Smart, P. R. (2020) Planet Braitenberg: Experiments in Virtual Psychology. *Cognitive Systems Research*, 64: 73–95. [DOI: 10.1016/j.cogsys.2020.06.001]

Planet Braitenberg: Experiments in Virtual Psychology

Paul R. Smart[a,*]

[a] *Instituto de Filosofia da Nova, Faculdade de Ciências Sociais e Humanas, Universidade Nova de Lisboa, Campus de Campolide - Colégio Almada Negreiros, 1099-032 Lisbon, Portugal.*

Abstract

Braitenberg vehicles are simple robotic platforms, equipped with rudimentary sensor and motor components. Such vehicles have typically featured as part of thought experiments that are intended to show how complex behaviours are apt to emerge from the interaction of inner control mechanisms with aspects of bodily structure and features of the wider (extra-agential) environment. The present paper describes a framework for creating Braitenberg-like vehicles, which is built on top of a widely used and freely available game engine, namely, the Unity game engine. The framework can be used to study the behaviour of virtual vehicles within a multiplicity of virtual environments. All aspects of the vehicle's design, as well as the wider virtual environment in which the vehicle is situated, can be modified during the design phase, as well as at runtime. The result is a general-purpose simulation capability that is intended to provide the foundation for studies in so-called *computational situated cognition*—a field of study whose primary objective is to support the computational modelling of cognitive processes associated with the physically-embodied, environmentally-embedded, and materially-extended mind.

Keywords: Virtual Environment, Virtual Robotics, Artificial Intelligence, Game Engine, Computational Simulation, Artificial Life, Situated Cognition, Embodied Cognition, Unity

1. Introduction

The work of the Italian neuroscientist and cyberneticist, Valentino Braitenberg, marks an important milestone in the study of intelligent behaviour, especially when it comes to our understanding of the behaviour of environmentally-situated agents. The simple robotic platforms described by Braitenberg (1984) in his wonderfully engaging book, *Vehicles: Experiments in Synthetic Psychology*, form the basis of thought experiments in which the interplay between simple internal control systems and suitably positioned sensors and actuators leads to

*Corresponding author
Email address: ps02v@ecs.soton.ac.uk (Paul R. Smart)

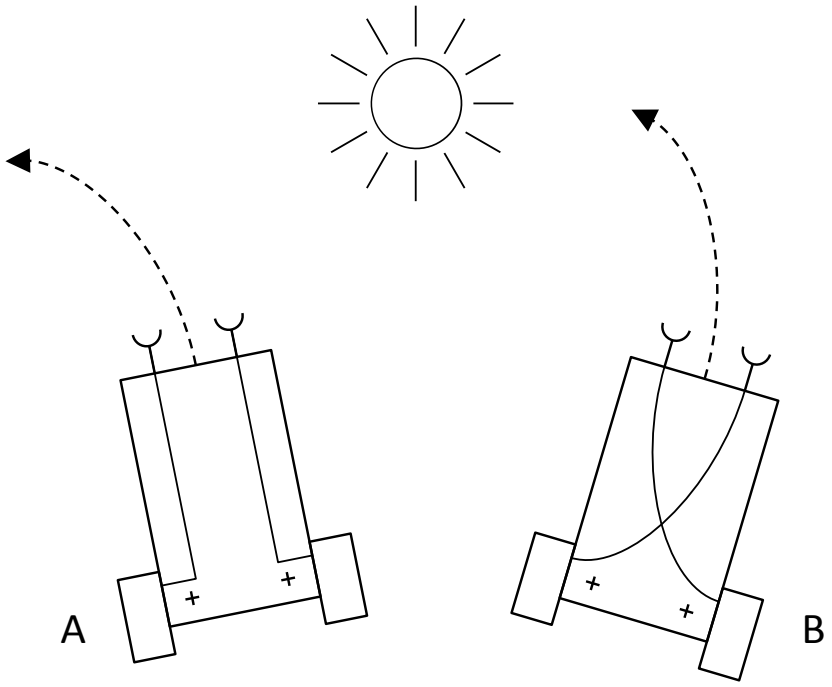


Figure 1: The trajectories of two types of Braitenberg vehicle. Vehicle A moves away from a light source, while vehicle B moves towards a light source. (Relative to Braitenberg's (1984) taxonomy, these vehicles correspond to vehicles 2A and 2B, respectively.)

the emergence of a rich array of behavioural responses. Crucially, Braitenberg suggests that the complexity of such behavioural responses invites certain kinds of intentional characterization, ones that seem to warrant the ascription of intentional mental states. Commenting on the behaviour of the two vehicles illustrated in Figure 1, for example, Braitenberg (1984, p. 9) suggests that we will tend to see Vehicle A as a 'coward', constantly trying to avoid illuminated areas. Vehicle B, by contrast, is 'aggressive'; Vehicle B will always turn towards a light source and continue approaching it until it collides with the light, perhaps with considerable force.

Braitenberg advocates an approach to the study of behaviour that is both situated and synthetic. It is synthetic in the sense that Braitenberg attempts to understand how psychologically-interesting patterns of behaviour arise from mechanisms that are designed from the bottom up. This contrasts with the typical approach adopted in psychology, where the goal is to analyse an existing system with a view to unravelling the mechanisms responsible for some episode of intelligent behaviour. The distinction between these two approaches is nicely captured in the following quote by Mirolli and Parisi (2011):

> Traditionally, science has been trying to understand reality through systematic observation and experiment. This can be called the 'analytic approach' to science. But since the advent of the computer in the late 1940s a new kind of approach to science has appeared, one which tries to understand reality by modeling it in computers or robots. This is what can be called the 'synthetic approach' to science. The rationale for the synthetic approach is that, once you have built a system that reproduces some phenomenon of reality, you have a candidate explanation of that phenomenon in that it is possible—even though by no means certain—that the principles you have used to build your artificial system are the same principles that underlie the real phenomenon and explain it. (Mirolli & Parisi, 2011, p. 298–299)

A characteristic feature of the synthetic approach to psychology—or synthetic psychology—is thus to understand the mechanisms that are sufficient to generate or give rise to patterns of behaviour that we deem to be intelligent (or, at any rate, psychologically interesting).[1] Although Braitenberg's own work relies on the use of thought experiments, it should be clear that this synthetic approach is perfectly compatible with the use of computer simulation techniques and the design of physical robots.

In addition to advocating a synthetic approach, Braitenberg also adopts a situated approach to intelligent behaviour. In this respect, Braitenberg anticipates the flavour of recent work in cognitive science, especially that which appeals to the causal or constitutive relevance of extra-neural elements in the genesis of intelligent behaviour. Much of the behavioural complexity of Braitenberg's vehicles is thus tied to the way in which some inner control mechanism interacts with aspects of a vehicle's bodily design (e.g., the placement and orientation of sensory transducers) as well as features of the wider (extra-agential or extra-vehicular) environment. Such insights resonate with the views of many of those who hail from the ranks of situated and embodied cognitive science (Clark, 1997; Pfeifer & Bongard, 2007).

Braitenberg's work has been the inspiration for a number of important strands of cognitive scientific research (Dewdney, 1987; French & Cañamero, 2005; Hogg et al., 1991; Lilienthal & Duckett, 2004; Rañó, 2014; Salomon, 1999; Thornton, 2017). For the most part, this work has centred on the use of either computational simulation techniques or the construction of physical robots. While both of these approaches are consistent with the overall spirit of Braitenberg's vision, I would like to suggest that neither approach has yielded much in the way of a general-purpose framework that supports experiments in synthetic (and situated) psychology. On the one hand, computer simulation experiments have

[1] A similar approach is sometimes adopted in the social sciences. In this case, computer simulations are used to explore the (social) mechanisms that are sufficient to give rise to (social) phenomena. Work in this area is sometimes referred to as *generative social science* (e.g., Epstein, 2006).

typically been performed with 2D representations of both the vehicle and the environment (e.g., Thornton, 2017). These studies have yielded important and compelling results, but they often require the modeller to make simplifying assumptions about the nature of sensorimotor exchanges with the environment or the kind of physical constraints (e.g. friction, gravity, and momentum) that influence the shape of overt behaviour. At the other end of the spectrum are studies that rely on the use of real-world physical robots (e.g., Hogg et al., 1991; Lilienthal & Duckett, 2004). These studies clearly obviate many of the concerns associated with computer simulation techniques; however, they risk introducing a range of further problems. Not everyone is capable of building physical robots, for example, and even if they are, it often takes considerable time and effort to construct (and reconstruct) vehicles in order to explore the effect of (e.g.) different bodily designs or sensor functionalities. In addition, the environments in which real-world vehicles are situated obviously need to be designed and configured by hand, and not every kind of environment is able to be implemented in the way we want. Problems arise if we want to simulate the effect of unusual or inaccessible environments, such as a deep ocean (or benthic) habitat or (worse still) an extra-terrestrial environment with alien gravity. This looks to be of potential importance, inasmuch as we seek to apply Braitenberg's approach to the design of off-world robots, especially if such robots are to benefit from many of the lessons emanating from the realms of situated and embodied cognitive science.

The ultimate realization of Braitenberg's vision, I suggest, is an approach that combines a flexible approach to the design of both vehicles and environments, while simultaneously preserving at least some of the constraints and opportunities associated with real-world environments (e.g., the effect of physical forces, such as friction). Other desiderata of potential relevance include those relating to usability, data analysis, and community engagement (e.g., the ability to share simulation assets, such as vehicle morphologies, control systems, and custom environments). There is, moreover, a concern with the ludic-like elements of Braitenberg's vision—ideally, the process of designing and building vehicles should be as much an exercise in productive play as it is serious science.

The present paper describes a framework that is intended to speak to these desiderata. In particular, it outlines an approach to Braitenberg-style synthetic psychology that is rooted in the use of 3D virtual environments—an approach that is most commonly associated with work in artificial life (e.g., Terzopoulos et al., 1997). The paper describes a general-purpose simulation framework—codenamed the Planet Braitenberg Framework (PBF)—which is implemented on top of the Unity game engine.[2] The PBF consists of a set of vehicle designs (implemented as simple 3D models), sample environments, and a set of modular code components that can be used to control vehicle behaviour.[3] Once a simulation is configured,

[2]See `http://unity3d.com/`.

[3]The source code for the PBF can be downloaded from the GitHub website: `https://github.com/ps02v/planet-braitenberg-framework`.

the game engine can be run, and the real-time behavioural responses of vehicles can be visualized within a perceptually-rich 3D environment that exploits the capabilities of the game engine with respect to both photo-realistic graphics rendering and real-time physics simulation.

The primary aim of the present paper is to describe the architecture and functionality of the PBF. To this end, Section 2 provides an overview of the general approach, with a particular emphasis on the motivation for choosing the Unity game engine. Section 3 focuses on the design of virtual vehicles within the PBF, describing some of the sensor, effector, and behavioural control systems that are available at the time of writing. Section 4 presents the results of simulations performed using the PBF. Some of these simulations resemble the thought experiments described by Braitenberg; others seek to showcase the capabilities of the PBF by focusing on novel behaviours. Section 5 outlines the support for recording (and replaying) simulations. This section also discusses some of the performance issues that may confront simulation efforts, as well as the strategies that may be used to resolve such issues. Finally, Section 6 summarizes some of the future uses of the PBF. The ultimate aim of the PBF is to provide a platform for what I will call *computational situated cognition*. This is an area of research that seeks to use computer simulations to further our understanding of issues arising in four areas of cognitive scientific research, namely, embodied (Shapiro, 2011, 2014), extended (Clark, 2008; Clark & Chalmers, 1998; Menary, 2010a), enactive (Froese & Ziemke, 2009), and embedded (Rupert, 2004) cognition. These are sometimes referred to as the four E's of cognitive science research, or more simply the study of 4E cognition (Menary, 2010b). The common feature of research in these areas is a commitment to the role of extra-neural (and sometimes extra-agential) resources in shaping the structure of intelligent behaviour. In general, proponents of 4E cognition emphasize the way in which cognitive success is achieved as the result of forces and factors that are distributed across the brain, the body, and the world (Clark, 1997). Such claims are broadly consistent with the flavour of recent work in robotics research (e.g., Pfeifer & Bongard, 2007), and they certainly echo the spirit of Braitenberg's initial vision. Although it is not the primary aim of the present paper to motivate or justify the use of virtual environments for 4E research, it is worth noting that the PBF is intended as a stepping stone in this general direction.

2. Planet Braitenberg

The PBF is an open-source collection of game assets that can be used for the purposes of building simulated 3D environments containing virtual Braitenberg-like vehicles. The 'game' assets contained within the framework are many and varied. They include 3D models of Braitenberg vehicles, data capture and visualization components, state machines, virtual sensors, editor utilities, behaviour control modules, and example simulations. Together, these components are intended to support the development of computer simulations involving simple robotic agents of the sort envisaged by Braitenberg (1984). Beyond this, however, the example simulations included with the PBF (some

of which are described below) showcase solutions to a number of problems that confront the effort to develop robust computational simulations of embodied cognitive agents. In this sense, the PBF could easily serve as a point of departure for artificial life simulations or work in virtual cognitive robotics.

The PBF is built on top of the Unity game engine, which can be used to create both 2D and 3D virtual environments. Although Unity is mostly used for the purposes of video game development, it has also been used in a number of other application contexts. These include the development of simulation capabilities and visualization environments for use in educational (Christel et al., 2012), medical (Rizzo et al., 2014), and engineering (Mattingly et al., 2012) applications.

The Unity game engine forms part of what is sometimes referred to as the Unity game creation system. This includes, in addition to the game engine, an Integrated Development Environment (IDE) (see Figure 2) and an object-oriented scripting framework based on the Microsoft C# language. The scripting framework enables developers to create custom code components that derive from a common class, called `MonoBehaviour`, and this provides access to over-ridable methods that are invoked at different stages of a game's execution, e.g., during the game engine's update loop. Scripts that contain classes derived from `MonoBehaviour` can be attached to game objects in order to control the behaviour of those objects at runtime. They can also be used to take actions in response to user input, implement custom user interfaces, store and load game data, and implement general game mechanics. Many of the core code components included with the PBF are implemented as subclasses of the `MonoBehaviour` class. These include components that represent the sensors (see Section 3.2), effectors (see Section 3.3), and behavioural control systems (see Section 3.4) of virtual Braitenberg-like vehicles.

In addition to code components that derive from `MonoBehaviour`, the PBF includes components that extend the existing functionality of the Unity IDE. Such extensions are typically used to facilitate the game design process, but they can be used equally well to support the process of designing simulations. The most common type of editor extension included with the PBF comes in the form of so-called *Custom Inspectors*. These extend the functionality of Unity's Inspector window (see Figure 2), providing support for the configuration of vehicle properties.

At this point, it should be clear that the PBF is not intended to be used in a stand-alone fashion. Instead, it is intended to be used as part of the Unity game creation system, with the Unity IDE itself serving as the environment in which specific simulations are created, run, and evaluated. There are a number of advantages to using Unity as the backdrop for research into computational situated cognition. These advantages include the following:

- **Technical Support:** Multiple forms of support are available to the Unity user base. These include extensive documentation, samples, and video tutorials.
- **Cost:** The latest version of the game engine (and IDE) are free to use

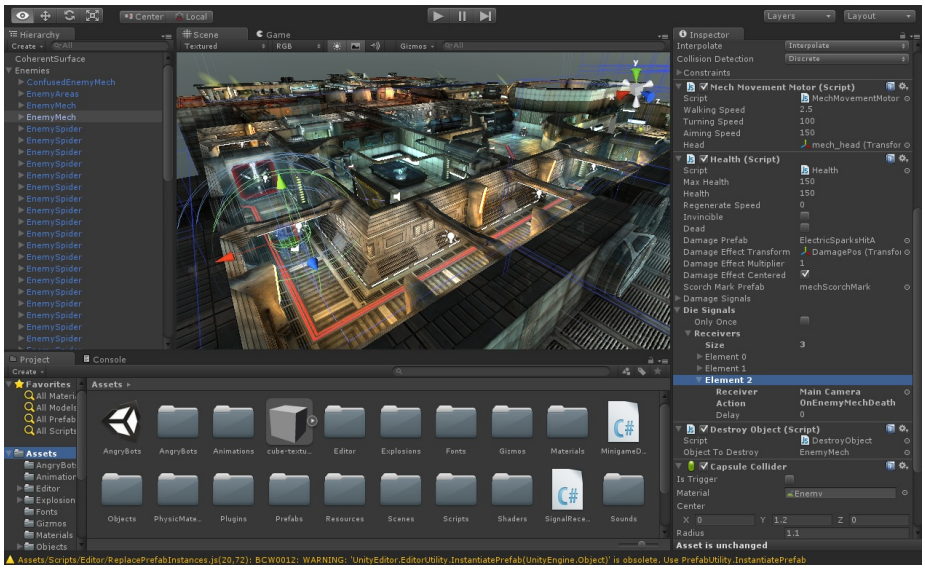


Figure 2: View of the Unity IDE showing the Scene, Hierarchy, Project, and Inspector windows. These enable a developer to visualize the current scene (Scene window); adjust the properties of game/simulation objects (Inspector window); manage game/simulation assets (Project window), such as texture, audio, code, and model assets; view objects that are instantiated in a given scene (Hierarchy window); and 'play test' the current state of the game/simulation (Game window). The IDE also integrates seamlessly with external code editors, such as MonoDevelop and Microsoft Visual Studio.

for research and development purposes. The free version includes all the functionality required to support the PBF simulations described below.

- **Multiplatform Support:** The Unity game creation system is available for Linux, MacOS, and Windows platforms.

- **Deployment:** The simulations developed with Unity, and thus the PBF, can be compiled into executable files and deployed to multiple platforms. These include the World Wide Web (via WebGL), game consoles (e.g., Xbox, Wii, and PS3), mobile devices (e.g., iPhone, iPad, and Android devices), and personal computers (e.g., Windows, MacOS, and Linux).

- **Community & Sharing:** The Unity Asset Store[4] provides access to a rich array of assets (e.g., models, terrains, textures, sounds, etc.) that can be used to create simulations. In addition, simulations developed with Unity (and the PBF) can be easily shared with other interested parties by exporting the simulations as Unity Asset Packages.[5]

[4]See `https://assetstore.unity.com/`.
[5]See `https://docs.unity3d.com/Manual/AssetPackages.html`.

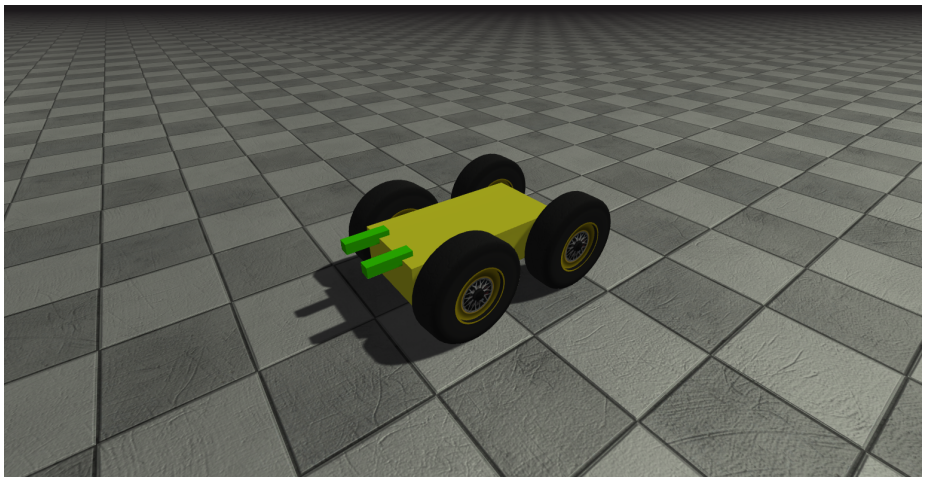


Figure 3: A basic virtual vehicle.

These features make Unity a compelling target for simulation environments that are intended to support cognitive systems research.

3. Virtual Vehicles

3.1. Bodies

Each virtual vehicle has a simple body plan consisting of a main chassis, to which are attached two eye stalks and four wheels (see Figure 3). By default, two game cameras are attached to the eye stalks and function as the vehicle's eyes. These cameras (implemented using the Unity `Camera`[6] component) render the 3D scene from the vehicle's perspective. A range of additional sensors are available to support the processing of information in other modalities (see Section 3.2).

A `TrailRenderer` component is positioned at the rear of the vehicle. This generates a persistent trail behind the vehicle as it moves around the environment (e.g., Figure 9). Such visual traces are useful for tracking vehicle movement in different situations and under different experimental conditions.

A `Rigidbody` component is attached to the top-level game object in the hierarchy of objects that make up the vehicle's body. This component enables the vehicle to participate in the physics calculations made by the Unity physics engine. Essentially, the addition of a `Rigidbody` component to the vehicle enables a designer to specify some of the 'physical' properties of the vehicle, such as its mass and drag. It also enables the vehicle to respond to physical forces and engage in physical interactions with other objects in the environment. A set of

[6]All code components (including methods) are highlighted using a `teletype` font. More information about these components can be found on the Unity website (`https://docs.unity3d.com/ScriptReference/index.html`) or the documentation that accompanies the PBF.

`Collider` components are attached to each element of the vehicle's body in order to define the collision geometry of the vehicle. This geometry establishes the 'physical presence' of the vehicle within the virtual environment, i.e., it defines the points at which collisions with other physical objects in the environment occur. The wheels of the vehicle are implemented using a set of special colliders, called `Wheel Colliders`. These colliders are specifically designed for wheeled vehicles within game environments. They feature built-in collision detection, wheel physics, and a slip-based tire friction model.

3.2. Sensors

A number of sensors can be added to virtual vehicles in order to support behavioural responses to different features of the environment. In general, the purpose of sensors is to extract information from the virtual environment in which a vehicle is situated. Each sensor is implemented as a Unity `MonoBehaviour` component that is independent of other modelling components. This means that sensors can be added in a modular fashion to yield vehicles with different 'perceptual' capabilities.

Relative to the simulations described below, two kinds of sensor are of particular importance: eyes and radial sonars. The functionality of these sensors is described below. Other sensors are available to support olfaction, taction, and magnetoception. In addition, vehicles have access to various forms of proprioceptive information. This includes information about the directional orientation of the vehicle's eyes, as well as information about the vehicle's locomotor system (e.g., wheel rotation rate). (At present, there are no sensors that yield anything resembling what—in a biological context—might be regarded as interoceptive information.)

Each vehicle is equipped with two eyes that are capable of imaging the virtual scene throughout the course of a simulation. Each eye is implemented using a `Camera` component and a collection of custom code components, the most important of which is the `Eye` component. While `Camera` components typically render their view of the scene to the main game window during game play, they can also be used to render a scene to a special type of 2D image asset, called a `RenderTexture`. This `RenderTexture` asset can be processed using standard 2D graphic processing routines, such as those built into the Unity game engine, or those provided by Microsoft's .NET Framework. Using the methods of the Unity game engine, it is possible to retrieve information about the red, green, and blue (RGB) values of each pixel in the `RenderTexture`. This provides the raw data for further visual processing. For example, it is possible to compute average luminance levels across the three RGB colour channels in order to get an estimate of the average light intensity recorded by each eye camera (see Section 4.1). It is also possible to split the image into its component colour channels as a means of accessing information about the chromatic content of a scene (see Section 4.2).

In the context of the PBF, the `RenderTexture` asset generated by a vehicle's eye is referred to as the vehicle's 'retina'. By default, the retina is 200×200 pixels, although this can be changed to alter the resolution of a vehicle's visual

system. Each eye can be configured as either a 'simple' eye or a 'complex' eye. These two configurations differ with respect to the way the visual scene—as represented at the retinal level—is processed. In the case of a simple eye, the entire retina is processed to yield global image statistics, such as average or maximum brightness levels. Exactly the same image statistics are provided in the case of a complex eye, with the exception that the retina is divided into a number of retinal patches and summary statistics are provided for each patch. By default, the retina of a complex eye is divided into 5 vertically-oriented patches (or columns), which (in the case of a default retina) are 40×200 pixels in size. As we will see, this sort of configuration is crucial to the capacity of virtual vehicles to exhibit certain kinds of motion sensitivity (see Section 4.3).

By default, the eye cameras point directly ahead (i.e., zero degrees rotation with respect to Unity's y or 'up' axis). This can, however, be modified during the design phase, or at runtime. The ability to rotate a vehicle's eyes during the course of a simulation (i.e., at runtime) provides the basis for additional forms of visuo-motor control beyond those used to support changes in locomotor behaviour. In particular, future work in this area could explore the use of oculomotor responses in the context of active vision simulations, where eye movements are made in response to the presence of particular features within the visual field. This, in turn, could provide the basis for visual tracking capabilities, similar to those described by Terzopolous et al. (1997). Additional areas of research include the use of more complex forms of visual processing, such as those used in computer vision research (Nixon & Aguado, 2012), as well as the use of predictive coding techniques, similar to those described for biological retinas (Hosoya et al., 2005; Srinivasan et al., 1982). It is important to note that Unity provides access to a number of image post-processing techniques that can be used as part of retinal processing. These include edge detection algorithms of the sort that are commonly used in machine vision applications.

In addition to eyes, a second sensor system plays an important role in the simulations described below. This is the radial sonar sensor. The radial sonar is a specialized sensor system that provides information about the distance and relative location of 'physical' objects in the vehicle's environment. The radial sonar is intended to echo the functionality of real-world sonar systems that one sometimes encounters in cognitive robotics research (e.g., Mataric, 1991).

At runtime, the radial sonar emits a number of rays that are projected radially from a central point. The orientation and length of these rays can be controlled by the user; for most simulations, however, the default configuration of 12 rays and a projection/detection distance (i.e., length) of 10 meters is likely to be sufficient. During the course of a simulation, each ray detects the presence of physical objects in the environment. It does so by using the Unity `Physics.RayCast` function, which projects rays into the scene with the same length and orientation as the rays specified by the radial sonar. If the ray projected by a `Physics.RayCast` intersects with a collider in the scene, it records this information and makes it available to other components. In most cases, the informational output of the radial sonar is used by a vehicle behaviour module (see Section 3.4) for the purpose of coordinating behavioural output (see

Section 4.5).

3.3. Effectors

The behaviour of virtual vehicles can be controlled by a number of effector systems, the most important of which are the wheel motor system, the steering system, and the oculomotor system. These systems control the torque that is applied to the vehicle's wheels, the steering angle of the vehicle's front and/or rear wheels, and the rotational orientation of the vehicle's eyes in the x (vertical) and y (horizontal) planes. Given the centrality of these motor systems to most simulations, they are implemented as part of the `Vehicle` component. There are, as such, no specialized components corresponding to (e.g.) the locomotor system within the PBF. This marks a point of departure from the approach taken with respect to the implementation of vehicle sensors, which, recall, were implemented using a highly modular design pattern. The absence of specialized motor components in the PBF reflects the central importance of at least certain kinds of motor output (e.g., locomotor responses) to what makes something a virtual vehicle (at least a vehicle of the Braitenbergian variety). In the absence of any sort of behavioural output, a vehicle is no longer an *active* entity that warrants any form of intentional characterization. It is at best, perhaps, a sensor platform, or merely an object in the (virtual) world.

There are, of course, occasions when the addition of more specialized motor systems is warranted. This is likely to be the case when new bodily components are added to a vehicle. In previous work, for example, we have reported the results of simulations with vehicles that attempt to retrieve illuminated spheres from an enclosed arena (Smart & Sycara, 2015b). The vehicles in these simulations were equipped with a proboscis that was under the control of a specialized motor control system.

In terms of a vehicle's locomotor behaviour, three parameters are of crucial importance. These are motor torque, steering speed, and steering angle. The motor torque parameter specifies the amount of torque that is applied to the wheels of the vehicle. In most cases, the torque is applied to the rear wheels and is positive. This results in forward movement of the vehicle. The torque can also be inverted to reverse the rotation of the wheels and thus reverse the movement of the vehicle. The steering angle parameter specifies the orientation of the vehicle's front or rear wheels. It can be used to steer the vehicle towards or away from particular targets. The rate at which a vehicle turns its wheels to align them with the steering angle is determined by the steering speed parameter. This parameter specifies the rate (in degrees per second) at which the vehicle will orient its wheels to the target steering angle. Rapid steering rates tend to yield more responsive behaviour; however, they can also produce erratic behaviour and, occasionally, maladaptive precipitant responses.[7]

[7] A sharp left turn at high speed, for example, may result in a luminescent object disappearing from the vehicle's field of view, thereby causing the vehicle to lose 'sight' of an otherwise attractive visual target.

The vehicle's actual movement through the environment relies on the physics system, which is built into the Unity game engine. This is important, for the use of the physics system means that the behaviour of a virtual vehicle is, to some extent, constrained by the sort of physical forces that constrain the behaviour of real-world robotic vehicles. In attempting to initiate forward movement, for example, a vehicle needs to apply sufficient torque to overcome friction; just because motor torque has a non-zero value, this does not mean that a vehicle will always move. Similarly, a moving vehicle will not immediately stop just because its motor torque is reduced to zero; as with physical objects in the real world, virtual vehicles are subject to inertia.

Of course, as noted in Section 1, one of the virtues of virtual worlds is that it is possible to alter the physical forces that operate on virtual vehicles. This provides the basis for simulations that compare vehicle performance in different 'physical' environments. For example, one could simulate a Martian or Lunar environment by (among other things) changing the gravity setting of the game engine's physics system (this is set to Earth gravity by default).

3.4. Behavioural Control

In building a virtual vehicle within the PBF, we begin with a hierarchy of game objects that define the basic geometry and 'physical' presence of the vehicle within a virtual world (see Section 3.1). To that we add a `Vehicle` component to provide our vehicle with some basic motoric capabilities (see Section 3.3). Next, we add one or more sensor systems to allow our vehicle to process information about its virtual ecological niche. Finally, we need to add a component that brings all this together—a component that unites the functionality exhibited by all the other components. Within the PBF, such components are known as *vehicle behaviour components* or *vehicle behaviour modules*. They play a key role in sensorimotor coordination, helping to determine the value of motoric parameters in response to certain patterns of sensory input. Vehicle behaviour modules are, in this sense, a key component (although only one component) of the computationally-realized causal nexus that shapes, influences, and constrains the behaviour of virtual vehicles.

Figure 4 shows the relationship of vehicle behaviour components to other elements of the PBF. As can be seen from Figure 4, `Vehicle` components rely on vehicle behaviour components to support their functionality (i.e., to generate motor output). Similarly, vehicle behaviour components require a (single) `Vehicle` component in order for their own functionality (i.e., behavioural control) to be expressed.

Figure 4 also shows some of the vehicle behaviour components that are currently implemented within the PBF. These include components to support a variety of behaviours, some of which are described below (see Section 4). As is perhaps clear from the names assigned to vehicle behaviour components (see Figure 4), vehicle behaviour components are responsible for different kinds of behaviour. In this sense, they are the primary means of distinguishing between different types (or species?) of vehicles. For the most part, a simulation involving virtual vehicles will draw on a common set of sensor components, vehicle body

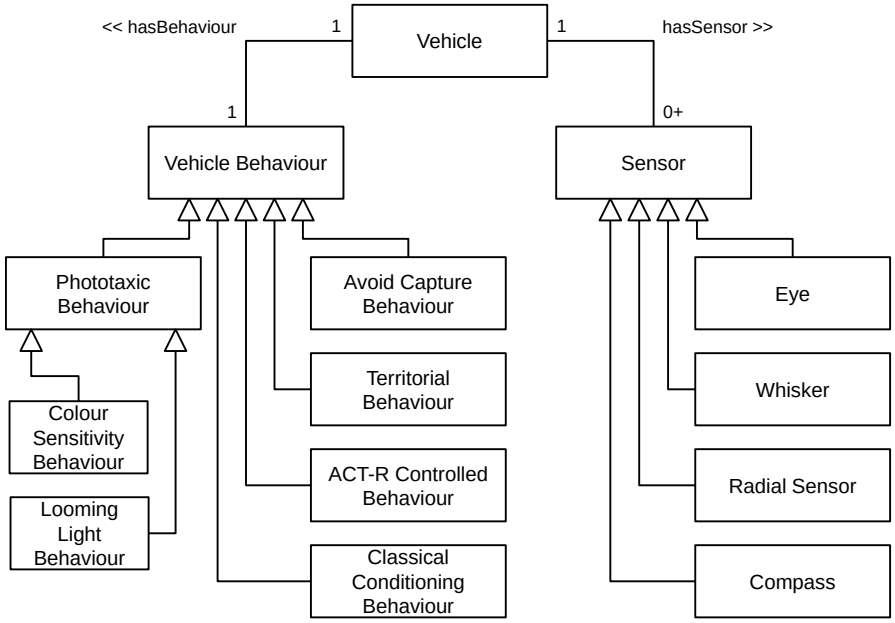


Figure 4: A UML class diagram showing some of the key classes of the PBF. Virtual vehicles are created using the `Vehicle` class, which is associated with at least one `VehicleBehaviour` module and any number of `Sensor` components.

designs, and a mandatory `Vehicle` component. What differs between simulations is the kind of vehicle behaviour component that is assigned to a virtual vehicle. This does not mean, of course, that the effort to create new simulations does not, at times, involve the creation of new sensor systems, vehicle designs, or specialized motor components. Nevertheless, in most cases, much of the creative effort will be directed to the development of new vehicle behaviour components.

Programmatically, vehicle behaviours are implemented as Unity `MonoBehaviour` components that inherit from a custom abstract base class, called `VehicleBehaviourBase`. All concrete subclasses of `VehicleBehaviourBase` are required to override a method, called `Execute()`, which is invoked every time the motor output of a vehicle needs to be recalculated. In practice, the `Execute()` method is called by the `Vehicle` component on a periodic basis. The `Vehicle` component includes a user-configurable property, named `BehaviourUpdateInterval`, which specifies the time interval (in seconds) between successive invocations of the `Execute()` method. Given that most of the computational routines controlling a vehicle's behaviour are performed in the context of the `Execute()` method, the `BehaviourUpdateInterval` can be used to manage the computational overhead associated with a particular simulation. For most simulations, a value of 0.25 or 0.5 (corresponding to an update frequency of 4Hz or 2Hz, respectively) is sufficient, although simulations requiring faster rates of sensory processing may utilize values as low as 0.1 (i.e.,

10Hz). Values lower than 0.1 are not recommended for performance reasons, especially for simulations that require complex forms of visual processing (see Section 5).

The goal of the `Execute()` method is to determine the setting of key motor parameters (e.g., motor torque, eye rotation, and steering angle), which are then used by the `Vehicle` component to implement changes in the vehicle's overt behaviour (e.g., by applying motor torque to the vehicle's wheel colliders). These changes are implemented as part of Unity's fixed update loop, which (by default) runs at a rate of 50Hz.

4. Intentional Agents

The design of virtual vehicles supports their use in a wide variety of simulation contexts. Vehicles can be equipped with different sensors, and the properties of these sensors can be modified to assess behavioural responses under different environmental conditions. In addition, subtle differences in behaviour can emerge by changing the response properties of the motor system. Modifying the rate at which a vehicle is able to orient its wheels to a specified steering angle can make all the difference between a vehicle successfully homing in on a visual target and missing it altogether. Often, of course, a designer will start out with a specific behavioural capability in mind and then attempt to generate that behaviour by implementing custom control systems. This can be a highly creative activity, requiring considerable ingenuity. The emphasis on producing coherent behaviour from a limited number of sensor inputs is demanding but tractable, and the ability to immediately test new vehicle designs by visualizing vehicle behaviour within a virtual 3D scene provides feedback that is both instructive and rewarding.

In order to demonstrate vehicle behaviour, the results of a number of simulations are presented in the current section. These exemplify the use of many of the vehicle behaviour modules presented in Figure 4. Due to space limitations, the description of these simulations will be necessarily brief.

4.1. Love, Lust, and Loathing

The Braitenberg vehicles illustrated in Figure 1 highlight two kinds of phototactic response to light stimuli: approach and avoidance. Braitenberg (1984, p. 9) characterizes the behaviours of these vehicles in different ways, suggesting that Vehicle A would be regarded as a 'coward', while Vehicle B would be deemed to be 'aggressive'. Braitenberg (1984) goes on to identify additional kinds of phototactic response, in each case proposing that we, as human observers, would be inclined to interpret the behaviour in different ways. By including thresholds in a vehicle's positive (photophilic) responses to light stimuli, for example, Braitenberg suggests that a vehicle will come to rest in the vicinity of a light source. Such a response profile, he suggests, may be viewed as a form of 'love'.

In the context of the PBF, these kinds of phototactic response can be implemented via a single component: the `PhototaxicBehaviour` component

(see Figure 4). This component exposes a single property, called `ResponseType`, that can be set to one of two values: `Photophilia` or `Photophobia`. This setting determines the overall response of a vehicle to light stimuli, with a value of `Photophilia` eliciting positive (approach) responses to light stimuli and a value of `Photophobia` eliciting negative (avoidance) responses.

From a programmatic perspective, the operation of the `PhototaxicBehaviour` component is relatively simple. As with all `VehicleBehaviour` components, it includes an `Execute()` routine that is invoked by a `Vehicle` component on the same game object. (This routine, recall, is invoked on a periodic basis, according to the value assigned to the `Vehicle.BehaviourUpdateInterval` property.) When this routine is called, it processes the retinal information associated with each of the vehicle's retinas, computing the average luminosity (i.e., brightness value) of pixels across the entire retinal surface. It is this value that is ultimately used to inform the vehicle's motor responses, determining the level of torque applied to the vehicle's motor and the steering angle of (in this case) the vehicle's front wheels. It is at this point that we encounter the main difference between vehicles of the photophilic and photophobic variety: photophilic vehicles always adjust the steering angle in the direction of the brightest light source (i.e., they make an ipsilateral steering response relative to the retina reporting the greatest average illumination); photophobic vehicles, by contrast, always steer away from the brightest light source (i.e., they make a contralateral steering response relative to the retina reporting the greatest average illumination).

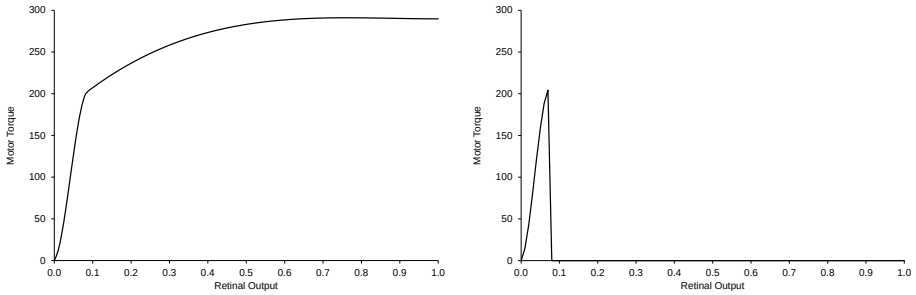
It is important to note that much of the subtlety of a vehicle's responses to light is determined by what are called *transfer functions*. At a general level, these are functions that yield some output value for a particular input value. In the context of the PBF, transfer functions are implemented as animation curves, which, as their name suggests, are typically used for animation purposes as part of game development. For the most part, animation curves are used for the purposes of keyframe animation, with keyframes existing as user-defined points on the curve and the overall shape of the curve being used for interpolation purposes. Within Unity, animation curves are depicted as two-dimensional graphs, with time represented on the x-axis. Crucially, Unity provides a graphical editor for animation curves. This editor, available via the Animation window, enables a user to create custom animation curves that specify the behaviour of game objects (or, more precisely, the value of game object properties) throughout the course of a game. A linear animation 'curve', for example, might be applied to the transform of a game object in order to move the game object a fixed amount for each second of the game. More complex animation curves (e.g., a sine-wave animation curve) will obviously yield different motion dynamics.

The PBF relies on animation curves to control much of the complexity of a vehicle's sensorimotor processing. The reason for this is twofold: firstly, animation curves can be created (and modified) using a familiar (and highly flexible) part of the Unity editor; secondly, animation curves provide a graphical alternative to mathematically-defined functions, which would otherwise need to be specified programmatically. Animation curves thus reduce the need for programming and obviate the need for mathematical understanding, while simultaneously

exploiting the existing expertise of the Unity user community.

As a concrete example of the way in which animation curves are used in the PBF, let us consider a specific kind of transfer function, namely, the visuo-motor transfer function. This is a transfer function that (as with many transfer functions) is implemented at the level of the `Vehicle` component. The visuo-motor transfer function is, in essence, a two-dimensional graph with a Cartesian coordinate system. The abscissa (or x-axis), in this case, is a normalized value that represents the output of some form of retinal processing. For present purposes, let us assume that the value represents the average brightness of pixels across the retinal surface.[8] A value of zero will thus indicate zero illumination (i.e., total darkness), while a value of one will indicate maximal illumination. This value can now be evaluated with respect to the visuo-motor transfer function. The value is entered as the abscissa, on the x-axis, and the result is the corresponding value on the y-axis, i.e., the ordinate. In the case of the visuo-motor transfer function, this output value represents the level of motor torque that will be applied to the vehicle's wheels. For a simple linear animation curve defined by two keyframes—(0,0) and (1, 300)—the level of motor torque will increase in a linear fashion from a minimum value of zero (no illumination) to a value of 300 (maximum illumination). Changing the value of the keyframes by dragging them to different positions on the curve editor interface will define a different curve. For the purposes of illustration, suppose we redefine the linear animation curve so that it is represented by the points (0,300) and (1,0). Here, the level of motor torque will change (again in a linear fashion) from a value of 300 (no illumination) to a value of zero (maximum illumination). This reversal in motor torque dynamics, relative to the photic properties of the ambient environment, will yield very different kinds of behavioural response. A vehicle that is maximally activated by light with one kind of visuo-motor transfer function may remain completely motionless with another, and this is despite the fact that all that has changed between the two situations is the input/output profile of a single transfer function. Different visuo-motor transfer functions can thus be used to create vehicles with very different response profiles. Given that it is these response profiles that underwrite intentional characterizations of vehicle behaviour (e.g., whether we see the vehicle as exhibiting 'cowardice' or 'aggression', or as being a 'lover' or a 'fighter'), it is possible to create many different kinds of vehicles—each of which is defined with respect to psychologically-significant patterns of behaviour—without resorting to code. Equally important is the fact that the animation curves defining specific transfer functions can be saved as separate assets (technically, the curves are saved as instances of the `SerializedObject` class) and applied to vehicles in the same or different scenes. This provides a simple means by which transfer functions can be shared amongst the members

[8]For a binocular vehicle, the input to the visuo-motor transfer function is typically the average value of information derived from the two retinas (e.g., average pixel luminosity). Ultimately, however, it is up to the human designer to specify what information should be passed to the transfer function, based on the results of retinal processing.



(a) Lust (b) Love

Figure 5: Motor torque transfer functions for (a) 'lust' and (b) 'love' behaviour.

of a user community.[9]

Let us now compare the effect of two kinds of visuo-motor transfer function that were used to implement two kinds of vehicle, which I will dub 'LustBot' and 'LoveBot'. Both of these vehicles come equipped with the aforementioned `PhototaxicBehaviour` component, and they are, as such, vehicles that have an in-built predisposition to respond to light sources within their local environments. Furthermore, both vehicles have precisely the same configuration and they are embedded in precisely the same environment. The only difference between the vehicles is the nature of their visuo-motor transfer functions, which are illustrated in Figure 5. Figure 5a depicts the visuo-motor transfer function for the LustBot. As can be seen from this figure, motor torque increases in response to increasing illumination, and this is particularly evident at low illumination levels (i.e., from zero to 0.1). Above a value of 0.1, the increase in motor torque begins to tail off, reaching a value of 300 at the maximal level of illumination. The visuo-motor transfer function associated with the LoveBot has a very different profile. Here, we again see a rapid increase in motor torque at low illumination levels; but, in contrast to LustBot, motor torque is eliminated at higher levels of illumination.

Let us now see how these two visuo-motor transfer functions affect the respective behavioural responses of the LustBot and LoveBot vehicles. Figure 6 shows the path taken by the LustBot as it navigates its way to a light source. As shown in the figure, the vehicle takes approximately 20 seconds to reach the target from a central start position. The labelled circles within this figure indicate the location of the vehicle at different points in the simulation, and the vehicle's view of the scene, as rendered by its eye cameras, is depicted for each of these time points in Figure 7. Perhaps unsurprisingly, given the nature of the corresponding visuo-motor transfer function (see Figure 5a), the LustBot exhibits a response profile characterized by ever-greater levels of motor output

[9]This is in addition to the usual route by which game project assets are shared using (e.g.) the Unity Asset Store.

(and thus speed) as it approaches the light. This response persists even when the vehicle comes into contact with the light source and its forward motion is halted by the 'physical' presence of the light. As can be seen from the video that accompanies this simulation,[10] the LustBot exhibits a non-attenuating response to the light. Braitenberg (1984, p. 9) characterized this particular kind of behaviour (i.e., Braitenberg Vehicle 2A) as consistent with aggressive tendencies. Personally, I prefer a somewhat more libidinous characterization, based on the results obtained with certain simulations. In particular, if the light is positioned at a certain height, the LustBot will approach the light with such force as to effectively 'mount' the light and come to rest on top of it. The 'aggressive' characterization looks to be more apt in situations where the light is deactivated (i.e., switched off) in response to a sufficiently robust collision (see the video for the aggravation simulation below[11]). In this case, the behaviour of the vehicle looks to have a particular purpose, i.e., to deactivate or destroy the light, and this (at least from the author's standpoint) is reminiscent of irritation, aggravation, or aggression.[12]

The sole difference between LustBot and LoveBot concerns the distinctive shape of their respective visuo-motor transfer functions. This difference accounts for the somewhat more 'refined' response of LoveBot to light stimuli. At low light levels, LoveBot will, like LustBot, move in the direction of the greatest light intensity. Above a certain threshold, however, LoveBot's motor torque drops to zero, and it should thus cease to exhibit locomotory movement. In actual fact, things are somewhat more complicated than this, for LoveBot's motion is subject to all the usual forces that are applied to `RigidBody` objects. LoveBot will thus not come to an immediate halt simply because its motor torque is zero; instead, it will continue to move forward until (e.g.) frictional forces bring it to a complete standstill.[13] In the present simulations, vehicles are configured to have both a mass and drag; LoveBot will thus come to rest once it has detected light levels above a certain intensity *and* the physics engine calculates that friction is sufficient to overcome inertial forces. The distance of LoveBot and LustBot

[10] See `https://youtu.be/8K3liDgYnA8`.

[11] See `https://youtu.be/FmZFqv8aw1Q`.

[12] As an aside, it is worth noting the way in which intentional characterizations of what is in effect the same behaviour are influenced by the 'responses' of external objects, i.e., the objects that the vehicle interacts with. In this case, the folk psychological characterization of the agent appears to supervene on properties that are not wholly contained within (or even associated with) the physical borders of an individual agent. Such a perspective resonates with the views of those advocating an active externalist approach to mind and cognition (Clark, 2008; Clark & Chalmers, 1998). They also speak to the idea that folk psychological characterizations of behaviour (i.e., belief and desire talk) are to be understood as something of a "holistic net thrown across a body of the behaviour of an embodied being acting in the world" (Clark, 1989, p. 5).

[13] It should also be noted that LoveBot's behaviour is affected by ambient light levels. If ambient light is of sufficient intensity as to yield a motor torque of zero, then LoveBot's motor will not be activated, and it will remain motionless until either the ambient light intensity is reduced (at dusk perhaps) or it is affected by other physical forces (e.g., collisions with other vehicles) that affect its exposure to light sources within the scene.

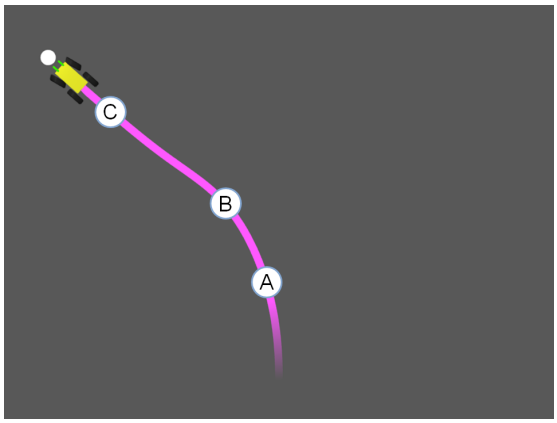


Figure 6: The path of a photophilic vehicle configured with the 'lust' motor torque transfer function (see Figure 5a). The labelled circles indicate the position of the vehicle at particular points in time (A = 10.0 seconds, B = 14.0 seconds, and C = 18.0 seconds) within the simulation. The view of the scene from the vehicle's perspective at these time points is illustrated in Figure 7.

from the target as a function of time is shown in Figure 8a. As can be seen from this figure, both vehicles begin the simulation at a distance of 21 meters from the light. They then move towards the light at similar speeds, with LoveBot ultimately coming to rest at a distance of 3 meters from the light. Figure 8b shows the motor torque values for LoveBot and LustBot across the course of the simulation. As is evident from this figure, LoveBot's motor torque drops to zero at 20 seconds, which corresponds to a distance of 4.5 meters from the target light (i.e., 1.5 meters before the vehicle actually comes to rest).

Both LustBot and LoveBot are configured to exhibit positive phototactic responses to a light source (i.e., they are configured with the `Photophilia` response type). The third vehicle I wish to introduce is the LoathingBot. LoathingBot is, in fact, a slightly 'mutated' version of the LustBot.[14] It has the same settings as LustBot, including the very same visuo-motor transfer function. The only difference between LustBot and LoathingBot concerns the value of the `ResponseType` property. Whereas this property is set to a value of `Photophilia` in the case of LustBot, LoathingBot is configured as a photophobic vehicle; it is thus configured to exhibit negative phototactic responses to a light source.

Let us now observe the effect of this change in the context of a specific simulation. Figure 9 illustrates the path of the LoathingBot as it encounters a light source positioned ahead of it. As can be seen from the figure, the vehicle

[14] In this particular case, of course, the 'mutation' is generated by a human designer; there is, however, no reason why vehicle properties, including the shape of specific transfer functions, should not be the target of evolutionary computation algorithms. This reveals one of the ways that the PBF may be used in the context of evolutionary robotics research (see Harvey et al., 2005).

(a) Vehicle's view at 10 seconds.

(b) Vehicle's view at 14 seconds.

(c) Vehicle's view at 18 seconds.

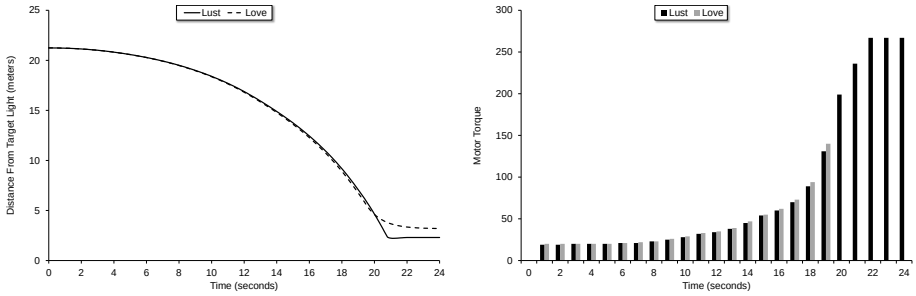
Figure 7: The vehicle's view of the scene through left and right eyes at the points indicated in Figure 6.

initially moves towards the light. Then, as it nears the light, it suddenly veers to the right, thereby circumventing the light source.[15]

At first sight, this pattern of behaviour may seem a little surprising. Inasmuch as the vehicle is configured to exhibit a phobic response to a light source, why does it not simply move away from the light source at the beginning of the simulation? Why does it appear, at least initially, to move in the direction of a light source that is placed directly ahead of it?

We could, of course, configure a vehicle to exhibit this more extreme form of

[15] See `https://youtu.be/8K3liDgYnA8`.



(a) Distance From Target (b) Motor Torque

Figure 8: (a) The distance of two vehicles from a fixed light source across the course of a simulation. One vehicle is configured with the 'lust' motor torque transfer function (see Figure 5a), while the other vehicle is configured with the 'love' motor torque transfer function (see Figure 5b). (b) The computed motor torque values for LustBot and LoveBot across the course of the simulation.

photic aversion, characterized by the complete avoidance of all light sources. I will, however, leave this as a challenge for users of the PBF and limit myself to an explanation of LoathingBot's behaviour.

Recall that LoathingBot is a subtly morphed version of LustBot. It thus inherits all of LustBot's response tendencies, with the exception of those related to the idiosyncrasies of the steering system. In contrast to a photophilic vehicle, LoathingBot will not steer in the direction of a light source. In fact, its steering angle is inverted, so that (e.g.) a 10° turn *towards* a light source is transformed into a 10° turn *away* from a light source.[16] Despite this, LoathingBot is still, to a large degree, 'energized' by the presence of light stimuli. High intensity illumination will thus continue to drive the vehicle's motors in precisely the same manner as that seen for the LustBot. (The more light a vehicle detects, the greater the amount of torque is applied to its wheels.) The only difference is that once a disparity in illumination is revealed by the vehicle's retinas, the vehicle will orient its front wheels so as to steer in the direction of the lowest illumination. Given LoathingBot's position at the beginning of the simulation, the amount of light falling on its retinas is equal, and the vehicle does not thereby commit to a steering response—it simply moves forward. Once a disparity is detected, the vehicle initiates a turning response by altering its steering angle. But even this does not take immediate effect, since it typically takes time for the *actual* steering angle to match the vehicle's *target* steering angle. The result, in this case, is that the vehicle initially moves towards the light, and then veers off to one side or the other.

At this point, it should be noted that all these simulations (i.e., those

[16] In fact, the response characteristics of the vehicle's steering system is controlled via a separate transfer function, called the *rotation function*.

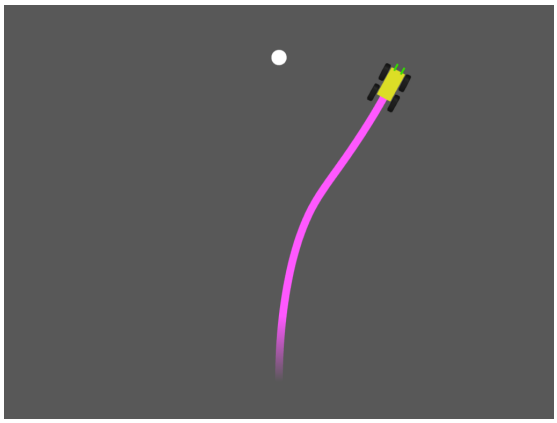


Figure 9: The path of a photophobic vehicle (i.e., LoathingBot) taken in response to a light placed directly ahead of it (simulation duration = 16 seconds).

pertaining to Lust, Love, and Loathing) were all performed with vehicles whose eyes were oriented with a particular offset. Instead of pointing directly ahead (i.e., 0° rotation), each eye was oriented 30° relative to the vehicle's y (or 'up' axis). The effect of this manipulation is to provide each eye with a somewhat different view of the scene, thereby reducing (while not entirely eliminating) the overlap in the eye cameras' view frustums. Different eye orientations will obviously yield differences in vehicle behaviour, with greater overlap in the vehicle's field of view reducing the magnitude of steering responses[17] and making it harder for vehicles (or any rate, Lust, Love, and Loathing bots) to exhibit phototactically-relevant behaviours. Such observations highlight the importance of bodily structure to overt behaviour and thus reveal the relevance of virtual world simulations to issues in embodied cognitive science. In fact, to the extent that intentional characterizations of vehicle behaviour are made with respect to responses that depend on bodily structure, it may begin to look as though intentionality has as much to do with a vehicle's morphological design as it does with the computational organization of its inner control mechanisms.

4.2. Discrimination

The behavioural responses of LustBot, LoveBot, and LoathingBot are all based on the intensity of a light source as 'perceived' by a particular vehicle. There is, however, no reason why vehicle behaviours cannot be coordinated with respect to more complex features of the visual environment. As we have seen, visual processing in the PBF is really just a form of image processing (see Section 3.2). In creating the information for retinal processing, each eye

[17] When the eyes are oriented directly forward, the two eyes share largely overlapping fields of view and any differences in light intensity between the eyes is minimal. This leads to a much-reduced level of turning behaviour.

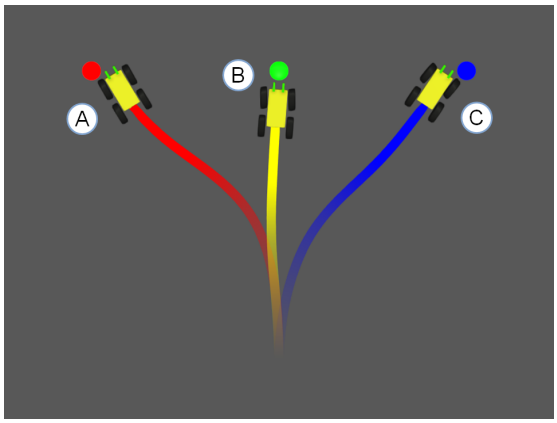


Figure 10: Trails showing the movement of vehicles with different colour preferences, relative to the position of red, green, and blue lights. Vehicle A is configured to prefer red (R = 255, G = 0, B = 0) objects, vehicle B is configured to prefer yellow (R = 255, G = 255, B = 0) objects, and vehicle C is configured to prefer blue (R = 0, G = 0, B = 255) objects.

renders scene information to a `RenderTexture` asset, and this asset can be processed in the same way as any other image. We can thus process information about each of the pixels in the `RenderTexture` asset with a view to calculating general, image-level properties (e.g., average luminance levels) or engaging in more advanced forms of visual processing (e.g., object recognition).

In processing a `RenderTexture` asset, it is possible to split the image into its constituent red, green, and blue colour channels, thereby yielding information about the chromatic content of a scene. This provides a means of refining the behavioural responses exhibited by LustBot, LoveBot, and LoathingBot. In particular, it provides the basis for a form of colour sensitivity, in which vehicles are able to discriminate between lights of different colours and selectively approach some lights while avoiding others.

Let us therefore become acquainted with ChromaBot. ChromaBot is more or less identical to the vehicles we have already encountered. Its only distinguishing feature is that in place of the familiar `PhototaxicBehaviour` component it possesses a new behaviour component, called `ColourSensitivityBehaviour`. Although this sounds like a pretty significant alteration, the `ColourSensitivityBehaviour` is a component that, in fact, derives from `PhototaxicBehaviour`, and it thus inherits all the functionality of `PhototaxicBehaviour`. The only thing it adds to this base level of functionality is the capacity to process colour information and coordinate a vehicle's behaviour with respect to a 'preferred' colour value—a value that is exposed as a property in Unity's Inspector window and which can thus be set by a human user.

Figure 10 shows the paths taken by three vehicles, each of which begins the simulation in the same starting location. The colour of the trails marking the paths of the three vehicles indicates their preferred colour. Vehicle A is thus configured to prefer red lights, vehicle B is configured to prefer yellow lights,

and vehicle C is configured to prefer blue lights.[18] The responses of vehicles A and C are unsurprising: when confronted with an array of red, green, and blue lights, vehicle A navigates towards the red light, while vehicle C navigates towards the blue light. The response of vehicle B is somewhat less predictable. Vehicle B is configured to prefer yellow illumination, which corresponds to an RGB colour value with equal (and maximal) brightness in the red and green colour channels. Relative to this setting, it might be thought that Vehicle B would be caught between a choice of the red and green lights, both of which are equally 'enticing'. The upshot, we might think, is something akin to a Buridan's ass state-of-affairs, in which the vehicle opts for neither the red nor the green lights, but happily settles for an intermediate course, steering a path between the lights and sailing off into the dimly lit virtual yonder. As is clear from Figure 10, this is not what happens. Vehicle B settles for (chooses?) the green light, and this result is consistently obtained across multiple simulations.

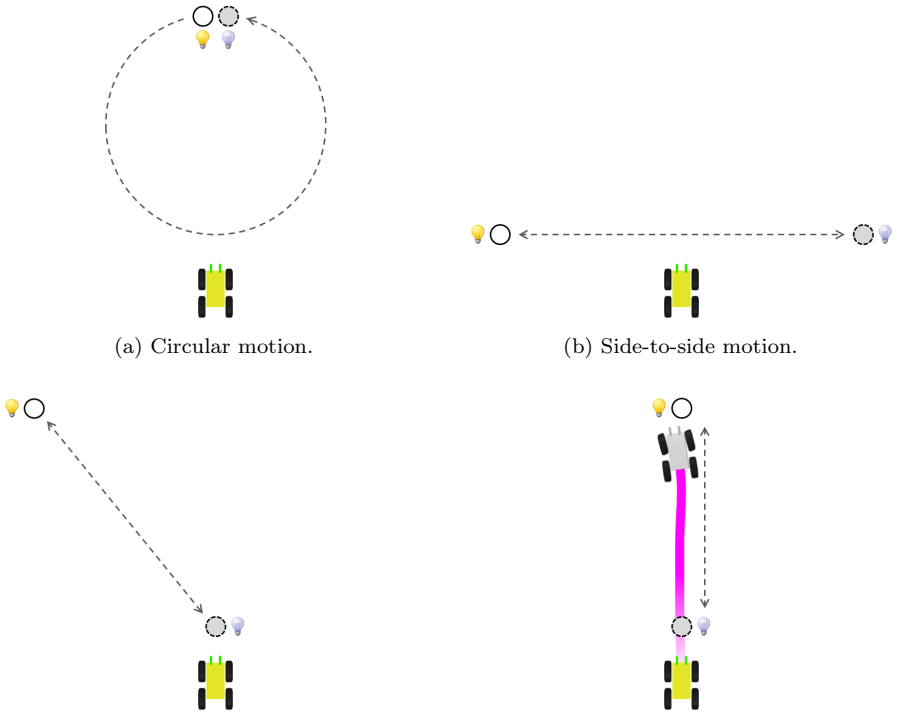
In order to understand (at least in a mechanistic sense) the reasons for vehicle B's behaviour, it is important to focus attention on the starting position of the vehicles in this simulation. From vehicle B's perspective, the red and green lights are equally attractive, but the green light is the closer of the two lights and thus yields the greatest level of illumination. Even from the outset of the simulation, therefore, vehicle B is poised to 'perceive' the green light as the more attractive option. It is thus the relative proximity of the red and green lights that ultimately leads to the observed response bias.

Setting aside the explanatory details, what is interesting about this particular case is the way in which a vehicle's physical location within a spatial environment helps to resolve a state-of-affairs that might otherwise have yielded a somewhat suboptimal outcome (i.e., a photophilic vehicle failing to home in on an easily observable visual target). The case also highlights the value of performing simulations as opposed to relying solely on thought experiments. In particular, a thought experiment might lead us to think that vehicle B would choose an intermediate course between the red and green lights, thereby leading us to overlook aspects of the vehicle's behavioural repertoire and thus (perhaps) its psychological complexity. This is not to deny or downplay the utility of thought experiments; it is simply a reminder of the fact that the 'real' world is full of surprises.

4.3. Aggravation

As mentioned in Section 3.2, there are two types of eyes within the PBF: simple and complex. A complex eye, recall, is an eye in which the retinal surface is divided into a number of regions or retinal patches. These patches provide the basis for a form of directional sensitivity in a vehicle's responses to a light source. In particular, a vehicle can be configured to respond to lights that move in a particular direction. Examples include a light that moves left to right, or a light that moves towards or away from a vehicle.

[18] See `https://youtu.be/q8gdrGFHOpQ`.



(a) Circular motion.

(b) Side-to-side motion.

(c) Ipsilateral diagonal motion.

(d) Forward/backward motion.

Figure 11: Virtual vehicle exhibiting directional sensitivity. The movement of a light in (a) circular, (b) side-to-side, or (c) diagonal directions fails to elicit any movement of the vehicle. (d) Only when the light moves in the forward/backward direction is the (photophilic) behaviour of the vehicle triggered. This form of directional sensitivity is realized by the capacity of the vehicle to detect a medio–lateral increase in luminosity across its virtual retinas.

Figure 11 shows the response of a vehicle to a light that moves in one of four motion paths: a circular path, a side-to-side path, an ipsilateral diagonal path, and, finally, a forward/backward path. In these simulations, the vehicle was configured to respond to a specific pattern of visual stimulation, namely, one yielded by a light that approaches the vehicle from the front (see Figure 11d). As a light with this motion path approaches the vehicle it will cause a progressive, time-dependent change in illumination across each of the vehicle's retinas, and this will alter the average illumination that is computed for each of the vehicle's retinal patches. Providing the light is moving directly ahead of the vehicle, and the vehicle's eyes are not rotated too far from the vehicle's forward direction, then the average illumination of each retinal patch will be greatest for the most medial parts of the retina (i.e., those closest to the vehicle's midline) and the direction of increase will be in the medio–lateral direction.

It is by detecting such patterns of stimulation that the vehicle in Figure 11 is able to exhibit a selective response to a light that moves in the forward/backward

direction. The vehicle, in this case, is equipped with a specific vehicle behaviour component, called `LoomingLightBehaviour`. The purpose of this component is to monitor changing patterns of visual stimulation and detect a pattern of stimulation corresponding to a looming light (i.e., a light that approaches the vehicle from directly ahead). Once detected, the component activates the vehicle's motor, and the vehicle then responds in the manner of the LustBot described in Section 4.1. In other words, this particular vehicle, let's name it AggroBot, is, at root, a LustBot that, in a manner similar to a subsumption architecture (see Brooks, 1986), has an additional layer of control. This additional layer of control monitors the incoming stream of sensory information and inhibits the default (locomotory) behaviour of the LustBot. It continues to do so until a particular sensory contingency (i.e., that corresponding to a looming light) is detected. At this point, control is ceded to the (more primitive or 'evolutionary' ancient?) layer controlling phototactic responses and the true nature of the LustBot vehicle is thereby revealed.

As is illustrated by the video accompanying this simulation,[19] once the vehicle's motor is activated, it approaches the light in the manner of the earlier LustBot simulation (see Section 4.1). In this case, however, the target light is programmed to respond to collisions with other objects and will deactivate if collisions occur with sufficient force. Upon observing such behaviour, it is easy to conclude that the vehicle in this simulation is attempting to ignore the light that dances before it. Its patience persists until the light taunts it with a particular pattern of motion, and at this point, the vehicle's patience gives out and it attacks the light. Such a vehicle, we might say, is 'annoyed' or 'irritated' by the light.

4.4. Attraction

The simulations described thus far have all focused on a single vehicle. There is, however, no reason why a simulation cannot involve multiple vehicles. Naturally, there are limits regarding the complexity of simulations (here cashed out as the number of vehicles included in a scene) and the ability of the host machine to run simulations in real-time (see Section 5). In general, however, simulations involving two or three vehicles should not present much of a problem.

The inclusion of additional vehicles into a simulation provides an opportunity to study issues of 'social' interaction. Consider, for example, a situation in which we take the LustBot described in Section 4.1 and place it in a scene that features a different kind of vehicle. This new vehicle (let us call it TeaseBot) is something of an odd critter. It has a light attached to its posterior, and this furnishes it with (let us suppose) a bio-luminescent capability. Obviously, the LustBot will be attracted to the light in the same manner as it was attracted to lights within its native (or, at any rate, natal) environment. But this sort of scenario is very different to any that LustBot has previously encountered. Here, the light is associated with another vehicle, and its location (and perhaps other properties,

[19] See `https://youtu.be/FmZFqv8aw1Q`.

such as colour and intensity) will alter based on TeaseBot's own psycho-motor proclivities.

With this in mind, let us take a closer look at TeaseBot. TeaseBot, as we have seen, comes equipped with a light affixed to its hindquarters. But, in addition to this, TeaseBot is equipped with a radial sonar sensor. The radial sonar, recall, allows a vehicle to detect the proximity of physical objects relative to its own location in the virtual world (see Section 3.2). TeaseBot is thus able to detect when a vehicle (or some other mobile entity) encroaches on its personal space. It is also able to determine the proximity of a vehicle, and, more importantly perhaps, the vehicle's direction of approach. Such information is used by TeaseBot's behaviour module, which, in the present case, is a component named `AvoidCaptureBehaviour`. The purpose of this module is to avoid direct contact with other vehicles by first detecting their presence (via the radial sonar sensor) and then activating the vehicle's locomotor system so as to move it beyond the reach of the approaching vehicle.

In one sense, this all seems straightforward. It requires, let us assume, an ability to process information coming from the radial sonar sensor and use the output of that processing to 1) activate the vehicle's motor by applying motor torque, and 2) change the steering angle in such a way as to move the vehicle away from any approaching vehicle. But given that we are following in Braitenberg's footsteps, and Planet Braitenberg is really just a means of supporting our pseudo-imaginary explorations of a fictional, cinematic[20] world, let us instead suppose that TeaseBot is merely the latest in a long line of vehicles whose evolutionary heritage includes an overriding concern with photic responses. In this case, it may make sense for TeaseBot's adaptive responses to piggyback on existing functionality, exploiting its pre-existing responses to light stimuli as a means of orchestrating its responses to new situations. It is for this reason that TeaseBot's motor responses, as generated by the `AvoidCaptureBehaviour` module, are only indirectly linked to the outputs of its radial sensor system. In fact, the way in which TeaseBot coordinates its motor responses is by 'imagining' certain kinds of visual stimuli. When TeaseBot thus detects an approaching vehicle (in this case, LustBot), it simply imagines the presence of a light in a particular region of its visual field. For example, if LustBot approaches from TeaseBot's left side, then TeaseBot will pretend (i.e., imagine) that it has detected a light in its right visual field, with the imagined intensity of the light changing in response to LustBot's proximity.[21] Based on the assumption that TeaseBot and LustBot are kindred light lovers, TeaseBot will now move in the direction of the imagined light, just as it would have moved if it had 'perceived' a real (albeit virtual) light within its

[20] Virtual worlds are cinematic in the sense that are rendered in the same way as a conventional movie or film. That is say, the virtual world of a Braitenberg vehicle, and, indeed, the vehicle itself, are rendered as a rapid succession of computer-generated images (see Gaut, 2009).

[21] In essence, TeaseBot's sonar sensor plays no direct role in controlling TeaseBot's overt behaviour. Instead, the sonar sensor influences behavioural output by exploiting TeaseBot's capacity to imagine counterfactual states-of-affairs, in this case, states-of-affairs featuring stimuli of motivational significance.

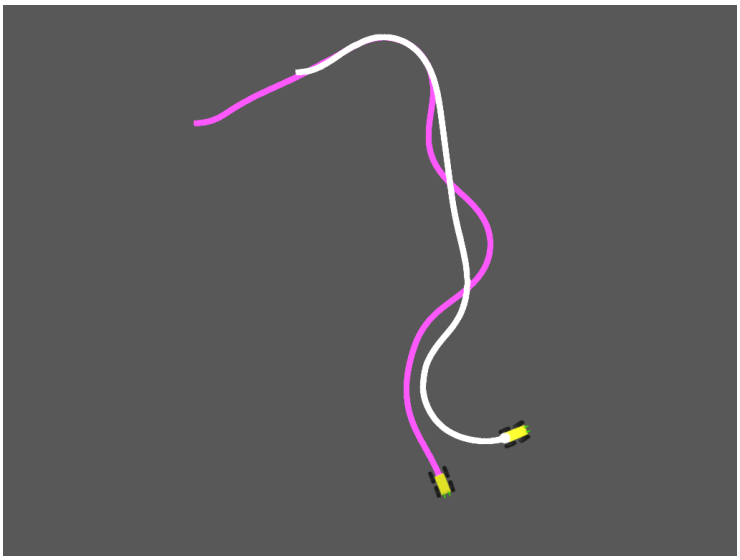


Figure 12: The paths traced by two vehicles as one vehicle pursues the other. LustBot's trajectory is represented by the pink line, while TeaseBot's trajectory is represented by the white line.

visual field. The advantage of this organizational scheme is that TeaseBot is able to exploit whatever functionality was bequeathed by its evolutionary forebears. More importantly, at least from the author's perspective, this approach reduces the need to write a large amount of computer code, much of which probably has limited applicability beyond the present simulation.

Figure 12 shows the actual paths taken by TeaseBot and LustBot during the course of one simulation. LustBot is initially positioned behind TeaseBot. As LustBot approaches, TeaseBot begins to move away. There ensues a complex, albeit transient, dance between the two vehicles, with LustBot attempting to adjust its trajectory in response to TeaseBot's movements and TeaseBot attempting to counter LustBot's advances. Given the nature of this reciprocal causal coupling between the two vehicles, it will probably come as no surprise to learn that social interactions of this sort are inherently chaotic. Despite the fact that all simulations begin with the same starting conditions, it is seldom the case that any two simulations will yield the same results, at least with respect to the paths taken by the two vehicles. This is in stark contrast to other simulations, where the same starting conditions tend to culminate in more or less the same behavioural outcomes.

4.5. Territoriality

For the most part, vehicle behaviour modules are self-contained code components that control the sensorimotor interactions of a vehicle with its surrounding environment. But not all vehicle behaviours need to operate independently of

other resources. In fact, vehicle behaviour modules may draw on other resources to implement complex forms of behavioural control. One example of this comes from a simulation that attempts to model territorial behaviour in a virtual vehicle.

The simulation in question features a vehicle (let us call it TerriBot) that is equipped with a behaviour component called `TerritorialBehaviourSM`. The goal of this component is to coordinate the responses of a vehicle with respect to an object (in this case, an illuminated sphere) that penetrates a region corresponding to the vehicle's territory. The vehicle is initially located at the centre of its territory. Once an invading object has been detected, the vehicle engages in a number of manoeuvres that are intended to remove the object from its territory. Once this goal has been achieved, the vehicle reorients itself and returns to the centre of the territory (see Figure 13).

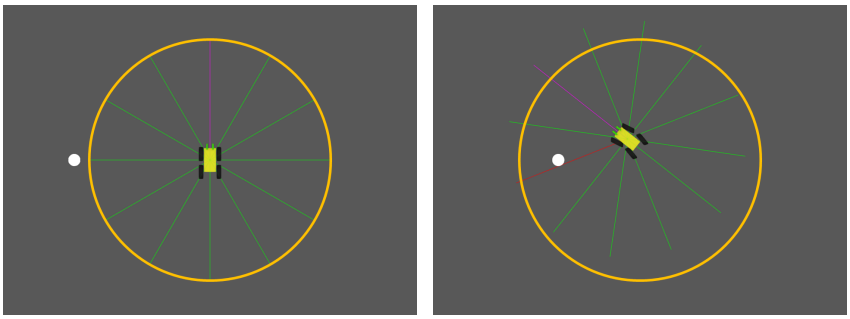
Despite the use of a light as part of this simulation, no aspect of TerriBot's behaviour is, in fact, coordinated with respect to visual stimuli—the light is simply used for demonstration purposes. TerriBot instead relies on a combination of tactile information and information supplied by a radial sonar sensor. For the purposes of this simulation, the radial sonar sensor was configured with an array of 12 sonar beams. By processing the information supplied by these beams throughout the course of the simulation, TerriBot is able to orient itself until it is facing the invading object; at which point, it proceeds to push the object out of its territory.[22]

Unlike the simulations described above, TerriBot's behaviour module (i.e., `TerritorialBehaviourSM`) actually does very little of the information processing required to implement this behaviour. Much of the computational burden is, in fact, borne by a state machine, which was created using Unity's in-built Mecanim animation system.[23] While Mecanim is mostly used to implement state machines for character animation, especially for humanoid game characters, it can also be used as a general state machine editor, enabling users to create state machines of the more conventional variety, e.g., those used to control the behaviour of Artificial Intelligence (AI) systems.

The state machine developed for the TerriBot simulation is depicted in Figure 14. Here, the individual states of the state machine correspond to behavioural states of the vehicle and state transitions occur in response to the satisfaction of specific conditions. The default state is 'AtHome', which represents the vehicle's initial resting state in the centre of its territory. Information from TerriBot's sonar system is periodically monitored by the `TerritorialBehaviourSM` behaviour module, and this information is then used to set the value of parameters associated with the state machine. It is these parameters that control the transition between the various states of the state machine. As each state becomes active, it triggers the execution of a custom script, which is implemented as a

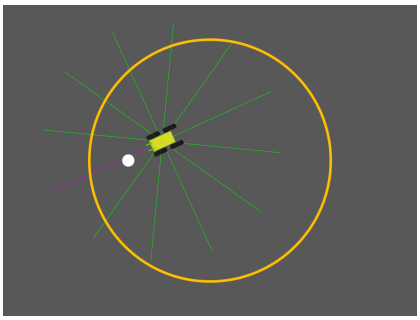
[22] See `https://youtu.be/G2V4ctOLwK4`.

[23] See `https://docs.unity3d.com/462/Documentation/Manual/MecanimAnimationSystem.html`.

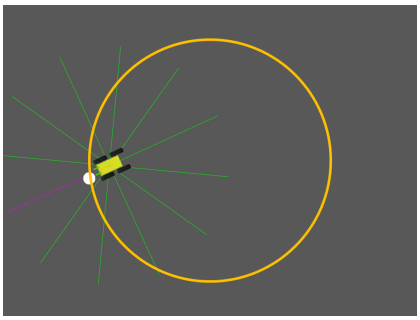


(a) At home base.

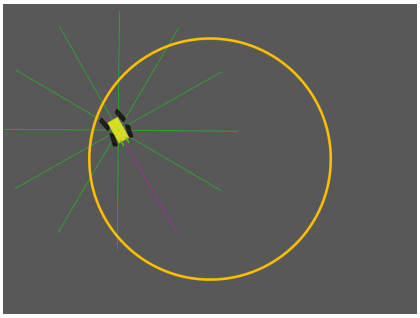
(b) Orienting to target.



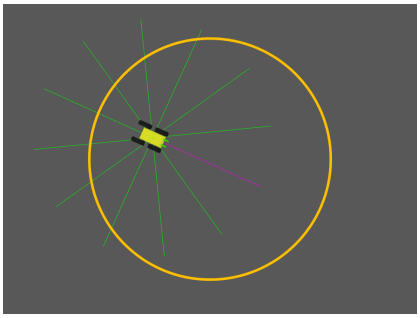
(c) Attacking target.



(d) Removing target.



(e) Reorienting to home base.



(f) Returning to home base.

Figure 13: Screenshots of a virtual vehicle exhibiting territorial behaviour. The coloured lines emanating from the vehicle's body correspond to the beams of a radial sonar sensor. The beams report contacts with any 'physical' objects in the vicinity of the vehicle.

subclass of the Unity `StateMachineBehaviour` class. It is this script, rather than `TerritorialBehaviourSM`, that sets the value of motoric variables and thus influences TerriBot's overt behaviour.

Why resort to the use of state machines to control behaviour, especially since the functionality of a state machine could, in principle, be implemented by code within the vehicle's behaviour module? The primary reason relates to the nature

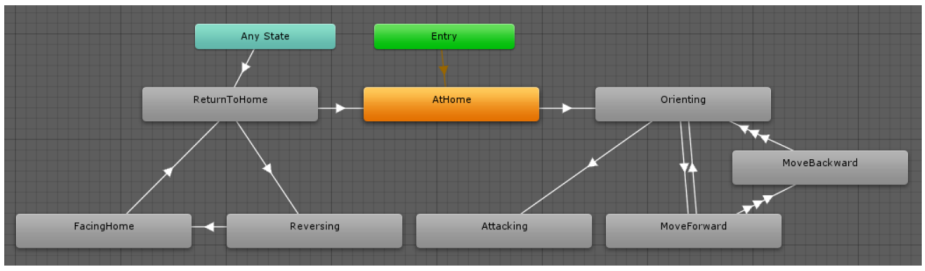


Figure 14: A state machine used to control the responses of a virtual vehicle engaged in territorial behaviour.

of the graphical representation supported by the Unity editor. This supports a visual approach to the design of behaviour control systems, which helps to explicate the different behavioural states and sensorimotor contingencies that are relevant to particular forms of behavioural competence. In many cases, this affords an approach to the design of vehicle control mechanisms that is much more intuitive, straightforward, and fun than purely code-based approaches.

There are a couple of additional features of this simulation that are worth mentioning. The first is that state machines are only one of the assets that can be used by a vehicle behaviour module to control behaviour. Another kind of asset is a neural network, which could be created as a `SerializedObject` and then associated with a custom vehicle behaviour component. This highlights one of the ways in which the current work could be extended (see Section 6). In particular, it is possible to imagine situations in which the task of sensorimotor coordination is achieved via the use of one or more neural networks, each of which is used to control some aspect of a vehicle's behaviour within a given (virtual) ecological niche.

A second feature of the simulation that is worth mentioning concerns the way in which the `TerritorialBehaviourSM` component serves as an interface for control systems that are, to some extent, independent of the TerriBot simulation (and even the Unity environment!). The state machine used by the TerriBot simulation is, of course, a Unity-specific asset that is intended to be used by applications built on top of the Unity game engine. Other resources, however, need not form part of the Unity game creation system. In previous work, for example, we have implemented a network interface to the ACT-R cognitive architecture,[24] which enables vehicles to be controlled by ACT-R models that run on separate machines (Smart et al., 2016; Smart & Sycara, 2015a). In these situations, a custom vehicle behaviour module, dubbed `ACTRControlledBehaviour` is used as the computational point of contact between the Unity environment and an external ACT-R cognitive model, with the vehicle behaviour module posting sensory information to the ACT-R model and

[24] See Anderson et al. (2004).

responding to whatever motor commands are returned by the ACT-R system.

5. Data Analysis & Performance

As noted in Section 3, one of the virtues of virtual worlds is that they simplify the task of capturing and analysing data. In fact, the open-source nature of the PBF makes it relatively easy to create components that record data about more or less any aspect of a simulation. A custom data capture component could, for example, record data about the position and orientation of a particular vehicle throughout the course of a simulation by including routines that execute in the context of Unity's game update cycle (i.e., as part of the `Update()` or `FixedUpdate()` functions).

The PBF includes a number of built-in components that support the acquisition of behavioural data. These components inherit from an abstract class, called `DataCapture`, which is itself a descendant of Unity's `MonoBehaviour` class. The `DistanceDeltaRecorder` class, for example, tracks the relative distance of a target vehicle from another object in the scene, which could be either a static object (i.e., an object whose location does not change throughout the course of a simulation) or a dynamic object (e.g., another vehicle) whose location varies according to the result of animation routines or the application of physical forces. (This class was used to yield the data for Figure 8a in the present paper.) From a procedural perspective, the `DistanceDeltaRecorder` class records the distance between two objects throughout the course of a simulation and then serializes the recorded data (as a series of comma-separated time/distance values) to a specified file at the end of the simulation.

As with any kind of computer simulation, the PBF is hostage to the ever-present trade-off between complexity and performance. As a general rule, performance can be measured in terms of the simulation's framerate or Frames Per Second (FPS), i.e., the number of frames that the game engine manages to render for each second of the simulation. This will obviously vary based on the nature of the computational platform on which the PBF is run; nevertheless, a number of steps can be taken to ensure that simulations are able to run in real-time. These include limiting the number of vehicles instantiated within a particular scene and reducing the rate of a vehicle's behaviour update cycle. It is important to bear in mind that with multiple vehicles, computational demands can quickly escalate. This is especially true in situations where retinal output is subject to complex forms of visual analysis. With a retinal size of 200×200 pixels, the raw visual data for a (binocular) vehicle comprises a data set of 80,000 ($200 \times 200 \times 2$) pixels, each of which yields values for the red, green, and blue colour channels. Clearly, if the purpose of a simulation is to study the behaviour of multiple vehicles in scenes requiring complex forms of visual analysis, then the computational demands of the simulation are likely to be prohibitive.

There are a number of ways of dealing with the performance overheads imposed by complex simulations within the PBF. The most obvious of these is to alter the real-time speed of the simulation using Unity's `Time.captureFramerate` property. This is typically used to slow the speed of game playback to allow

for the execution of computationally costly procedures, and it is thus suited for situations where simulation routines must execute with a particular frequency, irrespective of their computational complexity. The use of this strategy, of course, means that a simulation no longer runs in real-time. Crucially, however, this does not affect the operation of the aforementioned `DataCapture` components, which continue to log data based on the timestamps that would have been obtained if the simulation had been run in real-time. This means that `DataCapture` components can be used to record data from a simulation at a reduced rate, and the data from the simulation can then be used to implement re-runs of the simulation at real-time speeds.

As an example of this technique, the PBF includes a `VehicleDataRecorder` component, which records data about the motoric variables (e.g., motor torque, eye rotation, etc.) of a vehicle throughout the course of a simulation. The data file generated by this component can be referenced by another component, called `RecordedVehicleBehaviour`, which can be used in a subsequent simulation to recreate the pattern of motor output that would have been obtained if the relevant vehicle had deployed the sorts of sensory processing featured in the original simulation (i.e., the one for which real-time simulation was not possible). Using this strategy—which amounts to a form of offline rendering of vehicle behaviour (or "vehicle behaviour baking")—it is possible to replay complex simulations in real-time for the purpose of studying (e.g.) the response of human observers to patterns of vehicle behaviour.

6. Future Work

As it stands, the PBF provides a platform for studying the behaviour of virtual entities that are modelled after the (even more virtual?) inhabitants of Valentino Braitenberg's imaginarium. This does not mean, however, that the PBF is limited to the design of Braitenbergian vehicles. Extensions of the PBF could thus be used to undertake simulations similar to those encountered in the context of artificial life research (Nakada et al., 2018; Terzopoulos et al., 1997). In particular, the sensor and motor modules described above could be adapted (and extended) to model the sensorimotor ecologies of different organisms, with the aim of better understanding the (perhaps extended) mechanisms that underpin various forms of intelligent response. Consider, by way of an example, the claim that some of the behaviours exhibited by spiders may stem from the operation of extended mechanisms (Japyassú & Laland, 2017; Smart et al., 2010). Such mechanisms, it has been suggested, reduce the computational burden associated with tasks like web weaving by factoring in the contributions made by a spider's body and the structure of the web itself (see Smart et al., 2010). By using the PBF as the basis for computational simulations of spider web weaving, it is relatively easy to see how virtual environments could be used to study the dynamics of an ostensibly extended (cognitive?) routine and thereby contribute to our understanding of the way in which silk-extended 'minds' are quite literally spun from a material fabric that transcends the traditional metabolic boundaries of skin and skull (or, in this case, the biological borders

of the cephalothorax). It is in this sense, I suggest, that the PBF provides a starting point for research that seeks to develop computational models of extended cognitive systems and (perhaps) extended minds (Clark, 2008; Clark & Chalmers, 1998). Given that synthetic approaches mandate the bottom-up design of extended mechanisms, there should, in this case, be little doubt as to what (intra- or extra-organismic) resources are constitutively relevant to the realization of a simulated cognitive routine. This perhaps provides the basis for a synthetically-oriented mechanistic approach to extended cognition, one that potentially avoids the ostensible shortcomings associated with more analytically-oriented approaches (see Baumgartner & Wilutzky, 2017).

The PBF also provides the foundation for work that explores different ways of controlling the behaviour of virtual agents. The simulations described above rely on computational components that, for the most part, emulate the kinds of control mechanism envisioned by Braitenberg (1984). There is, however, no reason why future uses of the PBF should be limited to these kinds of control system. Schrodt et al. (2017), for example, describe a cognitive architecture, dubbed SEMLINCS, which was used to control the behaviour of an 'embodied' character within a 2D game. Interestingly, Schrodt et al. (2017) suggest that their cognitive architecture allows a virtual character to learn "discrete production rule-like structures from its own, autonomously gathered, continuous sensorimotor experiences" within the game environment (Schrodt et al., 2017, p. 343). Such work exemplifies the recent interest in using cognitive architectures to control the behaviour of virtual characters (see Turner et al., 2016), and this identifies one of the future targets of research using the PBF. As noted in Section 4.5, there is no reason why the PBF cannot be used in conjunction with cognitive architectures, and the PBF already contains components that can be used to support interactions with one of the most widely used cognitive architectures, namely, ACT-R (see Anderson et al., 2004).

Another strand of work that relies on the use of virtual environments to test cognitively-potent forms of computational control is epitomized by work into so-called deep learning systems. Perhaps the best example of this work emanates from research at Google DeepMind,[25] with recent studies demonstrating the capacity of deep neural network architectures to learn sophisticated game-play responses (Mnih et al., 2015). From a cognitive science perspective, this link with deep neural networks looks to be of particular interest given the approximate similarity of deep learning architectures to predictive coding or predictive processing accounts of cognition (Clark, 2013; Friston, 2009; Huang & Rao, 2011). This speaks to an additional use of the PBF as part of attempts to study the operation of hierarchically-organized predictive processing regimes in situations that require complex forms of sensory processing and motor response. In particular, the PBF could be used as the basis for simulations that explore the contribution of action-oriented predictive processing architectures (see Clark, 2013) to aspects of embodied and situated intelligence. As a means of supporting

[25] See `https://deepmind.com/`.

such simulations it is likely that a richer set of sensory and motor components will need to be developed.[26] This is especially true when it comes to the processing of information that originates from within an agent's bodily borders (i.e., interoceptive information). According to the proponents of predictive processing, such information is deemed to be relevant to aspects of conscious experience, including the subjective experience of emotion (Seth, 2013). Relative to such claims, it seems that greater attention will need to be paid to the representation and processing of computational equivalents to interoceptive information, especially if future versions of the work reported here are to be used to study (and perhaps instantiate!) the information flow patterns that undergird an agent's phenomenal experience of (in this case, virtual) reality.

Ultimately, of course, the PBF is intended as a community resource—as something that can be used, adapted, and extended by different individuals (or organizations) for different purposes. This makes it hard to provide an exhaustive list of the ways in which the framework may evolve. In the first instance, the framework, as it is presented here, is likely to be useful for educational or research purposes, especially when the use of physical robots is ruled out due to (e.g.) cost constraints. The use of the framework in a more entertainment-oriented context (e.g., to implement commercial video games) is somewhat harder to imagine given the performance overheads associated with (e.g.) sensor processing. Nevertheless, there may be situations in which the PBF could be used as part of so-called serious games or games-with-a-purpose (von Ahn, 2006; Michael & Chen, 2006). As an example, imagine a game in which the objective is to configure a vehicle's sensorimotor control architecture with the aim of supporting its autonomous movement through a variety of different (perhaps 'extra-terrestrial') landscapes. Such games could provide an introduction to the complexities of autonomous agent control, with more advanced versions (e.g., a game to control a future Martian Rover or Europan Ocean Explorer) perhaps helping to distribute the challenge of discovering and testing disparate behavioural control algorithms to a global community of 'gamers'. Such approaches look to be of particular value given the use of games in a variety of other scientific contexts (e.g. Good & Su, 2011). In particular, such approaches promise to transform a formidable (and perhaps ultimately intractable) *individual* engineering effort into something that more closely resembles a large-scale, socially-distributed search for interesting (and perhaps innovative) design solutions. It is in this sense, I suggest, that we are provided with a useful insight into at least one of the ways in which the current widespread enthusiasm for computer games could be applied to some of the problems confronting contemporary cognitive science, helping to bring about a state-of-affairs in which cognitive systems research is brought firmly and squarely into the realms of citizen science.

[26] Although see Thornton (2017) for an example of recent work that purports to study the behaviour of Braitenberg vehicles within a predictive processing framework.

7. Conclusion

The present paper has described how a contemporary game engine, commonly used in the construction of 3D and 2D video games, can be used to create a virtual environment that supports empirical simulations inspired by the work of Valentino Braitenberg (1984). Braitenberg's approach is one that can be characterized as both situated and synthetic. It is situated in the sense that Braitenberg sought to show how complex behaviour might emerge from the interaction between simple control systems, aspects of bodily morphology, and features of the wider (extra-agential) environment. In this respect, Braitenberg's vision resonates with work of a more recent vintage, especially that pertaining to the study of the embodied and environmentally-embedded mind (Clark, 1997, 2008; Pfeifer & Bongard, 2007). Braitenberg's approach can also be characterized as synthetic in the sense that he sought to show how psychologically-salient patterns of behaviour might emerge from the operation of specifically designed (and thus well-understood) 'generative' mechanisms.

The PBF is intended to support an approach to cognitive science that echoes both the situated and synthetic aspects of Braitenberg's work. By building and manipulating virtual representations of both the vehicles and the environments originally described by Braitenberg (1984), we are able to empirically investigate some of the claims he made regarding the mechanistic realization of psychologically-interesting patterns of behaviour. We can, for example, use the PBF to study the different behaviours that emerge when we change the orientation and placement of light sensors on a vehicle's body, the position and intensity of lights within an environment, or the sensitivity of a vehicle's visuo-motor system to lights of different colours and intensities. Such manipulations provide us with an important opportunity to develop a deeper understanding of how a variety of forces and factors (operating at the level of sensorimotor coordination mechanisms, body morphology, and environmental structure) conspire to give rise to behaviour that warrants the ascription of intentional mental states to an environmentally-embedded agent. Such work, it should be clear, builds on the situated elements of Braitenberg's work, helping us understand how different forms of cognitive response might be rooted in a complex web of causally-relevant forces and factors that are distributed across the brain, the body, and the world (Clark, 1997). In this sense, the PBF provides the basis for work into what might be called *computational situated cognition*, a field of study whose aim is to use computer simulation techniques to study issues pertaining to the physically-embodied, environmentally-embedded, and materially-extended mind (see Clark, 2008). Clearly, the simulations reported in the present paper—which rely on the use of relatively simple control mechanisms—are not intended to approximate the kinds of processes seen in the case of human cognition. Nevertheless, we can perhaps begin to imagine how the current framework could be extended with richer perceptual environments (similar to those encountered in contemporary video games), more sophisticated behavioural agents (such as humanoid avatars), and more advanced forms of sensorimotor processing (such as those provided by deep neural networks and/or cognitive

architectures).

It might be thought that a synthetic approach to psychology is one that is best served by the design and development of physical robots. There are, however, a number of reasons to commend the use of virtual environments for cognitive systems research. By using a virtual environment, for example, it is possible to rapidly design and reconfigure virtual vehicles, exploring the effect of morphological and environmental manipulations without the overhead that often comes with the (re-)design of real-world robots. This does not mean, of course, that the process of building virtual vehicles is easy; it often takes considerable ingenuity and expertise to engineer a simulation that yields a particular pattern of behaviour, and simulations are often apt to yield surprises. Just as it takes a lot of practice to become an expert game-player, so too it takes a lot of practice to become an expert virtual psychologist. It is this allusion to the realm of games and game-play that, I think, best reflects the pedagogical and scientific significance of the PBF. The design of virtual vehicles is intended to be both challenging and informative, but it is also intended to be fun. Given the spirit of Valentino Braitenberg's work, it is perhaps this latter feature that best represents the nature of his enduring intellectual legacy.

Acknowledgements

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

von Ahn, L. (2006). Games with a purpose. *Computer*, *39*, 96–98.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036–1060.

Baumgartner, M., & Wilutzky, W. (2017). Is it possible to experimentally determine the extension of cognition? *Philosophical Psychology*, *30*, 1104–1125.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. Cambridge, Massachusetts, USA: MIT Press.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *Journal of Robotics and Automation*, *2*, 14–23.

Christel, M. G., Stevens, S. M., Maher, B. S., Brice, S., Champer, M., Jayapalan, L., Chen, Q., Jin, J., Hausmann, D., & Bastida, N. (2012). RumbleBlocks: Teaching science concepts to young children through a Unity game. In Q. H. Mehdi (Ed.), *17th International Conference on Computer Games* (pp. 162–166). Louisville, Kentucky, USA: IEEE.

Clark, A. (1989). *Microcognition: Philosophy, Cognitive Science, and Parallel Distributed Processing*. Cambridge, Massachusetts, USA: MIT Press.

Clark, A. (1997). *Being There: Putting Brain, Body and World Together Again*. Cambridge, Massachusetts, USA: MIT Press.

Clark, A. (2008). *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. New York, New York, USA: Oxford University Press.

Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, *36*, 181–253.

Clark, A., & Chalmers, D. (1998). The extended mind. *Analysis*, *58*, 7–19.

Dewdney, A. K. (1987). Braitenberg memoirs: vehicles for probing behavior roam a dark plain marked by lights. *Scientific American*, *256*, 16–24.

Epstein, J. M. (Ed.) (2006). *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton, New Jersey, USA: Princeton University Press.

French, R. L. B., & Cañamero, L. (2005). Introducing neuromodulation to a Braitenberg vehicle. In *IEEE International Conference on Robotics and Automation* (pp. 4188–4193). Barcelona, Spain: IEEE.

Friston, K. (2009). The free-energy principle: A rough guide to the brain? *Trends in Cognitive Sciences*, *13*, 293–301.

Froese, T., & Ziemke, T. (2009). Enactive artificial intelligence: Investigating the systemic organization of life and mind. *Artificial Intelligence*, *173*, 466–500.

Gaut, B. (2009). Digital cinema. In P. Livingstone, & C. Plantinga (Eds.), *The Routledge Companion to Philosophy and Film* (pp. 75–85). Oxon, UK: Routledge.

Good, B. M., & Su, A. I. (2011). Games with a scientific purpose. *Genome Biology*, *12*, 1–3.

Harvey, I., Di Paolo, E., Wood, R., Quinn, M., & Tuci, E. (2005). Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life*, *11*, 79–98.

Hogg, D. W., Martin, F., & Resnick, M. (1991). *Braitenberg Creatures*. Technical Report E&L Memo No. 13, Media Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

Hosoya, T., Baccus, S. A., & Meister, M. (2005). Dynamic predictive coding by the retina. *Nature*, *436*, 71–77.

Huang, Y., & Rao, R. P. (2011). Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, *2*, 580–593.

Japyassú, H. F., & Laland, K. N. (2017). Extended spider cognition. *Animal Cognition*, *20*, 375–395.

Lilienthal, A., & Duckett, T. (2004). Experimental analysis of gas-sensitive Braitenberg vehicles. *Advanced Robotics*, *18*, 817–834.

Mataric, M. J. (1991). Navigating with a rat brain: A neurobiologically-inspired model. In J. Meyer, & S. W. Wilson (Eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behaviour* (pp. 169–175). Boston, Massachusetts, USA: MIT Press.

Mattingly, W. A., Chang, D., Paris, R., Smith, N., Blevins, J., & Ouyang, M. (2012). Robot design using Unity for computer games and robotic simulations. In Q. H. Mehdi (Ed.), *17th International Conference on Computer Games* (pp. 56–59). Louisville, Kentucky, USA: IEEE.

Menary, R. (Ed.) (2010a). *The Extended Mind*. Cambridge, Massachusetts, USA: MIT Press.

Menary, R. (2010b). Introduction to the special issue on 4E cognition. *Phenomenology and the Cognitive Sciences*, *9*, 459–463.

Michael, D., & Chen, S. (2006). *Serious Games: Games that Educate, Train, and Inform*. Mason, Ohio, USA: Course Technology.

Mirolli, M., & Parisi, D. (2011). Towards a Vygotskyan cognitive robotics: The role of language as a cognitive tool. *New Ideas in Psychology*, *29*, 298–311.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

Nakada, M., Chen, H., & Terzopoulos, D. (2018). Deep learning of biomimetic visual perception for virtual humans. In C. Grimm, P. Willemsen, J. Kearney, & B. Riecke (Eds.), *15th ACM Symposium on Applied Perception* (pp. 1–8). Vancouver, British Columbia, Canada: ACM.

Nixon, M., & Aguado, A. (2012). *Feature Extraction & Image Processing for Computer Vision*. (3rd ed.). Oxford, UK: Academic Press.

Pfeifer, R., & Bongard, J. (2007). *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, Massachusetts, USA: MIT Press.

Rañó, I. (2014). Results on the convergence of Braitenberg Vehicle 3A. *Artificial Life*, *20*, 223–235.

Rizzo, A., Hartholt, A., Grimani, M., Leeds, A., & Liewer, M. (2014). Virtual reality exposure therapy for combat-related posttraumatic stress disorder. *Computer*, *47*, 31–37.

Rupert, R. D. (2004). Challenges to the hypothesis of extended cognition. *Journal of Philosophy*, *101*, 389–428.

Salomon, R. (1999). Evolving and optimizing Braitenberg vehicles by means of evolution strategies. *International Journal of Smart Engineering System Design*, *2*, 69–77.

Schrodt, F., Kneissler, J., Ehrenfeld, S., & Butz, M. V. (2017). Mario becomes cognitive. *Topics in Cognitive Science*, *9*, 343–373.

Seth, A. K. (2013). Interoceptive inference, emotion, and the embodied self. *Trends in Cognitive Sciences*, *17*, 565–573.

Shapiro, L. A. (2011). *Embodied Cognition*. Abingdon, Oxon, UK: Routledge.

Shapiro, L. A. (Ed.) (2014). *The Routledge Handbook of Embodied Cognition*. New York, New York, USA: Routledge.

Smart, P. R., Engelbrecht, P. C., Braines, D., Strub, M., & Giammanco, C. (2010). The network-extended mind. In D. Verma (Ed.), *Network Science for Military Coalition Operations: Information Extraction and Interaction* (pp. 191–236). Hershey, Pennsylvania, USA: IGI Global.

Smart, P. R., Scutt, T., Sycara, K., & Shadbolt, N. R. (2016). Integrating ACT-R cognitive models with the Unity game engine. In J. O. Turner, M. Nixon, U. Bernardet, & S. DiPaola (Eds.), *Integrating Cognitive Architectures into Virtual Character Design* (pp. 35–64). Hershey, Pennsylvania, USA: IGI Global.

Smart, P. R., & Sycara, K. (2015a). Place recognition and topological map learning in a virtual cognitive robot. In *17th International Conference on Artificial Intelligence* (pp. 3–9). Las Vegas, Nevada, USA: CSREA Press volume Volume 1.

Smart, P. R., & Sycara, K. (2015b). Using a cognitive architecture to control the behaviour of virtual robots. In G. Airenti, B. G. Bara, & G. Sandini (Eds.), *EuroAsianPacific Joint Conference on Cognitive Science* (pp. 447–452). Turin, Italy: CEUR-WS.org volume 1419.

Srinivasan, M. V., Laughlin, S. B., & Dubs, A. (1982). Predictive coding: a fresh view of inhibition in the retina. *Proceedings of the Royal Society B: Biological Sciences*, *216*, 427–459.

Terzopoulos, D., Rabie, T., & Grzeszczuk, R. (1997). Perception and learning in artificial animals. In C. G. Langton, & K. Shimohara (Eds.), *Artificial Life V: Proceedings of the 5th International Workshop on the Synthesis and Simulation of Living Systems* (pp. 346–353). Nara, Japan: MIT Press.

Thornton, C. (2017). Predictive processing simplified: The infotropic machine. *Brain and Cognition*, *112*, 13–24.

Turner, J. O., Nixon, M., Bernardet, U., & DiPaola, S. (Eds.) (2016). *Integrating Cognitive Architectures into Virtual Character Design*. Advances in Computational Intelligence and Robotics. Hershey, Pennsylvania, USA: IGI Global.