Vehicles Manual

The Braitenberg Vehicle Simulator – Vehicles - is a tool for exploring the nature of intelligent behaviour. In itself, Vehicles is an elegant 3d graphical display tool integrated with a 2.5d physics engine and a sophisticated neural network processor. Vehicles is configured using an easy-to-learn vehicle language that enables the user to create and modify vehicles and specify the world in which the vehicles live.

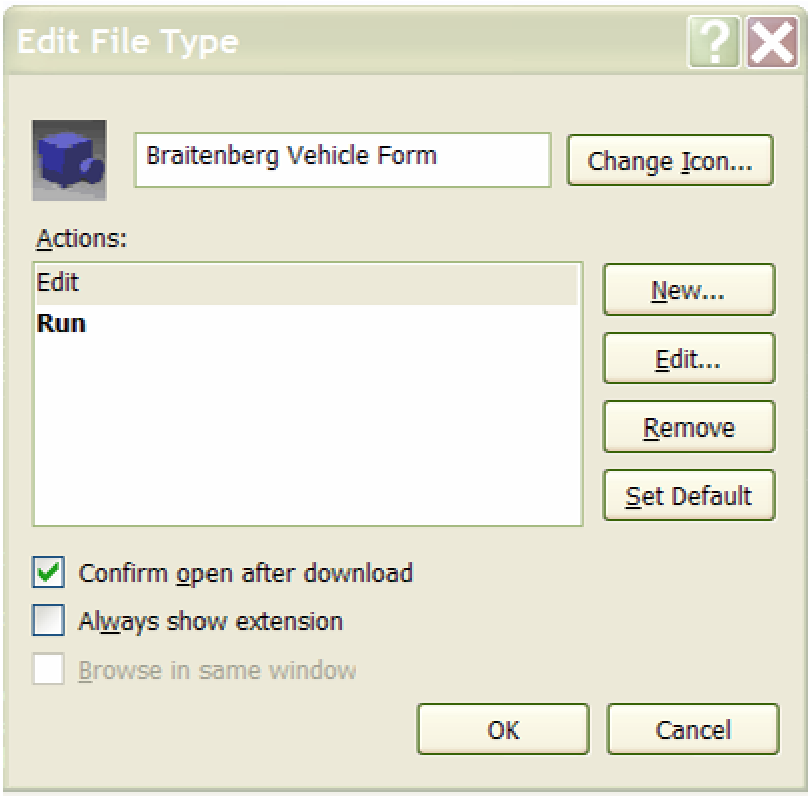
Installation

Vehicles currently runs on Windows platforms using Open-GL for the graphics engine. Cross-platform versions are planned. The instructions below are for a typical installation in a Windows environment.

The Vehicles folder contains the vehicles.exe, bv.ico and glut32.dll files. This folder would typically be placed under Program Files, but may be kept elsewhere if convenient. The Experiments folder contains the sample .bvf files that are used in the book as configuration files for each experiment. The Experiments folder should be placed in a folder in the user's work area, such as My Documents, as you may wish to edit these files or use them as the basis for your own experiments.

Under Windows, you can create a simple development environment by associating the .bvf extension with the Vehicles program, and also a text editor such as Notepad. Text editors with line numbering, auto indenting and brace matching (as found in programmer's editors) are recommended. The example below shows how this would done for a typical Windows installation:

1. Open Tools / Folder Options in My Documents or My Computer.
2. On the File Types tab, click New.
3. Type `BVF` in the File Extension entry box, and then click OK.
4. Select the `BVF` extension in the list of Registered file types, and then click Advanced.
5. You should now be viewing the Edit File Type dialog box. Type Braitenberg Vehicle File in the entry box at the top of the form, and the click Change Icon … .
6. Browse to where you installed your Vehicles folder and click on bv.ico.
7. Select the icon, and click OK.
8. You should now have returned to the Edit File Type dialog box. Click New… .
9. Type Run in the Action entry box. Click on Browse… .
10. Browse to where you installed your Vehicles folder and double-click on vehicles.exe.
11. Click on OK in the New Action dialog box. You should now have returned to the Edit File Type dialog box, and there should be a Run entry in the action list. Click New… .
12. Type Edit in the Action entry box. Click on Browse… .
13. Browse to your favourite text editor, or C:\Windows\Notepad.exe, and double click on the program.
14. Click on OK in the New Action dialog box. You should now have returned to the Edit File Type dialog box, which should now look like the figure below.
15. Click on OK as required to complete the operation.


Edit File Type

Braitenberg Vehicle Form

Actions:

Edit

Run

[✓] Confirm open after download

[ ] Always show extension

[ ] Browse in same window

Starting the Program

If you have associated the .bvf files with the program then the easiest way to run the program is to double click on the .bvf file you wish to execute. Alternatively, the Vehicles program may be started from the command line with the name of the configuration file (with the optional extension):

```
vehicles [filename][.bvf]
```

If no filename is supplied, then you will be prompted to enter a filename.

Loading the .BVF file

When the program starts it creates the features in the .BVF file based on the instructions within. If there is an error in the syntax of the file, the loader will stop and print a description of the error. You will be asked if you wish to continue with the load. If you continue, the loader will use default values for data that could not be read which may lead to undesirable results.

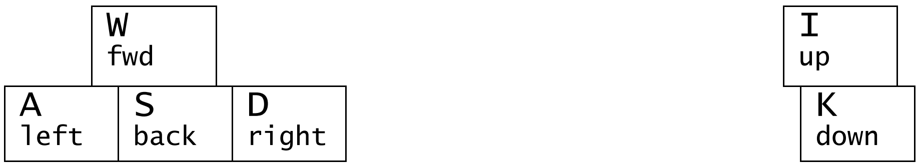
Note that the line number reported will be near the error in the file, but the error may have occurred a few lines earlier.

Interacting with the Simulator

While the simulator is running you may move about in the simulated environment, move vehicles around, inspect vehicle's internal data and control the rate of simulation.

Camera Movement

The camera may be translated (moved without rotation) using the following keys:



The camera may rotated by clicking on the left mouse button and dragging the point of view to the desired location. Click on a point away from a vehicle, or you may inadvertently move the vehicle.

Vehicle Inspection and Movement

To inspect the activity within a vehicle, click on that vehicle. The left hand side of the screen displays the internal values of the vehicle using the component names defined in the configuration file. The text will update as the vehicle moves about. To clear the text press the T key.

To move a vehicle, click on it and drag the vehicle. The vehicle will move in the direction of the drag.

Controlling the Simulation

To exit the simulation, hit the ESC or Q key.

To pause and un-pause the simulation hit the P key.

To step through the simulation and the minimum simulation rate, hit the spacebar. The simulation will move one simulation step each time the spacebar is pressed. To resume normal operation hit the P key.

Reference

Simulations are described by writing a Braitenberg Vehicle File (BVF). The BVF describes:

- the simulation parameters and configuration
- the field that the vehicles live upon
- the light sources in the field, and
- the sensors, brains and motors of the vehicles themselves.

These components are arranged in a nested fashion in the BVF, so that multiple instances of a single entity can be created from a single description of the entity. For an overview of the nesting levels and possible entries at each level see the Quick Reference section.

The BVF always begins with the keyword **simulation** which defines the top level of the BVF hierarchy. The simulation is described by a number of entries contained between two curly braces { … }.

There are two entry types: simple and complex. Simple entry types have an element identifier and a value. The type of the value is determined by the element identifier. Complex entries have many sub-entries, again contained between curly braces { … }. The sub-entries in turn may be further simple and complex entry types.

The following subsections describe each level and entry type in the BVF.

Simulation

Element	**Type**	**Default**
min_render_step	int (milliseconds)	50
min_sim_step	int (milliseconds)	1
real_time	bool	“true”
auto_render	bool	“true”
paused	bool	“false”
name	string	“Vehicle Simulator”
win_size	int [2] (x, y)	full screen
camera	float [5] (x, y, z, pitch, yaw)	0.0 -2.0 4.0 20.0 60.0
field	field	{…}

This is the top level of the BVF. It describes the global settings for the simulation using the following parameters.

min_render_step : This integer value is the smallest amount of simulated time that will elapse between updates of the Open-GL rendering of the simulation. If the real_time parameter is false, the simulation will always render with this step size. If real time operation is requested by setting the real_time parameter to true, then the amount of elapsed time between renderings may be increased to keep elapsed simulation time in synchronicity with real time.

min_sim_step : This integer value is the smallest amount of simulated time that will elapse between updates of the simulation physics and vehicle controllers. If the real_time parameter is false, the simulation will always run with this step size. If real time operation is requested by setting the real_time parameter to true, then the amount of elapsed time between steps may be increased to keep elapsed simulation time in synchronicity with real time.

real_time : If this Boolean parameter is true, the simulation will try to match simulation and real time. If the parameter is false, the simulation will run at the specified step sizes which may result in faster or slower than real time operation.

auto_render : If this parameter is false the rendering will only be updated when the simulation is paused or stepped, otherwise the simulation is updated based on the minimum render step size.

paused : If this parameter is set to "true", the simulation will begin in the paused state. The simulation may be stepped at the minimum simulation rate by pressing the spacebar, or un-paused using the P key. The simulation may be paused or un-paused at any time using the P key regardless of the state of this variable; the variable only controls the pause state at the start of the simulation.

name : This is the string that is displayed in the window's title bar.

win_size : This sets the initial window size for the simulation in pixels. Windows can be re-sized at any time by dragging the corners.

camera : This sets the initial position and orientation of the camera used for rendering. The field is in the (x,y) plane with the height of the camera specified in z. Yaw specifies rotation around the z axis, and pitch is about an axis perpendicular to the viewing direction and parallel to the field plane.

field : Field is a complex element that has many sub-elements as specified in the next section. There may be only one field specified in the simulation.

Field – Sub-element of Simulation

Element	Type	Default
size	float (metres)	100.0
wrap	bool	"true"
Brownian	float (Newtons)	0.0
light*	light	NULL
vehicle*	vehicle	NULL

size : Length of the edge of the field in meters.

wrap : A Boolean describing whether vehicles will wrap back on to the opposite edge when they reach the edge of the field, or simply disappear.

Brownian : A random force and torque applied to each vehicle. The forces have a mean of 0 Newtons and a standard deviation of the size specified in this element. A value of 0.0 specifies no Brownian forces are applied to the vehicle. Brownian torques are normalised to the moment of the vehicle.

light : This a complex element. A field may have an unlimited number of light types, but only eight light sources may be positioned on the field.

vehicle : This is a complex element. A field may have an unlimited number of vehicles.

Light – Sub-element of Field

Element	**Type**	**Default**
colour	float [3] (R, G, B)	1.0 1.0 1.0
radius	float (metres)	0.5
brightness	float (GL units)	5.0
point	bool	“true”
attenuation	attenuation	“square”
place_at*	light_pose	{…}

attenuation = {none, linear, square}

colour : specifies the colour of the light type in red, green, blue components.

radius : specifies the radius of this type of light globe as it is rendered.

brightness : specifies the brightness of the light type in Open-GL units.

point : specifies whether the light acts a point source (like a light bulb) or a parallel source (like a light at an infinite distance).

attenutation : specifies the rate of decay of brightness over distance from the light source. “none” specifies no decay, “linear” specifies a linear decrease in brightness, “square” specifies a square law for computing the brightness which is closest to the true behaviour of light sources.

place_at : is a complex element of type light_pose that specifies the position of the light. This complex element may be repeated to place multiple lights of the same type at different positions. The simulation can have at most eight light sources.

Light_pose – Sub-element of Light

Element	**Type**	**Default**
position	float [3] (x, y, z) (m)	0.0 0.0 2.0

position : specifies the position of the light source in field coordinates. The origin of the field is in it’s centre.

Vehicle – Sub-element of Field

Element	**Type**	**Default**
size	float (metres)	1.0
colour	float [3] (R, G, B)	0.5 0.5 1.0
mass	float (kg)	1.0
viscous_friction	float (Ns/m)	0.1
Coulomb_friction	float (N)	0.0
sensors*	sensor	{…}
brains*	brain	{…}
drives*	drive	{…}
place_at*	vehicle_pose	{…}

size : specifies the edge length of the cube that forms of the body of the vehicle type. Used in rendering and moment calculations.

colour : specifies the RGB colour of the vehicle type used for rendering and for sensing by other vehicles.

mass : specifies the mass in kilograms of the vehicle type for dynamic calculations.

viscous_friction : specifies the resistance to motion experienced by each drive unit in Newtons per metre per second.

Coulomb_friction : specifies a constant friction force the resists the direction of motion of each drive.

sensors : complex elements specifying the sensors. Each vehicle may have an unlimited number of sensors.

brains : complex elements specifying the brains. Each vehicle may have an unlimited number of brains.

drives : complex elements specifying the drives. Each vehicle may have an unlimited number of drives.

place_at : is a complex element of type vehicle_pose that specifies the position of the vehicles. This complex element may be repeated to place multiple vehicles of the same type at different positions. The simulation can have an unlimited number of vehicles.

Sensor – Sub-element of Vehicle

Element	**Type**	**Default**
name	string	“sensors”
mode	sensor_type	“light”
colour	float [3] (R, G, B)	1.0 1.0 1.0
width	float (degrees)	0.0
place_at*	sensor_pose	{…}

sensor_type = {light, vehicle}

name : string identifying the sensor type. This name is used by the brain element to determine connectivity.

mode : specifies the type of entity that this sensor detects. “light” senses the brightness of light, “vehicle” senses the distance to another vehicle.

colour : specifies the colour of light or vehicle sensed. The colour must be an exact match to sense.

width : specifies the angular width of the sensing region. The width is the planar angle in degrees in the field plane.

place_at : is a complex element of type sensor_pose that specifies the position of the sensors. This complex element may be repeated to place multiple sensors of the same type at different positions. The simulation can have an unlimited number of sensors.

Sensor_pose – Sub-element of Sensor

Element	**Type**	**Default**
position	float [3] (x, y, z) (m)	0.0 0.0 0.0
angle	float (degrees)	0.0

position : specifies the position of the sensor in robot coordinates. The origin of the robot is in the centre of the body cube.

angle : refers to the angle in the field plane that the sensor points. The angle is with respect to the front-facing direction of the vehicle.

Brain – Sub-element of Vehicle

Element	**Type**	**Default**
name	string	“brain”
num_outputs	int	1
bias	float*	0.0
unit_type	unit_fn	“linear”
time_constant	float (seconds)	0.0
weights*	weights	{…}

unit_fn = {linear, sigmoid, tanh, threshold, Gaussian}

name : string identifying the brain type. This name is used by other brain elements and drive elements to determine connectivity, and for debug displays.

num_outputs : determines the number of outputs and hence units in the brain. Each output comes from a single unit. Units are arranged as a row vector.

bias : specifies the base activity of a unit before any influence from the weights. This row vector specifies the activity for each respective unit.

unit : specifies the transfer function of every unit in the brain type. “linear”computes the summed activity of the weighted inputs. “sigmoid” applies a sigmoidal transfer function to the linear output. “tanh” applies a hyperbolic tan transfer function to the output. “threshold” outputs 1 if the linear activity is positive, else 0. “Gaussian” applies a Gaussian transfer function to the linear output.

time_constant : adds a delay to the output of the unit. If the unit is a threshold unit, then the time_constant is a pure delay. Otherwise the output is computed by:

$$\frac{dy}{dt} = (r - y)\frac{t}{\tau}$$

where y is the output of the unit, r is the weighted sum of inputs with any transfer function applied, t is simulation time and τ is the specified time constant.

weights : is a complex element that represents the connections into the brain units. There may be any number of weight units, each typically used to connect the units to different sensor banks or to different brain units. A brain element may connect to itself.

Weights – Sub-element of Brain

Element	**Type**	**Default**
input	String	“sensors”
weights	float*	0.0, …

input : is the sensor bank or set of brain units that acts as input to the brain. Each element in the named sensor bank or set of brain units is connected by weights to each unit in the brain element. That is, the inputs are fully connected to the units in the brain element. Brain elements may connect to themselves.

weights : are the values of the connections between the designated inputs and the units. Each connection has a floating point value to define its strength. The number of weights must be the product of the number of units in the input by the number of units in the brain element.

Drive – Sub-element of Vehicle

Element	**Type**	**Default**
name	string	“drive”
input	string	“brain”
gain	float (Newtons)	1.0
lateral_grip	bool	“true”
place_at*	drive_pose	{…}

name : string identifying the drive type. Used in debug displays.

input : is the sensor bank or set of brain units that acts as input to the drive. Each element in the named sensor bank or set of brain units is connected one-to-one to the drive units. There must be an equal number of inputs to drive units. The number of drive units is specified by the number of *place_at* sub-elements.

gain : gain applied to each one-to-one connection to transform unit activity to force in Newtons.

lateral_grip : is a Boolean that specifies whether the vehicle is allowed to move in any direction other than the drive angle. If this element is set to false, multiple drives can be used to create swimming or floating, or even omni-drive type behaviour, whereas a setting of true makes the vehicle behave more like it has wheels.

place_at : is a complex element of type drive_pose that specifies the position of the drives. This complex element may be repeated to place multiple drives of the same type at different positions. The simulation can have an unlimited number of drives.

Drive_pose – Sub-element of Drive

Element	**Type**	**Default**
position	float [3] (x, y, z) (m)	0.0 0.0 0.0
angle	float (degrees)	0.0

position : specifies the position of the drive in robot coordinates. Drives always contact the field surface. The origin of the robot is in the centre of the body cube.

angle : refers to the angle in the field plane that the drive points. The angle is with respect to the front-facing direction of the vehicle.

Vehicle_pose – Sub-element of Vehicle

Element	**Type**	**Default**
position	float [3] (x, y) (m)	0.0 0.0
angle	float (degrees)	0.0

position : specifies the position of the vehicle in field coordinates. Vehicles always contact the field surface. The origin of the field is in the centre.

angle : refers to the angle in the field plane that the vehicle points.

Braitenberg Vehicle File - Quick Reference

simulation

Element	Type	Default
min_render_step	int (milliseconds)	50
min_sim_step	int (milliseconds)	1
real_time	bool	“true”
auto_render	bool	“true”
paused	bool	“false”
name	string	“Vehicle Simulator”
win_size	int [2] (x, y)	full screen
camera	float [5] (x, y, z, pitch, yaw)	0.0 -2.0 4.0 20.0 60.0
field	field	{…}

field

Element	Type	Default
size	float (metres)	100.0
wrap	bool	“true”
Brownian	float (Newtons)	0.0
light*	light	NULL
vehicle*	vehicle	NULL

light

Element	Type	Default
colour	float [3] (R, G, B)	1.0 1.0 1.0
radius	float (metres)	0.5
brightness	float (GL units)	5.0
point	bool	“true”
attenuation	attenuation	“square”
place_at*	light_pose	{…}

attenuation = {none, linear, square}

light_pose

Element	Type	Default
position	float [3] (x, y, z) (m)	0.0 0.0 2.0

vehicle

Element	Type	Default
size	float (metres)	1.0
colour	float [3] (R, G, B)	0.5 0.5 1.0
mass	float (kg)	1.0
viscous_friction	float (Ns/m)	0.1
Coulomb_friction	float (N)	0.0
sensors*	sensor	{…}
brains*	brain	{…}
drives*	drive	{…}
place_at*	vehicle_pose	{…}

sensor

Element	Type	Default
name	string	“sensors”
mode	sensor_type	“light”
colour	float [3] (R, G, B)	1.0 1.0 1.0
width	float (degrees)	0.0
place_at*	sensor_pose	{…}

sensor_type = {light, vehicle}

sensor_pose

Element	Type	Default
position	float [3] (x, y, z) (m)	0.0 0.0 0.0
angle	float (degrees)	0.0

brain

Element	Type	Default
name	string	“brain”
num_outputs	int	1
bias	float*	0.0
unit_type	unit_fn	“linear”
time_constant	float (seconds)	0.0
weights*	weights	{…}

unit_fn = {linear, sigmoid, tanh, threshold, Gaussian}

weights

Element	Type	Default
input	string	“sensors”
weights	float*	0.0, …

drive

Element	Type	Default
name	string	“drive”
input	string	“brain”
gain	float (Newtons)	1.0
lateral_grip	bool	“true”
place_at*	drive_pose	{…}

drive_pose

Element	Type	Default
position	float [2] (x, y) (m)	0.0, 0.0
angle	float (degrees)	0.0

vehicle_pose

Element	Type	Default
position	float [2] (x, y) (m)	0.0 0.0
angle	float (degrees)	0.0