Received October 15, 2020, accepted November 23, 2020, date of publication January 11, 2021, date of current version January 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3050882

Using Compression to Discover Interesting Behaviours in a Hybrid Braitenberg Vehicle

NADIM AHMED AND WILLIAM J. TEAHAN
School of Computer Science and Electronic Engineering, Bangor University, Bangor LL57 1UT, U.K.
Corresponding author: William J. Teahan (w.j.teahan@bangor.ac.uk)

ABSTRACT The simple rules that govern the interactions between the different components of a complex system often lead to interesting behaviours that are unexpected. The experiments described in this paper involved creating a variation of a traditional Braitenberg Vehicle, by placing sensors on ten different possible locations on a simulated vehicle, and incorporating an obstacle avoidance behaviour using a subsumption-like architecture, which resulted in different unusual and unexpected behaviours being produced. The vehicle was allowed to explore a simulated environment which contained a single bright light in the centre with walls on the border. By using a novel combination of the Prediction by Partial Matching compression algorithm and *k*-means clustering, interesting emergent behaviours were effectively discovered within a search space of over 10,000 simulations produced from a simple interaction of light and proximity sensors on a vehicle and a single light source. The clustering algorithm discovered five distinct behaviours: circling and spiralling behaviours; interesting behaviours creating intricate rose petal-like structures; behaviours that create simple rose petal-like structures; behaviours with large movements and low complexity; and behaviours with less movement. The novel algorithm demonstrated in this paper has useful potential in the science of complex systems and modelling to help expedite the systematic exploration of a substantial search space of simulations in order to discover interesting behaviours.

INDEX TERMS Entropy coding, robot motion, Braitenberg vehicles, subsumption architecture, prediction by partial matching.

I. INTRODUCTION

This paper expands upon previous work on using compression to find interesting one-dimensional cellular automata [1] by applying the same idea towards a model of a behaviour-based robot based on the Braitenberg Vehicle.

A. MOTIVATION AND BACKGROUND

In this paper, the Braitenberg Vehicle [2] is examined as an example of a robotic system. The idea was to investigate the behaviour of one modified Braitenberg Vehicle, and to see if it is possible to automatically discover if it produces any interesting behaviours.

Braitenberg Vehicles have been heavily researched in many different ways, such as creating robotic fish [3], snake robots [4] or simulating ethical behaviour [5]. Much of the research into Braitenberg Vehicles has used the classic Type 2 and Type 3 Vehicles with architectures that have a strong resemblance to Braitenberg's original diagram of rectangular vehicles with two pronged sensors out of the front of the vehicle [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Huiyu Zhou.

A natural question arises as to what might happen if the sensors were placed on a different part of the vehicle. Another question concerns what would happen to the vehicle's behaviour if a subsumption-like architecture [6] was used to prevent the vehicle from colliding with objects in the environment, thus creating an internal struggle within the vehicle. Would a variation of Braitenberg's Vehicle coalesced with Brooks' subsumption architecture create interesting behaviours?

Finding answers to these questions requires a large number of simulations to be performed, which means that an automated method is needed to search through the different configurations in order to find similar interesting behaviours. Since the number of variations even for simple configurations is essentially unbounded, an automated system would be very useful to work through various possible configurations and resulting simulations automatically to group similar behaviours together in order to find the interesting ones.

A detailed discussion on the concept of interestingness in relation to complex systems analysis has been given before [1], [7]. One of the most commonly used measures to calculate ''interestingness'' [8] is Shannon conditional entropy [9] which can be estimated by using compression. Higher compression code lengths generally indicate a higher level of interestingness; however, this is not always the case. The simulation output produced by a complex system with repetitive patterns will be highly compressible and give a low compression code length, while an output containing substantially random noise will be difficult to compress and therefore give a high compression code length. However, for both these two cases, the simulation output may not be classed as ''interesting'' [10]. Schmidhuber states that for something to be called interesting, it must have an essence of beauty, and therefore he also states that random noise is not ''interesting'' as a result [10].

The novelty introduced in this paper is how compression may be used in conjunction with clustering algorithms to cluster similar types of behaviours of complex systems together. From that it will help researchers discover the different types of behaviours that a complex system may exhibit and help discover surprising or interesting ones.

The rest of this paper is organised as follows. In the next section, Braitenberg Vehicles are introduced, followed by a discussion of the subsumption architecture. The design of a novel Braitenberg Vehicle combined with subsumption architecture is then discussed in detail showing a random selection of simulation outputs produced by the vehicle. The paper then discusses how compression was used to assist in clustering similar behaviours together in order to discover the interesting ones.

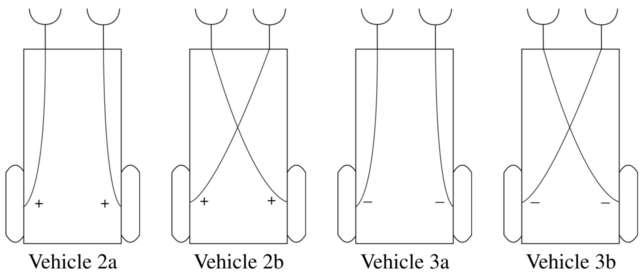
B. RELATED WORK

1) BRAITENBERG VEHICLES

Braitenberg Vehicles are the brainchild of Valentino Braitenberg [2] who used them to describe thought experiments to explore the concept of using simple ''brains'' to create different behaviours. Braitenberg Vehicles are two wheeled vehicles with sensors attached to each of the motors, with sensorimotor coupling where each motor drives a wheel, although they often have a free-moving castor that provides stability to the front of the vehicle. Braitenberg came up with the vehicles to explain how simple rules can create complex behaviours. By changing which sensor is wired to which motor, the vehicle behaves differently when it comes across a stimulus that affects the sensor's ''value''. The other variations that can be carried out with the simple vehicles is by changing the coupling; for example, the connection of each sensor to the motor can be ''positive'' or ''negative''. The four different standard Braitenberg Vehicles are shown in Table 1.

The stimulus received by the sensors is directly proportional to the output from the sensor; thus if a light sensor is placed under bright lights, the output will exceed that of a similar sensor in a dimly lit environment. With Vehicles 2a and 2b, there is a positive coupling between each sensor and the motors, such that a higher output from the sensor results in the attached motors turning the wheels faster.

TABLE 1. Four different Braitenberg Vehicles showing the way sensors are coupled to the motors. The sensors are the two ''cup-like'' devices on the two prongs at the front, while the motor is connected to the wheels at the back.



With Vehicle 2a, if the left sensor detects more light than the right sensor, then the left wheel turns faster than the right one. When the left wheel turns faster, the vehicle will then tend to turn to the right, and so move away from the light, but slowing down as it moves further away. The initial high velocity moving away from the light, slowing down when in the dark, gives the appearance to an observer of ''scared'' behaviour.

With Vehicle 2b, if the left sensor detects more light than the right sensor, due to the way the wires are crossed over, the right wheel will turn faster than the left wheel. This will make the vehicle move towards the light, gaining speed as it gets closer, eventually colliding with it at full speed. It will then rebound (due to the transfer of momentum between the vehicle and the light source) and then the vehicle will subsequently return to hit the light again. To an observer, it may appear that the vehicle is aggressive towards the light as it hits the light at full speed, then returning again for further collisions.

With Vehicles 3a and 3b, the difference in the coupling between the sensors and the motors involves ''switching the sign of the influence from positive to negative'' [2]. Braitenberg clarifies this statement by explaining that the connection is inhibitory, where a strong stimulus slows down the vehicle, while a weak one increases the speed.

In neurology, the brain has neurons that transmit signals to each other by using synapses. When a neuron transmits a signal, a spike is produced, and a change in potential difference (voltage) occurs. For positive changes, the synapse is excitatory, while for negative changes, the synapse is inhibitory [11]. This explains the + signs on vehicles 2a and 2b while vehicles 3a and 3b have − signs. This changes the behaviour such that Vehicle 3a now approaches the source of the stimulus slowly in a way that can be described by an observer that the vehicle ''adores'' the stimulus. In contrast, Vehicle 3b departs the vicinity of the stimulus slowly, such that to an observer it has the characteristics of ''exploring'' the environment.

Braitenberg's first three main vehicles have been explored in detail [12] and replicated as physical hardware [3]. The sensor positions have been well defined, with the only changes being the way the sensors are coupled to the motors. What has not been explored is how the behaviour would change if, instead of the sensors being placed at the front of the vehicle, they are placed instead at different locations around the vehicle.

2) SUBSUMPTION ARCHITECTURE

Designing a robot to act autonomously is a difficult task with many competing requirements. Robots can often have multiple goals, with some that seem to be contradictory. For example, a robot might have to avoid obstacles, while still travelling towards a light. The robot may have multiple sensors of different types, so for example may possess a small number of proximity sensors, light detectors and temperature sensors. The robot has to be robust enough such that if a sensor fails, it can carry on with those sensors that are still functioning, or if the environment changes, it should be able to cope with the new conditions instead of becoming catatonic or behaving in a reckless manner. Lastly, the robot must be extensible; that is, when more sensors are added, then the system can be upgraded easily.

The traditional way of designing robots involves decomposing a robot into a "sense, think, act" structure, where the sensor inputs are read by a perception task, which leads to a modelling task, then planning, which then leads to task execution which finally leads to motor control where the actuators of the robots are manipulated. As Winston [13] mentions, for a long time when using traditional approaches, robots were not able to achieve much at all. Researchers would leave the robot overnight to manoeuvre around a lab. However, by the morning, it would have moved only around 8 metres (25 feet), while avoiding a table [13]. Early robots would often take a long time just to carry out a simple task, so Brooks devised a different way of approaching this issue [13].

In 1986, Brooks described a new and easier approach to designing robots with multiple sensors and goals [6], and named it "subsumption architecture". Instead of having a separate "sense, think, act" structure, the tasks are broken into layers of "competence". The layers (or levels, as Brooks uses these terms interchangeably) start with the absolute minimum requirement of a robot. An example Brooks gives is obstacle avoidance, which runs continuously, unaware of any layers above it and is known as the "zeroth" layer. When the layer's performance has been verified and works as expected, then it is to be left alone, and the next layer is worked upon. The whole idea behind this architecture is that higher levels subsume the lower levels, hence the name "subsumption architecture" [6], [14]. Another advantage of using this architecture is that a working system is built early on in the creation process [6]. Brooks also argues that it is better to let a predetermined priority scheme to determine how to deal with conflicts within the system [15].

To build using subsumption, the first step is to decide what behaviours the robot should be capable of, which then need to be decomposed into different layers. An example of the layers are shown in Table 2. To build a layer, it may be such that only some of the sensor data is required and also only a subset of the behaviours should be implemented in each layer. Different pairs of sensors and tasks are allowed to be used for each layer [16]. Each layer can be built "from a set of small processors" [16] while each processor is designed using a finite state machine [16].

TABLE 2. The bottom four levels of Brooks' subsumption architecture levels taken from [6], [16].

Level	Name	Description
3	Build Map	Build a map of the environment and plan routes.
2	Explore	Explore the environment by heading towards places that are deemed reachable.
1	Wander	Wander around the environment without hitting anything.
0	Avoid	Avoid colliding with objects.

The subsumption architecture has been used to create many different physical robots. The first robot, Allen [17], had sensors on board (sonars and an odometer), while the subsumption architecture was simulated in Lisp off board on a separate machine. Three layers were used for this implementation, while the first layer made the robot avoid all obstacles, the second allowed the robot to wander around in random directions, while the final layer made it seek out distant places with its sonar. With just the first layer, the robot would stay motionless until approached, in which case it would then scurry away. With the second layer added on, the robot would wander around in random directions, but it would also avoid obstacles. With the final layer, an odometer would generate a heading, which often overrode the direction produced by the wander layer. As a result, this was a simple robot that would wander around the environment.

Another robot, called Herbert [17] would wander around the environment entering workspaces and taking empty drink cans from tables while using wall following techniques and obstacle avoidance. It was also able to use its laser triangulation scanner [18] to recognise objects, such as cans of drinks. There was no communication between the layers; instead, each layer was connected to sensors, with the output going to an "arbitration network" [17] which drove the actuators. The laser based scanner was used to drive the robot so then its arm was aligned in front of the drinks can. The arm controller monitored the shaft encoders, so when there was no more wheel movement, the arm movements were commenced which started other behaviours for picking up the can. This kind of set up allows the robot to react to its environment, such that if a can of drink is handed to the robot, or even if it miraculously appears, the robot arm can just pick it up and take it [19]. More recently, the subsumption architecture has also been used to design iRobot's Roomba robotic vacuum cleaner [20].

The subsumption architecture was the first iteration of what would become ''behaviour-based architecture'' [21], and Brooks first adopted behaviour-based architecture in 1994 [22]. In an interview conducted in 2015 he stated that subsumption architecture eventually became behaviour-based programming [23]. In an article exploring the story behind the subsumption architecture, Birrell quotes a programmer who worked with Brooks who said the problem with subsumption architecture was that it was able to do some complex tasks, but it could not progress much further [24]. In an interview at the South by Southwest (SXSW) conference, Brooks stated that the idea behind subsumption architecture was to replicate insects and then move on from there by building higher orders of animals, but the problem was that it did not have the power to be as good as an insect [25]. Another issue is the difficulty in designing the interaction between the different layers and the issue of how to design for emergence [26] or other interesting phenomena. Overcoming these issues have provided a primary motivation behind the work in this paper.

II. DESIGN OF A NOVEL BRAITENBERG VEHICLE

This section looks at a model created in NetLogo [27] that was used to simulate a novel vehicle based on Braitenberg's original vehicles that also adopts a simple subsumption architecture. The first subsection describes the overall design and the research questions that will be investigated by the experimental evaluation that was performed as described below.

A. VARIATIONS OF BRAITENBERG VEHICLES

The first change to the traditional Braitenberg Vehicle in our design includes changing the position of the light sensor from the front of the vehicle to arranging them in different positions around the vehicle. The second change is that instead of just using two light sensors coupled to the motors, two proximity sensors are also coupled to the same motors. This means that each motor will have one light and one proximity sensor coupled to it, with two types of couplings, ''positive'' and ''negative''.

In addition to the changes mentioned above, a subsumption architecture was used for the modified Braitenberg Vehicle with an additional behaviour for obstacle avoidance. The subsumption architecture uses proximity sensors to check for any obstructions, such as lights or the walls, and then prevents the vehicle from crashing into them. The idea is that when the sensor coupling part of the vehicle moves the vehicle towards obstacles, the subsumption layer will prevent it from colliding, providing an interesting possibility for conflict between the different behaviours.

A large number of simulation experiments described below have been performed with this new design. The main purpose of these experiments was to investigate the effectiveness of using an automated system to examine the output and then cluster similar behaviours together with the aim of discovering interesting behaviours.

The specific research questions for these experiments are as follows:

- Is it possible to create complex behaviours with the deviations discussed above to the basic Braitenberg Vehicles design?
- Can interesting behaviours be discovered when the configuration of sensor locations are changed?
- Can automatic methods be created to discover these interesting behaviours?

The next section describes the simulated environment in which the vehicle operates

B. THE ENVIRONMENT

The simulation model was created in NetLogo [27] which simulates the environment when a light is switched on and simulates the vehicle as it moves around the environment. The environment is a grid made up of patches (akin to tiles on a floor), with a width and height of 151 patches. The odd number was chosen so the light could be placed exactly in the centre of the environment with the light emanating symmetrically from the centre of the environment. The lighting was based upon the NetLogo Moth simulation [28] with the only changes being made to the initial brightness of the light and the visual look of the environment on the screen. The light level diminishes the further away it is from the central light source. The light levels on each patch are determined in advance as the light's position is fixed throughout the simulation. These are stored using a patch variable which can then be read directly by the vehicle as it moves around thereby simulating the sensing action the robot performs.

Walls have been placed around the edge of the environment. In the NetLogo simulation, the environment has world wrapping disabled in order to make the simulation more realistic. Each patch also stores the number of times the vehicle visits the patch, which is updated at each tick (NetLogo's time interval).

Once the environment has been created, the next step is to create the vehicle.

C. SENSORS

The simulation model simulates two types of sensors, light and proximity. These sensors can be placed on different positions on the vehicle: 0°, 30°, 60°, 120°, 150°, 180°, 210°, 240°, 300° and 330°, as shown in Figure 1. It is possible to choose angles at a more finer grain such as at every 5°; however, this would substantially increase the search space, so a compromise was set to every 30°, apart from where the wheels of the vehicle are placed.

The sensors also have settings that determine the length and angle of its cone of vision, which represents its sensitivity. To implement the sensor position within NetLogo, the vehicle rotates to the left by the sensor position, then uses the `in-cone` NetLogo keyword to simulate the sensor. The vehicle reads the maximum value of the light luminosity within its sensors' cones of vision. Alternative simulations

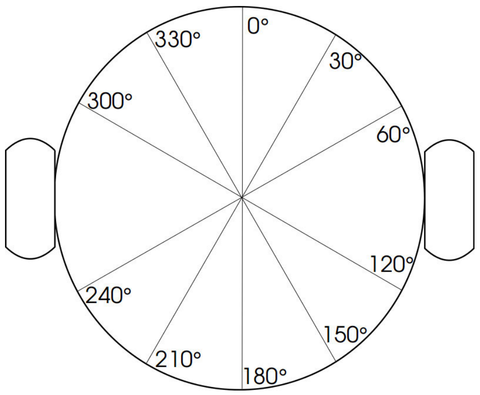


FIGURE 1. The new vehicle showing possible sensor locations in degrees.

using the mean or the minimum values within the vision cone showed the vehicle having the same behaviour albeit needing more ticks to reach the same position than when using the maximum value. After the sensor retrieves its reading, the vehicle returns to its original heading.

In order to closely simulate a Braitenberg Vehicle that would be constructable in real-life, the sensors have similar limitations to commercially available sensors. The light sensors are based on general light dependent resistors (LDR); however, data sheets do not generally give the sensor cone-angle for LDRs, so a value of 30° has been used. A light sensor's cone-angle can be restricted by using a physical cone around the LDR. The proximity sensor's specification is based on those for the HC-SR04 Ultrasonic sensor, which has a cone angle of 15°.

D. SENSORIMOTOR COUPLING

A sensor can be connected to either of the two motors with two types of coupling, the first being positive, and the second being negative. The different couplings indicate the type of connection, with positive being excitatory and negative being inhibitory. When an excitatory connection is used, the higher the sensor reading, the faster the motor will turn, while an inhibitory connection will make the motor move slower when the sensor reading is high. An inhibitory connection can be simulated by using a quotient and adjusting the values as required. However, it is important to anticipate for a division by zero error in the quotient before it can occur by ensuring that the denominator remains larger than zero.

E. USING SUBSUMPTION ARCHITECTURE FOR THE VEHICLE

The vehicle is modelled using a simple subsumption architecture. The vehicle in the simulation has proximity sensors to avoid collisions in order to protect itself from harm, and therefore is at a high priority, so if there is a danger of collision with either the wall or other objects, that behaviour will take precedence. The wall detection routine checks to see if a wall exists within a 45° vision cone in front of the vehicle, with the length of the cone being the same length as the vehicle. If a wall is detected, then the vehicle turns round by 180°. This behaviour can be reproduced in real-life using coloured walls so that a simple colour sensor can be used for detecting the wall.

For other obstacles, the vehicle checks its perimeter, and if an obstacle exists within the perimeter, it turns round by approximately 180° and moves forwards by two steps. When trying to reproduce this in real-life, to turn the vehicle round by 180°, it is possible to have the sensors on a sensor board placed on a servomotor connected to the base of the vehicle, and use a servomotor to twist the sensor board accordingly.

Once the obstacles are no longer a threat, the subsumption layer gives up control.

F. THE HYBRID BRAITENBERG VEHICLE

This subsection gives an outline of how the hybrid Braitenberg Vehicle simulation works. The overall behaviour of the Braitenberg Vehicle is described in Algorithm 1 with the obstacle-avoidance sub-behaviour described in Algorithm 2 and the Braitenberg sub-behaviour described in Algorithm 3. To initialise, the power is cut from both motors, so they are set to zero during the simulation setup. The overall behaviour is to check for any obstacles; if any are detected, then the obstacle avoidance sub-behaviour is activated (which simply involves turning around and moving forward a small distance if the obstacle is not a wall) otherwise the system defaults to the Braitenberg sub-behaviour.

Algorithm 1 Algorithm Describing How the Hybrid Braitenberg Vehicle Simulation Works

```
1 Function BraitenbergSubsumption (coupling)
2     possible-collision ← Detect-Collision
3     if possible-collision = true then
4         CollisionAvoidanceBehaviour
5     else
6         BraitenburgBehaviour(coupling)
```

Algorithm 2 Algorithm Describing the Collision Avoidance Sub-Behaviour Used for the Simulation

```
1 Function CollisionAvoidanceBehaviour
  (possible-collision)
2     if possible-collision = "wall" then
3         turn-around by 180°
4     else
5         turn-around by roughly 180°
6         move-forward by 2 steps
```

For the Braitenberg sub-behaviour, the vehicle checks the sensor readings and transfers the reading to the motor coupled to the sensor. If the coupling is positive, the motor sensor reading is added to the current velocity of the motor. With a negative coupling, a quotient $\frac{v}{R}$ is used where v is the velocity of the vehicle and R is the sensor reading. Using this formula

Algorithm 3 Algorithm Describing the Braitenberg Sub-Behaviour Used for the Simulation. Lines 9 and 10 Are Explained in Detail in the Text

```
1  Function BraitenbergBehaviour (coupling)
2      Read-Sensors
3      if coupling = positive then
4          left-motor-speed ← left-motor-speed + reading-from-sensor-coupled-to-left-motor
5          right-motor-speed ← right-motor-speed + reading-from-sensor-coupled-to-right-motor
6      else
7          left-motor-speed ← left-motor-speed + (500/ reading-from-sensor-coupled-to-left-motor)
8          right-motor-speed ← right-motor-speed + (500 / reading-from-sensor-coupled-to-right-motor)
9      turn-left right-motor-speed
10     turn-right left-motor-speed
11     move-forward by mean (left-motor-speed + right-motor-speed) / highest (light value)
```

allows the vehicle to slow down when the light levels are high, and to speed up when the levels are low. While experimenting with the simulation, it was discovered that for the negative coupling, a large value for v, such as 500, gave realistic speeds for the simulation. If the numerator was too small, the vehicle speed would be exceedingly slow, whereas if the numerator was too large, the vehicle would travel too quickly. As the use of the quotient for negative coupling involves a division, to avoid division-by-zero errors and undesirable behaviour when R is less than 1, the lowest value that was allowed for R was set to 1.

After the motor speeds have been assigned, the vehicle turns according to the ratio between the left and right motor, so if the left motor has a higher ratio, the vehicle will turn more to the left. It may seem counter-intuitive that the vehicle turns left by the amount the right motor spins, but this can be explained by the fact that when a vehicle's right wheel turns faster than the left, the vehicle will move left. The order the vehicle turns does not affect the overall behaviour and in any case does not matter, as it is commutative, so $-45°$, followed by $+5°$ is the same as $+5°$ followed by $-45°$. The final task is for the vehicle to proceed forward, which is determined by how fast the motors are spinning, so the faster they spin, the faster the vehicle moves.

III. THE EXPERIMENTAL SET-UP

The experimental simulations performed examine the behaviour of the hybrid Braitenberg Vehicle combined with a simple subsumption-like architecture mentioned above. In order to explore different behaviours, several variables are used: the positions of the sensors; the type of sensors used; the sensitivity of the sensor, and the coupling. The sensor sensitivity is determined by the angle of the vision cone and its length, while the coupling identifies which motor the sensor is connected to, and how the connection is made.

TABLE 3. Vehicle sensor variables set for each sensor used in the vehicle simulation.

Variable	Available Values
Sensor type	light, proximity
Sensor location	0, 30, 60, 120, 150, 180, 210, 240, 300, 330
Light Sensor angle	30
Proximity Sensor angle	15
Sensor range	55
Motor	left, right
Coupling	+, −

Two light and two proximity sensors were used in the simulations, where there was one of each sensor connected to each of the two wheels. The experimental variables were set to appropriate values as listed in Table 3. The sensor type variable defines whether the sensor used is a light or proximity sensor, with both being used for the different simulation variations. The sensor location variable shows where the sensor is placed on the vehicle (from a birds-eye perspective, centred on the vehicle as shown in Figure 1). The sensor angle variable and range variable define the cone of vision for the sensor. The motor variable refers to which wheel and motor the sensor is connected to, while the coupling variable indicates how the sensor readings are interpreted by the motor. Using four sensors, the various combinations of the selected values for the variables results in a total of 10,080 simulations: 10 possible locations $\times$ 9 remaining locations $\times$ 8 remaining locations $\times$ 7 remaining locations $\times$ 2 different couplings = 10080 simulations.

Each patch in the NetLogo environment keeps a counter of the number of times a vehicle visits. Each time a vehicle visits the patch, the visit counter is incremented by 1. This means that for those simulations where the vehicle visits many areas of the environment, there will be many visit counters that are greater than zero, whereas with those simulations where the vehicle remains fairly stationary, then there will be large numbers of zero visit counters. After the simulation has completed its run of 5,000 ticks (NetLogo's unit of time), the list of visit counters are then exported for processing.

A. RUNNING THE SIMULATION

As stated, each time the vehicle visits a patch in the environment, the visit counter for that patch is incremented by 1. Also mentioned before, each simulation used a maximum of two light and two proximity sensors. All the vehicles used proximity sensors within the subsumption architecture to prevent the vehicle from crashing into the light or the walls. The vehicle started in the same position, near the top centre of the environment at location (0, 50) to reduce the chance of a left/right bias of the vehicle. The vehicle's pen had been enabled in order to record the vehicle movements throughout the 5,000 ticks for the separate output image produced.

Table 5 illustrates the variety of behaviour that was produced by the different hybrid Braitenberg simulations with 40 randomly chosen outputs being shown from the pool of 10,080. The outputs shown display the vehicle at the end of each simulation along with the path taken by the vehicle as recorded by it's pen. To make identification of each simulation easier for this paper, a simulation number has been allocated to each simulation shown in Table 4.

TABLE 4. The sensor positions and coupling for all the simulations shown in this paper.

	Left Motor		Right Motor		
Simulation Number	Light Angle	Proximity Angle	Light Angle	Proximity Angle	Coupling
1	0	240	60	30	+
2	30	330	0	150	−
3	30	0	180	150	−
4	30	150	180	210	+
5	30	60	300	0	+
6	60	240	30	120	−
7	60	330	120	240	+
8	60	330	180	240	+
9	60	240	210	0	+
10	60	300	210	120	+
11	120	240	0	330	−
12	120	300	30	330	−
13	120	30	60	240	+
14	150	300	30	330	−
15	150	240	60	180	+
16	150	330	60	0	−
17	150	330	210	120	+
18	150	60	300	240	−
19	150	210	330	300	+
20	180	210	120	330	+
21	210	240	30	0	−
22	210	0	60	300	+
23	210	300	150	180	−
24	240	210	30	180	+
25	240	150	60	210	−
26	240	30	120	150	−
27	240	330	180	30	−
28	240	300	210	180	−
29	300	180	60	150	−
30	300	240	60	150	+
31	300	60	120	330	−
32	300	330	120	30	+
33	300	0	180	120	+
34	300	330	180	240	−
35	300	0	240	210	−
36	300	210	240	0	+
37	330	120	150	180	+
38	330	210	150	300	+
39	330	120	240	0	−
40	330	150	240	60	+

After the simulation has finished, all the patch visit counters for that simulation are exported for analysis. After the completion of all simulations, a compression-based analysis of robotic behaviour was carried out as described in the next sub-section.

B. COMPRESSION-BASED ANALYSIS USING PREDICTION BY PARTIAL MATCHING

Compression code length to used to help determine whether the output observed is interesting or not for the simulations in the previous sub-section. Compression code length is a measure of how many bits are required for all symbols to be represented in the compressed data, so a lower compression code length indicates better compression has been achieved. This sub-section explains how the compression code lengths were calculated. The compression-based analysis of the simulation output uses the Tawa Toolkit [29] to calculate the compression code lengths, using an adaptive statistical compression algorithm called Prediction by Partial Matching (PPM), which has been found to be one of the best performing text compression algorithms [30]. Adaptive statistical-based compression carries out two processes: modelling and coding. The first process builds a table of probabilities of all the previously encountered symbols in order to anticipate what the next symbol will be while the second coding process uses the probability distribution created from the model to encode the symbol.

PPM is a Markov based system where the context (last few symbols in the input stream) is used to forecast what the next symbol will be. The order of a model is determined by the number of symbols in the context, so a context length of 1 means that PPM will use a single symbol to predict the upcoming symbol. For the hybrid Braitenberg vehicle output, a context of length 1 was found to give superior compressed output. During the experimental process it was observed that better compression of the output results in a superior clustering of behaviours.

The probability distribution models for each order are constructed as each symbol is encoded, then they are blended into a single model by using an escape mechanism. The escape mechanism will use the highest order that has been chosen to predict the upcoming symbol; so if the highest order is set to 1, then the PPM modelling process will first use order 1 to check if the symbol has been encountered before. If the symbol has not been encountered before, then the PPM escape mechanism will switch to a lower order model. As mentioned earlier, a maximum fixed order of 1 was used for the compression-based analysis.

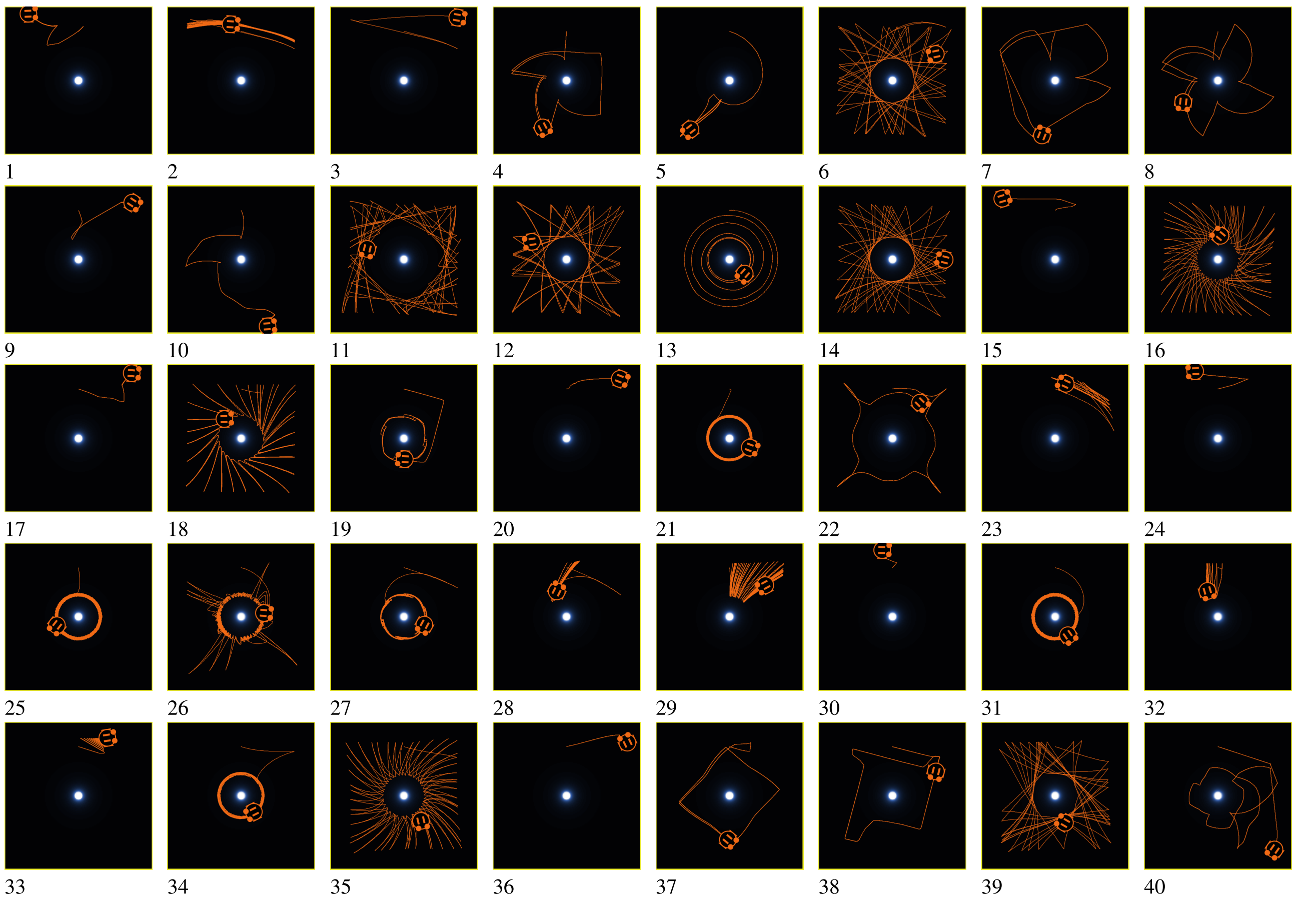
Equation 1 shows how the compression code length h for encoding a symbol s_i in bits is calculated using an order 1 PPM model.

$$h(s_i) = -log_2 P(s_i|s_{i-1}). \tag{1}$$

(Note that for the purposes of the simulation analysis described in this paper, the coding step which is usually performed for compression purposes is unnecessary—the Tawa toolkit allows modelling without coding, with reliable compression code lengths estimated using Equation 1, since the standard coder used for PPM, arithmetic coding, is known to result in compressed output close to this theoretical minimum.)

When using PPM for compression, the user sets the maximum order of the model, k. When a symbol that has not been encountered before is detected, an escape symbol is encoded which results in PPM decrementing the context order by 1 until the symbol is found in the model. If the context order reaches -1 due to this escaping or "back-off" mechanism,

TABLE 5. Output produced by 40 randomly chosen hybrid Braitenberg Vehicle simulations are shown along with their assigned simulation number.



the symbol is then given a probability of $\frac{1}{|A|}$, where A refers to the alphabet size. Since the experimental simulations run for 5000 ticks, the maximum number of visits possible for each patch of the environment is 5000, so the alphabet size has been set to 5001.

PPM uses "full exclusion" where when the escape mechanism is activated, all symbols previously predicted by higher orders are excluded from the probability estimations for each order. Moffat [31] introduced the novel idea of "update exclusions" to PPM, which was unique to PPM, where the symbol count for each context is incremented only when it has not been predicted by higher order contexts.

Two main variations of the PPM algorithm exist which use contrasting equations for calculating the escape and symbol probabilities: PPMC (method C) and PPMD (method D). These calculate the escape probability, e and the probability of a symbol occurring, $p(s)$, with the PPMC calculations shown in Equations 2 and 3, while the PPMD calculations are shown in Equations 4 and 5:

$$e_{\text{PPMC}} = \frac{t}{n+t} \tag{2}$$

$$p(s)_{\text{PPMC}} = \frac{c(s)}{n+t} \tag{3}$$

$$e_{\text{PPMD}} = \frac{t}{2n} \tag{4}$$

$$p(s)_{\text{PPMD}} = \frac{2c(s)-1}{2n} \tag{5}$$

where:

- t is the number of types (unique symbols) that follow the context;
- n is the number of times a context has occurred; and
- $c(s)$ is the number of times a context was followed by the symbol s.

PPMC estimates each symbol's probability by using its frequency of occurrence and assigns the number of types t to the escape count in order to estimate the probability that an escape will occur when a forthcoming symbol is absent from the context. When a previously seen symbol is confronted, PPMD increases the symbol counter by 2, but when symbols have not been observed before, it increments the escape count by 1 and allocates an initial symbol count of 1. In many experiments, including the prior work on cellular automata, PPMD often outperforms PPMC for compression, and for this reason, PPMD was used to compress the output data produced by the hybrid Braitenberg vehicle.

Table 7 shows the output of the trie data structure that was constructed by the Tawa toolkit after reading the first 50 rows of the output (i.e. after processing 7550 visit counters as they appear in the environment reading from top to bottom and from left to right). A trie is a type of search tree where the nodes are symbols found in the search keys and the paths down through the tree represent unique prefixes in those keys. A trie data structure is often used in PPM implementations to

TABLE 6. Placement of sensors for the hybrid Braitenberg vehicle used to demonstrate the workings of PPM compression.

Sensor	Angle	Coupling	Sensor	Angle	Coupling	Motor
Light	330°	+	Proximity	180°	+	Left
Light	60°	+	Proximity	0°	+	Right

TABLE 7. Dump of dynamic PPM trie built from the first 50 rows of the output visit count data produced by the hybrid Braitenberg Vehicle described in Table 6 showing the counts for both order 1 models PPMC′ (PPM using escape method C without update exclusions) and PPMD (PPM using escape method D with update exclusions, i.e. standard PPMD).

Depth	PPMC′ counts	PPMD counts	Path to trie node	Depth	PPMC′ counts	PPMD counts	Path to trie node
0	7550	12	.	2	175	349	.→2→0
1	6531	11	.→0	2	60	119	.→2→1
2	5929	11857	.→0→0	2	34	67	.→2→2
2	398	795	.→0→1	2	4	7	.→2→3
2	178	355	.→0→2	2	3	5	.→2→4
2	17	33	.→0→3	1	33	7	.→3
2	8	15	.→0→4	2	24	47	.→3→0
1	698	7	.→1	2	5	9	.→3→1
2	392	783	.→1→0	2	3	5	.→3→2
2	235	469	.→1→1	2			.→3→3
2	60	119	.→1→2	1	11	3	.→4
2	11	21	.→1→3	2	10	19	.→4→0
1	276	9	.→2	2			.→4→2

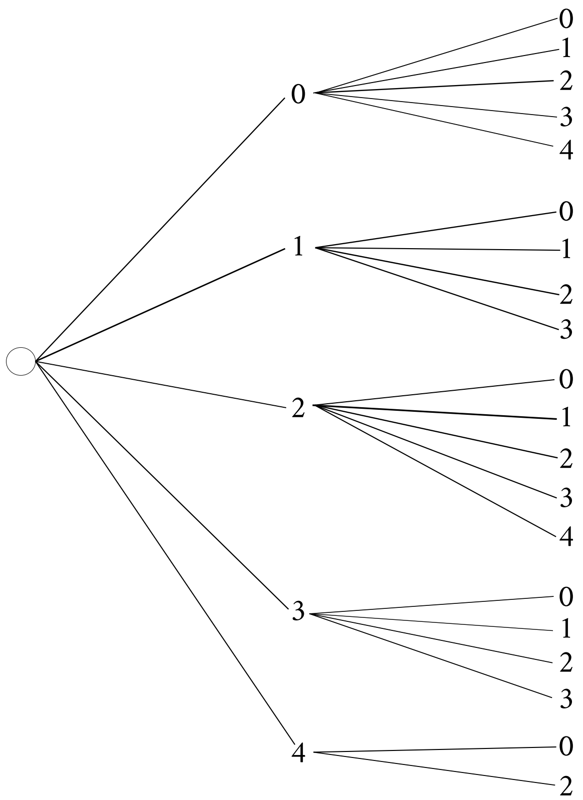


FIGURE 2. The trie structure retrieved from the dump of the dynamic PPM trie shown in Table 7 for the model configuration described in Table 6.

efficiently store the frequency counts for each of the contexts and their predictions. The depth column in the table shows the depth of the trie data structure, and since a PPM model of order 1 is being used, the maximum depth is 2, with a depth of 0 for the root node of the trie. The table shows the PPMC′ (PPM using escape method C without update exclusions) and PPMD counts calculated by these algorithms for the values given along with the path to the trie node (with "." presenting the root node of the trie, and the path to child nodes, labelled with their respective symbols, shown by using →). The relationship between the PPMC′ and PPMD counts for the path to the trie node with a depth of 2 can be summarised by the numerators of Equations 3 and 5. For PPMC, the count is represented by $c(s)$, whereas for PPMD, it is $2c(s) - 1$. For example, the PPMC′ count in the table for the ". → 0 → 1" path is 398, while for PPMD, it is 795, which is equal to $2 \times 398 - 1$. This reflects that there were 398 occasions when a "1" visit count followed a "0" visit count in the output data for this simulation. Figure 2 shows the corresponding trie structure with the root trie node on the left, which has a depth of 0, with the depth increasing towards the right of the diagram.

IV. COMPRESSION-BASED ANALYSIS OF ROBOTIC BEHAVIOUR

After the simulations were completed, the first task was to find out which PPM order gave the best compression by compressing the output from the simulation with PPM and collecting the code length values for all the simulations, and then calculating the mean. As the data produced is the number of visits each patch receives over the whole simulation period of 5,000 ticks, it is impossible for a patch to receive in excess of 5,000 visits by one vehicle, and thus the alphabet size was set to 5,001. It was found that the best overall compression was achieved by using an order model 1, with higher orders resulting in higher code length values (i.e. less compression).

TABLE 8. The Average Silhouette Width for the different number of clusters. The highest Average Silhouette Width occurred when 5 clusters were created.

Number of Clusters	Average Silhouette Width	Number of Clusters	Average Silhouette Width
2	0.1816	5	0.2925
3	0.2154	6	0.2873
4	0.2186	7	0.2635

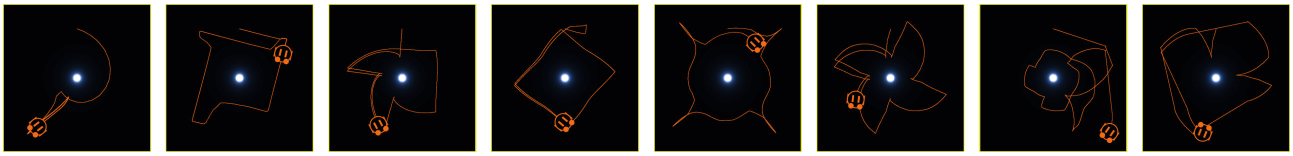
For each simulation, the code length of each row of the environment is calculated and is then exported to calculate the first difference of cross entropy as below.

A. FIRST DIFFERENCE OF CROSS ENTROPY CALCULATION

The next stage was to create the first difference of cross entropy values by using Tawa Toolkit's `codelength` application which calculates the code length of the data stream after compressing with the PPM algorithm. The cross-entropy $H(r)$ for each row r is calculated using Equation 6, with the code length required to encode each row divided by the

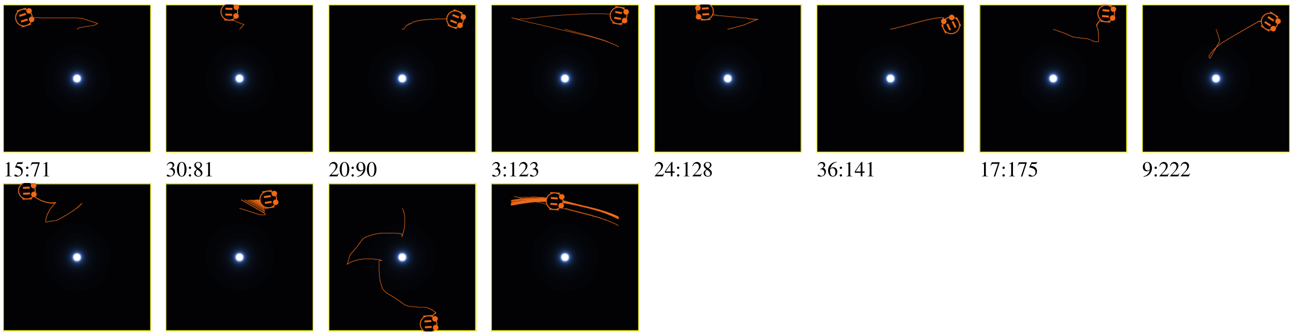
TABLE 9. Clustered output of the images shown in Table 5 with each cluster arranged in code length order. The simulation number followed by the compression code length is shown below each of the images.

Cluster A



5:527 38:628 4:639 37:704 22:717 8:776 40:786 7:792

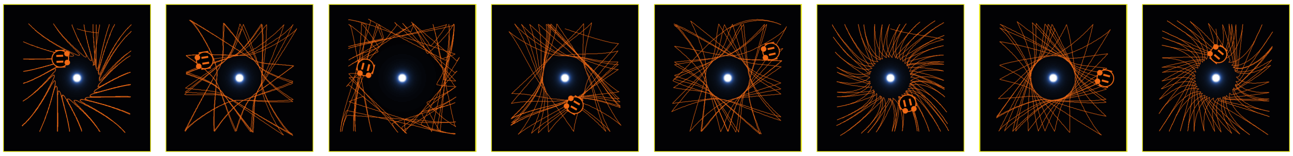
Cluster B



15:71 30:81 20:90 3:123 24:128 36:141 17:175 9:222

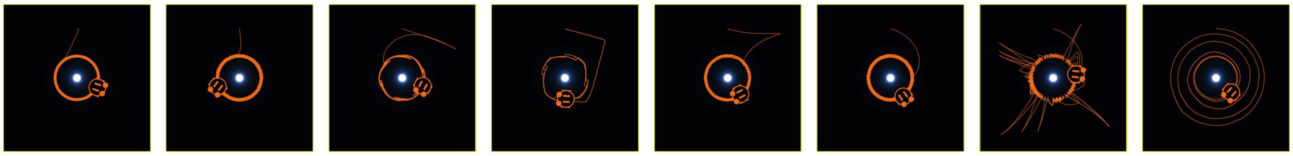
1:225 33:287 10:355 2:430

Cluster C



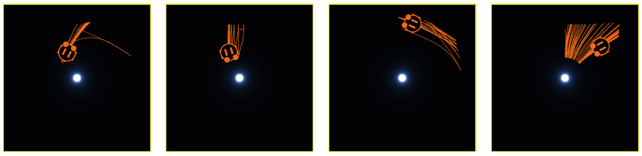
18:1481 12:1790 11:1857 39:1925 6:1936 35:1938 14:1978 16:2025

Cluster D



21:491 25:540 27:557 19:566 34:571 31:582 26:1156 13:1172

Cluster E



28:393 32:427 23:500 29:898

number of symbols c in each row, which is 151 in this case:

$$H(r) = \frac{1}{c} \sum_{i=cr}^{c(r+1)-1} h_i(r). \quad (6)$$

The `codelength` application can return the code length at any point of data compression, so the code lengths were retrieved for each row of patches. These were then subtracted from each row, dividing it by the number of patches per row to calculate the "first difference in cross entropy" for each row r, $\Delta H(r)$, defined as follows:

$$\Delta H(r) = H(r) - H(r-1). \quad (7)$$

Each simulation resulted in 150 first difference of cross entropy values, which were then used to cluster similar behaviours together.

V. CLUSTERING ROBOT MOTION BEHAVIOUR

As mentioned earlier in section I-A, the easiest way to find interesting behaviours was by assuming that more interesting behaviour results in higher entropy values. However, this was not sufficient in distinguishing between the different behaviours that occur, and for that, a clustering step was required. To check the performance of a cluster, and to determine the best number of clusters to use, the Average Silhouette Width [32] metric was used because a gold standard was not available, since Average Silhouette Width has been found to be suitable in these cases [33]. An Average Silhouette Width of 1 indicates perfect clustering, while 0 indicates that in some clusters, a cluster member is on the boundary of a different cluster. When clustering the first difference of cross entropy data, it was discovered that by using k-means clustering [34], the highest Average Silhouette Width was obtained, 0.2925, with four clusters, which indicates that for some of the clusters, there are some members that are borderline members of different clusters.

The output from all the 10,080 simulations were clustered using this approach. In order to illustrate the clusters that were produced, the same 40 randomly chosen outputs shown in Table 5 have been arranged in Table 9 into the clusters they were allocated by this clustering operation. The images within each of the five clusters shown in the table have been sorted in ascending order of compression code length. A visual analysis of these clusters shows that the clustering algorithm has been very effective at placing the different simulation outputs into clusters with similar behaviour, at least for the sample of 40 randomly chosen simulations shown here. A similar result is achieved if a different randomly chosen sample is used instead.

A. FINDING INTERESTING BEHAVIOURS

When similar behaviours are clustered using the compression-based analysis as described above, it is easier to get a better understanding for the range of behaviours that is produced by the system, as shown in Table 9. It is evident that cluster A has simulations which caused the vehicle to traverse much of the environment, with more complex patterns towards the end of the table row, while the vehicles in the simulations in cluster B on the other hand did not explore much of the environment at all. The simulations in cluster C have complex intricate patterns in their output, whereas those in cluster D have a circular pattern with some intricacies, especially simulation number 7412. In contrast, the output from the simulations in cluster E also show that the vehicle has not explored much of the environment.

Note that the compression code lengths of the output in clusters A, C and D are in general much higher than those in clusters B and most of cluster E. It was found that those simulations producing output with intricate patterns and covering large areas of the environment tended to have compression code lengths greater than 1000 bits/symbol. The sensor locations and their coupling for the simulations from the selection mentioned in Table 9 which produced compression code lengths that are greater than 1000 bits/symbol are shown in Table 10.

The experimental results show that changing the position of the sensors on the vehicle and adding a subsumption architecture has produced some unusual and interesting behaviours. When proximity sensors were attached to the motors, some of the behaviours observed were so surprising that it was necessary to manually enter the parameters and observe the vehicle in the simulation travelling along the same path to confirm the result. Clearly, this is an example where emergent behaviour has arisen from simple rules creating complex patterns resembling intricate petal-like structures (especially in cluster C) and many other different patterns. Some of these patterns may not have been discovered if it was not for the algorithm which saved the need to laboriously manually examine over 10,000 images to find which of the patterns were interesting.

TABLE 10. The sensor positions and coupling for the simulations where the compression code length was greater than 1000 bits/symbol.

	Left Motor		Right Motor			
Simulation Number	Light Angle	Proximity Angle	Light Angle	Proximity Angle	Coupling	Compression Code length
6	60	240	30	120	−	1936
11	120	240	0	330	−	1857
12	120	300	30	330	−	1790
13	120	30	60	240	+	1172
14	150	300	30	330	−	1978
16	150	330	60	0	−	2025
18	150	60	300	240	−	1481
26	240	30	120	150	−	1156
35	300	0	240	210	−	1938
39	330	120	240	0	−	1925

VI. FUTURE WORK

The current system is being adapted for a real life modified Braitenberg Vehicle which will explore how the simulation can be moved into the real world. It will allow the validation of the findings from the simulations to see whether the "interesting" configurations are able to produce similar results in real life despite the effects of friction and other physical effects that have not been factored into the simulations.

VII. CONCLUSION

In this paper, a modified variant of the Braitenberg Vehicle that was combined with obstacle avoidance behaviour within a subsumption architecture was discussed, where both light sensors were used accompanied by proximity sensors. The subsumption architecture was there to prevent the vehicle from crashing into the wall or other objects within the environment, and was chosen to cause an internal conflict to the vehicle in some situations. By changing the location of the light and proximity sensors on the vehicle, different behaviours occurred from this conflict creating intricate patterns around the light source in some of the simulations. The output was then compressed using PPM compression and the difference in cross entropy between the rows estimated using PPM was then calculated. From this data, it was possible to run the k-means clustering algorithm to cluster similar behaviours together. This then allowed interesting behaviours to be effectively discovered.

This paper examined a synthetic system that was not geared towards bio-inspired complex systems. However, the approach used for clustering behaviours of a complex system has far-reaching capabilities to discover surprising behaviours in different complex systems including bio-inspired ones.

REFERENCES

[1] N. Ahmed and W. J. Teahan, "Using compression to find interesting one-dimensional cellular automata," in *Complex Intelligent Systems*. Springer, Sep. 2019.

[2] V. Braitenberg, *Vehicles*. Cambridge, MA, USA: MIT Press, 1984.

[3] T. Salumae, I. Rano, O. Akanyeti, and M. Kruusmaa, "Against the flow: A braitenberg controller for a fish robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 4210–4215.

[4] I. Rano, A. G. Eguiluz, and F. Sanfilippo, "Bridging the gap between bio-inspired steering and locomotion: A Braitenberg 3a Snake robot," in *Proc. 15th Int. Conf. Control, Autom., Robot. Vis. (ICARCV)*, Nov. 2018, pp. 1394–1399.

[5] C. J. Headleand and W. Teahan, "Towards ethical robots: Revisiting Braitenberg's vehicles," in *Proc. SAI Comput. Conf. (SAI)*, Jul. 2016, pp. 469–477.

[6] R. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. 2, no. 1, pp. 14–23, 1986.

[7] N. Ahmed, "Discovering interesting behaviours in complex systems," Ph.D. dissertation, School Comput. Sci. Electron. Eng., Bangor Univ., Bangor, U.K., Nov. 2019.

[8] P. Hall and S. C. Morton, "On the estimation of entropy," *Ann. Inst. Statist. Math.*, vol. 45, no. 1, pp. 69–88, 1993.

[9] J. Blanchard, F. Guillet, R. Gras, and H. Briand, "Using information-theoretic measures to assess association rule interestingness," in *Proc. 5th IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2005, p. 8.

[10] J. Schmidhuber, "Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes," in *Anticipatory Behavior in Adaptive Learning Systems*. Berlin, Germany: Springer, 2009, pp. 48–76.

[11] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[12] I. Rano, "A model and formal analysis of braitenberg vehicles 2 and 3," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 910–915.

[13] P. H. Winston. (2010). *Lecture 19: Architectures: GPS, SOAR, Subsumption, Society of Mind*. Accessed: Jan. 1, 2019. [Online]. Available: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-19-architectures-gps-soar-subsumption-society-of-mind/

[14] R. Pfeifer and C. Scheier, *Understanding Intelligence*, 1st ed. Cambridge, MA, USA: MIT Press, 2001.

[15] R. A. Brooks, "How to build complete creatures rather than isolated cognitive simulators," in *Architectures for Intelligence*, K. V. Lehn, Ed. East Sussex, U.K.: Psychology Press, 2014.

[16] R. Brooks, "A hardware retargetable distributed layered architecture for mobile robot control," in *Proc. IEEE Int. Conf. Robot. Autom.*, Mar. 1987, pp. 106–110.

[17] R. A. Brooks, "Elephants Don't Play Chess," *Robot. Auto. Syst.*, vol. 6, nos. 1–2, pp. 3–15, Jun. 1990.

[18] R. A. Brooks, J. H. Connell, and P. Ning, "Herbert: A Second Generation Mobile Robot," Cambridge, MA, USA: MIT Press, Tech. Rep. AIM-1016, 1988.

[19] J. Connell. (1987). *Soda Can Collecting Robot*. Accessed: May 10, 2019. [Online]. Available: https://www.youtube.com/watch?v=YtNKuwiVYm0

[20] T. E. Kurt, *Hacking Roomba*. Hoboken, NJ, USA: Wiley, 2006.

[21] C. Flanagan, D. Toal, and M. Leyden, "Subsumption and fuzzy-logic, experiments in behavior-based control of mobile robots," *Int. J. Smart Eng. Syst. Des.*, vol. 5, no. 3, pp. 161–175, Jan. 2003.

[22] R. A. Brooks and L. A. Stein, "Building brains for bodies," *Auto. Robot.*, vol. 1, no. 1, pp. 7–25, 1994.

[23] Create. (2018). *Rethink Robotics Revisited: A Look Back at the Collaborative Robots Pioneer*. Accessed: Apr. 1, 2020. [Online]. Available: https://www.createdigital.org.au/rethink-robotics-collaborative-robots-pioneer/

[24] S. Birrell. (Jun. 7, 2016) *Robot Mind or Robot Body: Whatever Happened to the Subsumption Architecture*. Accessed: Apr. 1, 2020. [Online]. Available: http://www.artificialhumancompanions.com/robot-mind-robot-body-whatever-happened-subsumption-architecture/

[25] SXSW Interactive. (Mar. 17, 2016). *Rodney Brooks in Conversation with Nick Thompson*. Accessed: Apr. 1, 2020. [Online]. Available: https://www.youtube.com/ watch?v=k3_AFg2922Y

[26] R. Pfeifer and J. Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence*, 1st ed. Cambridge, MA, USA: MIT Press, 2006.

[27] U. Wilensky. (1999). *NetLogo*. Accessed: Jun. 1, 2019. [Online]. Available: http://ccl.northwestern.edu/netlogo/

[28] U. Wilensky. (2005). *Netlogo Moths Model*. Accessed: Nov. 1, 2019. [Online]. Available: https://ccl.northwestern.edu/netlogo/models/Moths

[29] W. Teahan, "A compression-based toolkit for modelling and processing natural language text," *Information*, vol. 9, no. 12, p. 294, Nov. 2018.

[30] M. Mahoney. (Nov. 2016). *Large Text Compression Benchmark*. Accessed: Jan. 15, 2018. [Online]. Available: http://mattmahoney.net/dc/text.html

[31] A. Moffat, "Implementing the PPM data compression scheme," *IEEE Trans. Commun.*, vol. 38, no. 11, pp. 1917–1921, Nov. 1990.

[32] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, Nov. 1987.

[33] S. S. im Walde, "Experiments on the automatic induction of german semantic verb classes," *Comput. Linguistics*, vol. 32, no. 2, pp. 159–194, Jun. 2006.

[34] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-means clustering algorithm," *Appl. Statist.*, vol. 28, no. 1, pp. 100–108, 1979.

NADIM AHMED received the Ph.D. degree in computer science from Bangor University, Wales, U.K., in 2019. He has worked as a Computer Science Lecturer with Soran University, Erbil, Iraq. His research interests include artificial intelligence and robotics.

WILLIAM J. TEAHAN received the Ph.D. degree in applying text compression models to the problem of modeling English text from the University of Waikato, in 1998. He was a Research Assistant with the Machine Learning and Digital Libraries Labs, University of Waikato, New Zealand, in 1998. From 1999 to 2000, he was a Research Fellow with the Information Retrieval Group, The Robert Gordon University in Aberdeen, Scotland, under Prof. David Harper. He was an Invited Researcher with the Information Theory Department, Lund University, Sweden, in 1999. He is currently a Senior Lecturer with the School of Computer Science and Electronic Engineering, Bangor University. His work involves research into artificial intelligence and intelligent agents. Ongoing research has also specifically focused on applying text compression-based language models to natural language processing (NLP), information retrieval (IR), and text mining.