly falls into four categories: graphic display, command line parsing and world creation, code for the objects that populate the world, and code for managing the world generally.

3 Vehicle basics and display

The most important objects in any Braitenberg vehicles system are the vehicles themselves. This section describes the vehicles' parts and shows how they are displayed.

Each vehicle has a chassis, an arbitrary number of motors, an arbitrary number of sensors, logic connecting the motors and sensors, and finally, optional attachments. In terms of the simulation's physics, the chassis is circular, but it is displayed as a box (see Figure 2). The motors are shown as smaller boxes

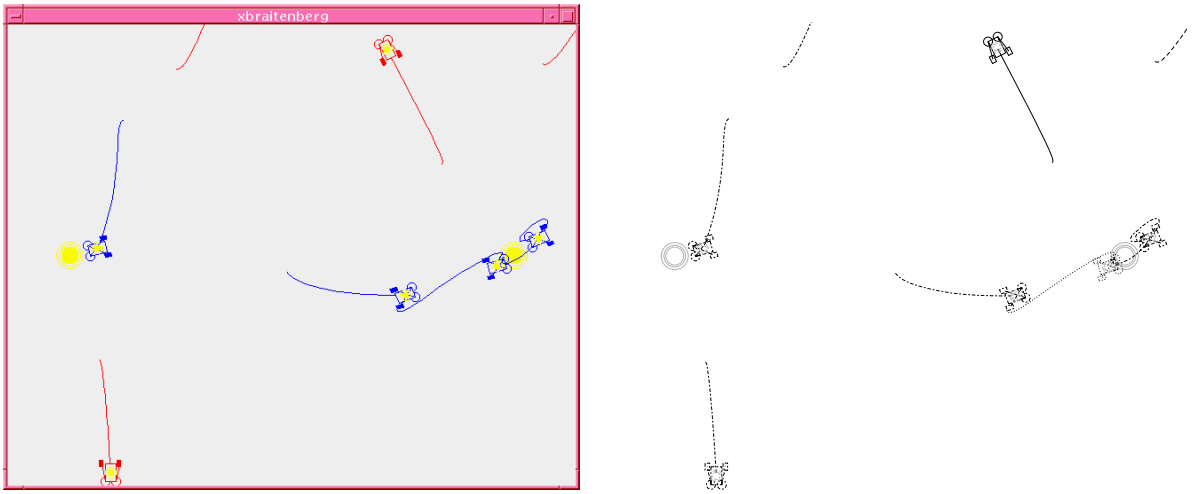

Figure 1: A sample *xbraitenberg* run with four Vehicles 2a, four Vehicles 2b, and two lights. Two of the Vehicles 2a, which are light avoiders, have already left the screen. A color screen display is on the left; the Encapsulated PostScript file generated for that screen display is on the right. (The actual screen display has a black background.)

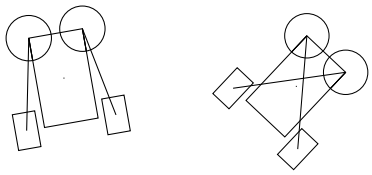

Figure 2: Vehicles 2a and 2b

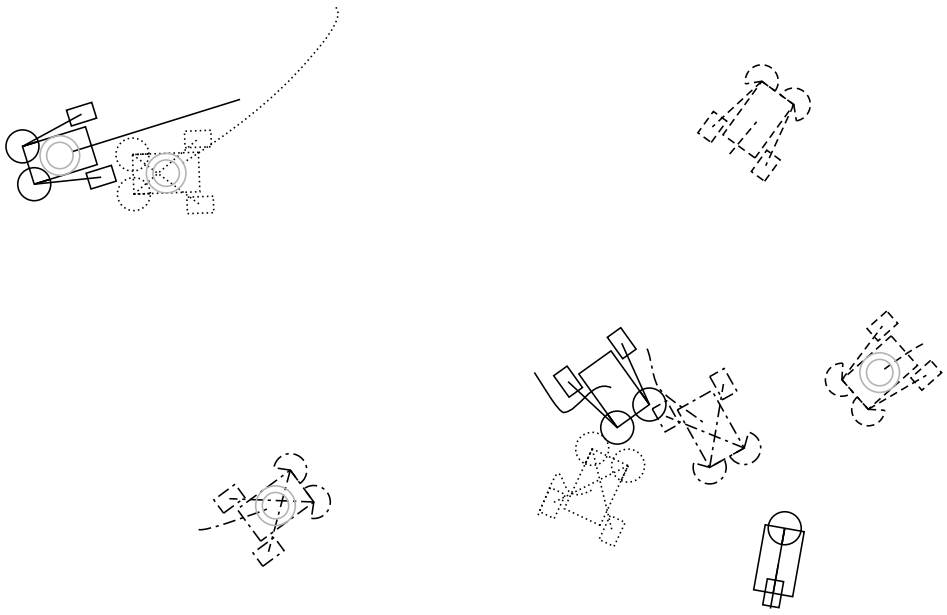

Figure 3: Vehicles with and without headlights, and with and without angle-limited sensors. Also, one instance of Braitenberg's Vehicle 1.

hanging off the chassis; the sensors are shown as circle sections; and the logic is shown as wires connecting motors and sensors. *Xbraitenberg* implements one important kind of optional attachment, a headlight, which is shown as some grey circles centered on the chassis.

The sensors detect light. (The system can be extended to any kind of sensation. We have built sensors that detect heat, such as heat generated from vehicle motors, but we only present light detection here.) Sensors can optionally have a limited angular range—that is, they can ignore signals that are not coming from some "pie slice" in front of the sensor. Limited sensors are shown as arcs, rather than complete circles, where the arc's extent shows the range the sensor can see. (See Figure 3 for some examples.) Sensors have uniform responsiveness along their angular range and zero responsiveness outside it, rather than, say, tapering off near the range's borders.

Each motor's response is controlled by a logical expression, often involving sensor values. The sensor have names; on most vehicles, the sensors are called "left" and "right". The user can supply their own logical expressions written in a C-like language. For instance, the command line option "`-left-logic 'left + 0.5*right'`" says that the left motor's response should be proportional to both sensors, but the left one has priority. Braitenberg's Vehicles 2a through 3b correspond to the following logics:

	Left logic	**Right logic**
Vehicle 2a (sensation excites)	`pin(left)`	`pin(right)`
Vehicle 2b (sensation excites)	`pin(right)`	`pin(left)`
Vehicle 3a (sensation inhibits)	`.3*(1-pin(left))`	`.3*(1-pin(right))`
Vehicle 3b (sensation inhibits)	`.3*(1-pin(right))`	`.3*(1-pin(left))`

(The `pin` function pins its argument between 0 and 1.) For each motor, lines are drawn between that motor and all sensors mentioned in its logical expression. This gives some visual feedback about how the vehicle is programmed, and corresponds to the diagrams in Braitenberg's book.

4 Vehicle physics

Xbraitenberg's vehicles act like water skippers or hovercraft, or hockey pucks on an infinite field of ice. Specifically, the vehicles' contact with the ground is uniform—there are no wheels, for example—and they are uniformly able to move in any direction. They have a mass, which is constant for all vehicles described in this paper. They both move and spin; that is, they have both velocity and angular velocity. Physically, they behave like circular masses of uniform density, although they are displayed as complex rectangular devices.

A vehicle's motors give it acceleration and angular acceleration (rather than directly setting its velocity). Each motor fires linearly in a single direction, like a jet engine, and has some offset and rotation relative to the center of the vehicle's body. The motor's firing applies both force and torque to the vehicle. The force's direction depends on the motor's rotation; the amount of torque depends on the motor's position and rotation.

The motion of each vehicle is hindered by friction. The world has four friction constants: static friction, dynamic friction, and two "angular friction" constants. The static and dynamic friction constants are classical physics approximations. The static friction constant determines how much force must be applied before the vehicle starts moving at all; the dynamic friction constant determines the strength of the friction force, which acts against the vehicle's motion. The two angular friction constants approximate the same functions, but for the vehicle's angular motion instead of its translation motion. They have no classical physics analogues. You might think the proper thing to do would be to integrate the normal friction on every point of the vehicle's underside. However, static and dynamic friction constants are empirical approximations anyway; angular friction is just another approximation, and not necessarily a worse one. While not sanctioned by elementary physics textbooks, this was recommended by a real live physics Ph.D. [4]

The friction experienced by the vehicle is independent of how fast it is moving. This is correct physically. (In the real world, there are oppositional forces that are larger the faster a vehicle moves—namely, air resistance—but *xbraitenberg* does not model any of these yet.) However, the angular friction "force" has a component that is proportional to angular velocity. We initially left these independent, but it did not dampen the angular velocity enough—vehicles always ended up spinning in place like mad dervishes.

The vehicle's acceleration equals the total force on the vehicle—obtained by summing the force vectors for each motor and the friction force—divided by its mass. The vehicle's angular acceleration is calculated based on its torque, the angular friction, and its moment of inertia. The actual calculation is shown in Appendix A.

Vehicle collisions are elastic: a simple, billiard-ball-like model. When two vehicles collide, momentum and kinetic energy are both conserved, and the vehicles themselves are not damaged. Furthermore, collisions do not currently affect angular velocity—even a glancing blow won't change a vehicle's orientation. Despite these limitations, collisions give the simulation an interesting, dangerous feel. Figure 4 shows a small trace with collisions.

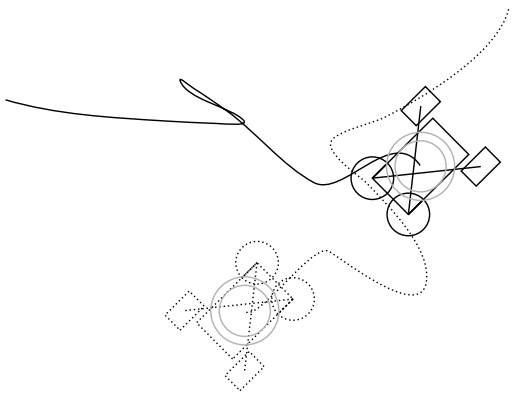

Figure 4: Two Vehicles 2b with headlights. Since Vehicle 2b is a light seeker, these head towards one another and collide. There have been two collisions; they show up as sharp changes in direction in the trails. These vehicles will continue to collide indefinitely.

5 Other objects

Xbraitenberg supports lights as well as vehicles. Lights currently come in three varieties: fixed, timed, and threshold. A fixed light always generates a fixed amount of radiation. A threshold light turns on only when some amount of light is already in the vicinity. A timed light begins as on, but gradually shuts off over a short period of time. (For example, this might act as an initial stimulus to excite any threshold lights.) Any of these can be attached to vehicles as headlights.

6 Vehicle or world?

This section contains a preliminary investigation of how vehicle behavior can change when the physics of the world changes, as opposed to the wiring of the vehicle itself. The first example we choose is from the Braitenberg vehicle problem set—specifically, the problem of making a vehicle go back and forth between two lights.

Figure 5 shows one back-and-forth vehicle at work. The vehicle's sensors are angularly limited—they can only see in front of them. Its motor-response curves are shown in Figure 6. The effect of the two curves is as follows: When total sensor input is low, the vehicle heads for the nearest light. However, when it gets reasonably close to the light, the vehicle shuts off. It drifts onward, slowing down because of friction, until it comes close enough to the light that the right motor switches back on; at that point, the vehicle quickly flips 180 degrees, away from the nearest light. It then, hopefully, is facing another light, and will head slowly for that.

This solution pretty much works; if the vehicle starts between two lights, it will go back and forth as desired. However, its behavior is quite fragile in terms of the world's friction constants. Figure 5 is shown with the default translation friction constants ($k_s = k_d = 0.2$) and second angular friction constant ($\kappa_2 = 1$), but a lower-than-usual first angular friction constant ($\kappa_1 = 0.1$, not 0.3). The somewhat complicated solution will only work when angular responsiveness is high, so that the vehicle can execute the 180-degree

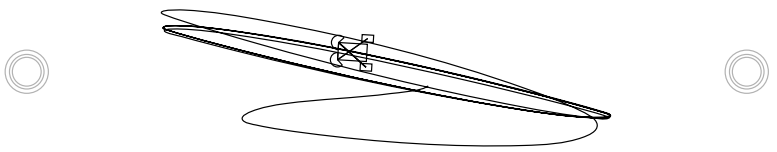

Figure 5: The back-and-forth vehicle at work.

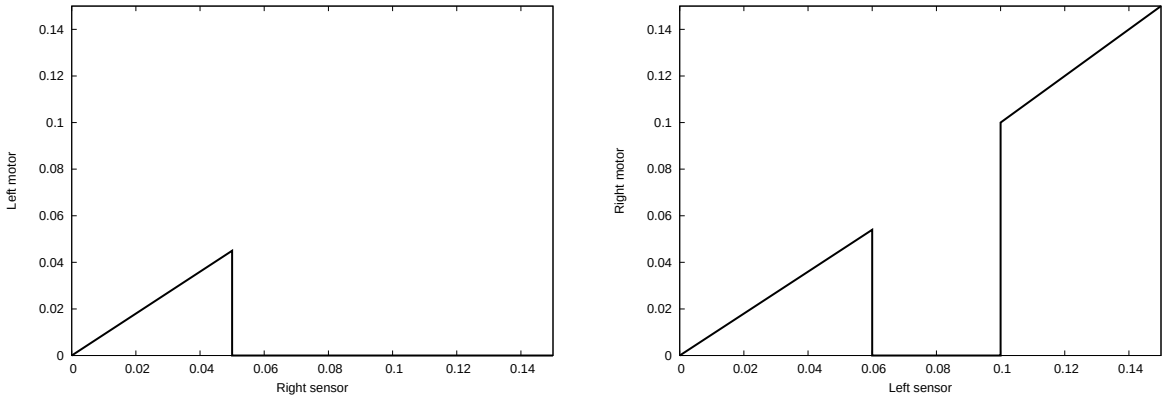

Figure 6: Left and right motor logic for the back-and-forth vehicle.

flip quickly.

What if we raise the angular friction? Figure 7 shows the same vehicle in a world with normal angular friction (0.3), and Figure 8 shows it in a world with ten times higher angular friction. As angular friction increases, the vehicle grows slack and cannot turn in time to execute its planned behavior. With angular friction lower than 0.1, the problem is different. If placed exactly between the lights, the vehicle still goes back and forth between them. However, if placed near a light, the vehicle is much more unstable now—the turning action can whip the vehicle around so fast that it overshoots 180 degrees; then the left sensor can see the light again, which causes another turning action. The result is that the vehicle wanders off while constantly spinning, until it gets out of range. The trace in Figure 9 demonstrates this. You can't really see the spinning action, but you can see how the vehicle circumnavigates the light sources. This is qualitatively different behavior than the same vehicle with angular friction 0.1: when that vehicle gets pointed away from the light sources, angular friction is hard enough to overcome that the vehicle tends to just stop.

Collisions also affect how the observer perceives the Braitenberg vehicles' intentions. In a version of the simulator without collisions, placing Vehicle 2a and Vehicle 2b together, each with a headlight, exhibits a predator-prey relationship. Vehicle 2b runs after Vehicle 2a, catches up to it, and runs it over; Vehicle 2b then passes onwards while Vehicle 2a acts broken (it slows down and changes direction). When collisions are enabled, however, the relationship is different. Vehicle 2b still runs for Vehicle 2a, but instead of "running it over", it runs into its rear. The elastic collision then gives Vehicle 2a some forward momentum, so it goes faster; but Vehicle 2b will catch up with it again and repeat the process. The two vehicles proceed onwards in an accelerating straight line. It's as if Vehicle 2b is goading Vehicle 2a into some action that Vehicle 2a doesn't really want to take—call it Macbeth and Lady Macbeth.

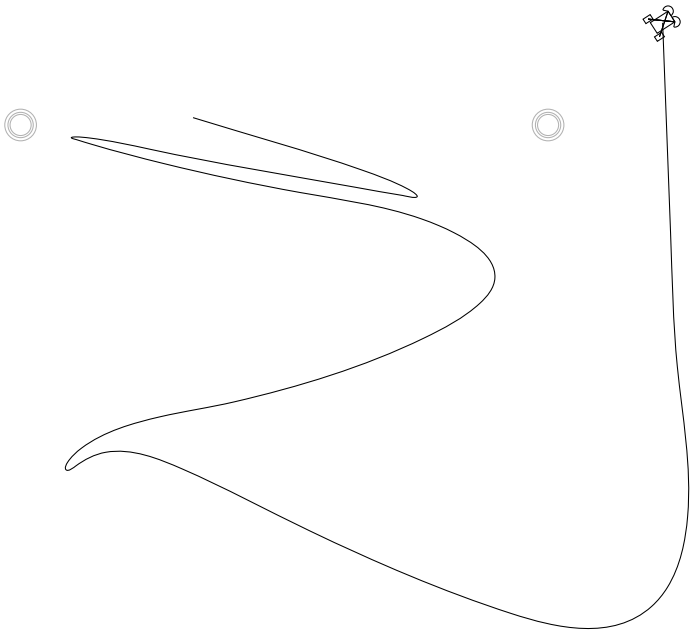

Figure 7: The back-and-forth vehicle with angular friction $= 0.3$.

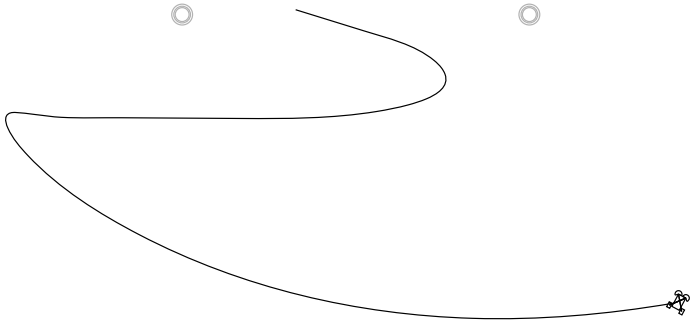

Figure 8: The back-and-forth vehicle with angular friction $= 1$.

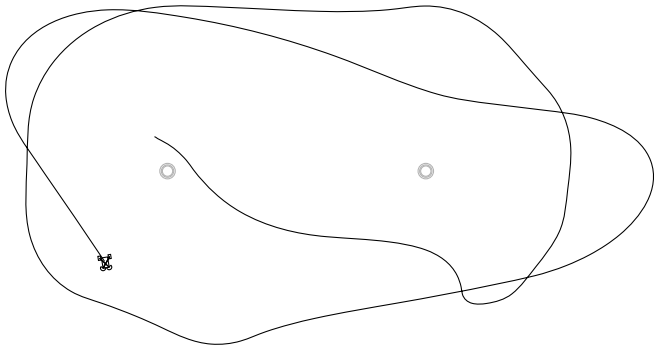

Figure 9: The back-and-forth vehicle with angular friction = .01.

7 Conclusion

Braitenberg vehicles are not pure thought experiments; they, too, are embedded in an environment that affects their behavior. A simple vehicle that has specialized its behavior for one environment—say, the back-and-forth vehicle, which was specialized for low angular friction—can have completely different behavior in another environment. A Braitenberg vehicle simulator that models physics, even partially, offers a richer and stranger world than a simple, physicsless model. It's a world that's fun to play with.

Xbraitenberg is available on the Web at the following address:

`http://www.lcdf.org/~eddietwo/xbraitenberg/`

A Vehicle motion calculation

We assume the vehicle's center of mass is at $(0, 0)$ and that the vehicle is moving along the positive x-axis (so its velocity equals $(v, 0)$ for some positive v). Some constants, such as the "gravitational constant", are left out for simplicity.

Constants for the world

k_s	coefficient of static friction
k_d	coefficient of dynamic friction
κ_1	first coefficient of angular friction
κ_2	second coefficient of angular friction

Constants for the vehicle as a whole

m	the vehicle's mass
r	the vehicle's radius
I	the vehicle's moment of inertia $= mr^2/2$

Constants for each motor

$\mathbf{p}_i$	position of motor i
(r_i, θ_i)	position of motor i in polar coordinates
ρ_i	rotation of motor i

The current state

$\mathbf{P}$	the old position $= (0, 0)$
Θ	the old rotation $= 0$
$\mathbf{v}$	the vehicle's velocity $= (v, 0)$
$\hat{\mathbf{v}}$	the velocity normal $= (1, 0)$
ω	the vehicle's angular velocity
$\hat{\omega}$	the "normal angular velocity" $=$ the sign of ω, either 1 or -1
f_i	the amount motor i is firing
Δt	the length of a time quantum

Calculated quantities

$\mathbf{P}'$	the new position (the old position equals $(0, 0)$)
Θ'	the new rotation (the old rotation equals 0)
$\mathbf{v}'$	the new velocity
ω'	the new angular velocity
$\mathbf{F}_m$	force on the vehicle due to motors
$\mathbf{F}_f$	total force on the vehicle ($\mathbf{F}_m$ plus friction)
T_m	torque on the vehicle due to motors

1. **Calculate the new position and rotation:**

$$\mathbf{P}' = \mathbf{P} + \mathbf{v}\Delta t$$
$$\Theta' = \Theta + \omega\Delta t$$

2. **Determine the force and torque due to motors:**

$$\mathbf{F}_m = \sum_i (f_i \cos\rho_i, f_i \sin\rho_i)$$
$$T_m = \sum_i -f_i r_i \sin(\theta_i - \rho_i)$$

3. **Adjust the force for friction:** If $\mathbf{v} = \mathbf{0}$ and $|\mathbf{F}_m| < k_s m$, then $\mathbf{F}_f = \mathbf{0}$; otherwise, $\mathbf{F}_f = \mathbf{F}_m - k_d m \hat{\mathbf{v}}$.

4. **Adjust the torque for angular friction:** If $\omega = 0$ and $|T_m| < \kappa_1 I$, then $T_f = 0$; otherwise, $T_f = T_m - \kappa_1 I \hat{\omega} - \kappa_2 \omega$.

5. **Accelerate the vehicle:**

$$\mathbf{v}' = \mathbf{v} + \mathbf{F}_f \Delta t / m$$
$$\omega' = \omega + T_f \Delta t / I$$

References

[1] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1984.

[2] Robin Edwards. Braitenberg vehicle simulator (one vehicle, C), 1999. `http://www.methedrine.demon.co.uk/vehicles.html`.

[3] Chris Gerken. Braitenberg vehicle simulator (many vehicles, Java), 1998. `http://www.mindspring.com/~gerken/vehicles/`.

[4] Rosalba Perna. Personal communication, February 1999.

[5] Chris Thornton. Braitenberg vehicle simulator (many vehicles, POPBUGS), 1999. `http://www.cogs.susx.ac.uk/users/christ/popbugs/braitenbergs.html`.

[6] Torsten Will. Braitenberg vehicle simulator (many vehicles, Java), 1999. `http://www.geocities.com/Colosseum/3141/Braitenberg.html`.

[7] John Wiseman. Braitenberg vehicle simulator (many vehicles, Lisp), 1999. `http://www.cs.uchicago.edu/~wiseman/vehicles/`.

[8] Mark Ziler. Braitenberg vehicle simulator (one vehicle, Macromedia Shockwave), May 1999. `http://www.duke.edu/~mrz/braitenberg/`.