August 2017

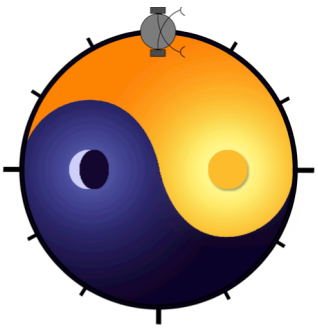
University of Sussex

Dissertation for
MSc Intelligent Systems

Towards Sleep-Wake Cycles with environment-modelling robotics.

Using NARX neural networks in a Braitenberg Vehicle simulation

By: Lucas Rijllart



Project supervisor: Chris Buckley

MSc Intelligent Systems
Informatics department
School of Engineering and Informatics

Abstract

Robotics has always faced the issue of adaptability to new environments and new situations. To create an adaptive behaviour, we make the agent create its own model of the environment. We design Sleep-Wake cycles to structure the behaviour of the agent into learning, then problem solving. By using a model of the environment to predict future movement, we can simulate all robotic trial-and-error, decreasing the amount of real time needed to get a solution. Results show one cycle performs nearly as well as the optimal answer, with a generalised behaviour generated from a short prediction, and solving the problem in less time than trial-and-error.

Preface

This report is submitted as part requirement for the degree of *MSc Intelligent Systems* at the University of Sussex. It is the product of my own labour except where indicated in the text.

Lucas Rijllart

29th of August 2017

Acknowledgements

I would like to thank Chris Buckley for his guidance and support throughout this project, Stathis Kagioulis for his help and motivation.

Table of Contents

1. Introduction

In the event of limb damage, humans and animals can correct their movements by compensating for the damaged mechanic. Unlike animals, machines lack the ability to adapt their behaviour in the event of an unexpected change, or predict new outcomes in an unseen territory. Robots are given precise, unbreakable rules that do not change over time, and they do not try to predict the outcome of an action, as it should never change. To build a more intelligent machine, we try to simulate an understanding of the environment so it can generate new behaviours.

Many modern machines are given increasingly complex models of their environment and the complicated equations of behaviours. This research will focus on the opposite approach: giving the agents a simpler model of their environment in the hope they can create their own simple behaviours. The model will not be humanly understandable, nor will it visually depict the environment. This model will be created through a NARX neural network. This recurrent dynamical neural network will serve as a model for the robot to predict future behaviour.

To test a new behaviour, a robot would have to execute the new actions in the real world, and observe the result. This method would take an unreasonable amount of time to discover a new behaviour. We are therefore looking to shift all real-time actions to offline processing, where the robot can use its understanding of the world to predict its next behaviour.

We decide to name the active movement of the robot a "wake" stage. To create an opportunity to do processing, we incorporate a "sleep" stage that can be considered like "thinking". The robot can then alternate between wake stage and sleep stage to minimise real movement.

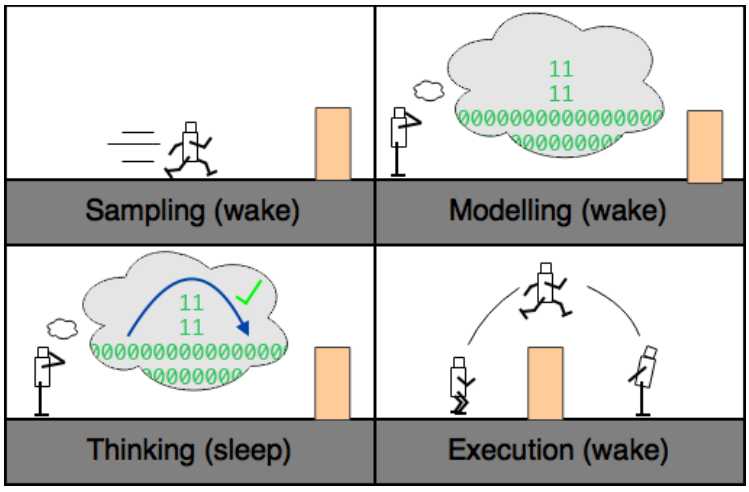


Figure 1-1: Caricature of a robot sampling data from the environment, creating a model of, then using the model to predict jumping over an obstacle before executing the action.

2. Background

In this section, we will explore the literature revolving around our research to set the basis of the project.

2.1. Self-models

The aim of Bongard's research is to create a robot capable of modelling itself to assess any damage done to the body [1]. Their research consists of making a robot create models of its body and pitch them against each other to asses which is more accurate, and then test the predicted behaviour of the model onto the robot's physical body. Our research is closely tied to theirs in the way they use cycles to explore, model, and perform actions. Our common goal is to create an autonomous robot able to use models to better control its behaviour [2-4].

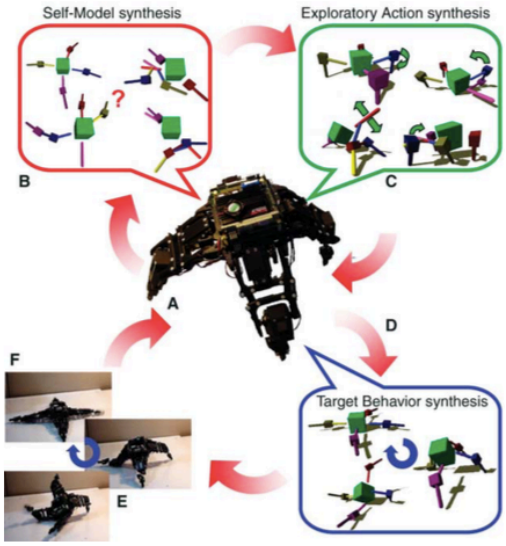


Fig. 1. Outline of the algorithm. The robot continuously cycles through action execution. (**A** and **B**) Self-model synthesis. The robot physically performs an action (A). Initially, this action is random; later, it is the best action found in (C). The robot then generates several self-models to match sensor data collected while performing previous actions (B). It does not know which model is correct. (**C**) Exploratory action synthesis. The robot generates several possible actions that disambiguate competing self-models. (**D**) Target behavior synthesis. After several cycles of (A) to (C), the currently best model is used to generate locomotion sequences through optimization. (**E**) The best locomotion sequence is executed by the physical device. (**F**) The cycle continues at step (B) to further refine models or at step (D) to create new behaviors.

Figure 2-1 Algorithm used in Bongard's Resilient Machines through Continuous Self-Modelling

They conclude the use of predictive models might lead to higher levels of machine cognition. However, their research is limited by the explicit representation of the body, which does not represent how organisms maintain self-models. In addition, they assess their models by visually checking with a human eye if the model corresponds to the robot's chassis. This might lead to the development of biased models that try to mimic the physical representation, eliminating the chance of the creation of a more accurate model, but that does not visually correspond to the robot. They explain "the use of implicit representations such as artificial neural networks – although more biologically plausible than explicit simulation – would make the validation of our theory more challenging". We believe the assessment of a model should be purely based on the level of functionality and accuracy of the model.

2.2. Predictive brains & Takens Theorem

A recurrent theme in research around neuroscience and robotics is the thought that the brain is a predictive machine [5-8]. The brain is described to be a network of cells constantly predicting and comparing with actual results, trying to reduce prediction error. Our project uses this theory in practice, as we wish to predict the future movement of our robot through a model. The more accurate the model of the environment, the more accurate the predictions in the future will be.

The prediction process depends on the system we are basing our model on. The environment that is modelled can be described as a dynamical system with precise states. Our predictions therefore depend on Takens Theorem [9], which depicts that a chaotic dynamical system can be reconstructed from a sequence of observations of its previous states.

2.3. Neural networks

Neural networks are a series of nodes and connections that mimic the functionality of a brain's neurons and synapses. Neural networks have been used in all sorts of research and are valued for their ability to learn patterns. A Recurrent Neural Network (RNN) allows connections to form a cycle, allowing it to produce temporal behaviour.

RNNs with gradient descent are good at short-term dependencies, for example with the learning of a musical structure [10], however struggled with learning the global behaviour. This can be understood as a difficulty to see repercussions of actions from the distant past, but an ease when these actions are from them near past. This effect is explained by the vanishing gradient problem [11-12], and is the reason gradient-descent methods struggle with long-term dependencies.

A new architectural approach was proposed to help deal with long-term dependencies called NARX recurrent neural networks (nonlinear autoregressive models with exogenous inputs) [13]. Research from July 2017 found that although NARX networks are not immune to the vanishing gradient problem, they tend to perform much better with long-term dependencies [14].

$$y(t) = f(u(t - D_u), \dots, u(t-1), u(t), y(t - D_y), \dots, y(t-1))$$

The previous equation represents a NARX recurrent neural network, where *u(t)* and *y(t)* represent input and output, *Du* and *Dy* describe input and output order, and *f* is a Multilayer Perceptron.

The NARX network is trained in a Series-Parallel architecture, where the inputs and targets are given to the machine-learning network, the model then learns the relation between input and output [15]. We call this "open-loop" training. When we use the NARX to predict, we provide the network with the inputs and also use the output of the network as one of the input to the next timestep prediction. This is called "closed-loop" prediction. Lin et al explain that the output delays help the network propagate gradient information, reducing the sensitivity of the network to long-term dependencies.

2.4. Genetic Algorithms

Genetic algorithms (GA) are a common approach to solve optimisation and search problems using the same process as natural selection [16]. They use genetic manipulation functions such as crossover and mutation to create better individuals. We will be basing our algorithm on Harvey's Microbial GA [17]. An important element of the GA is the fitness function that calculates and assigns a score to every individual's performance, which we then use to sort and eliminate.

2.5. Braitenberg vehicles & simulation

To support and test our project, we will need a simulation capable of representing the behaviours of robots. In robotics research, it is common to use a set of simple robots called Braitenberg Vehicles [18]. These theoretical robots simulate neural connections between sensors and motors and can express multiple behaviours such as moving towards the light or running away from the light (portrayed below).

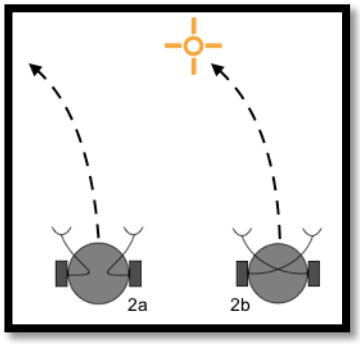


Figure 2-2: Example of Braitenberg vehicles 2a and 2b, where 2a connections incite movement away from the light, and 2b where the connections incite movement towards the light

The advantages of using these vehicles are that they are simple to produce and have a wide variety of possible behaviours from simple parameters. Their neural connections can be represented as a neural network with weights, which can represent genes for the genotype of the GA's individuals. Furthermore, research has been done on optimising fitness functions for Braitenberg vehicles, as they can allow a GA to get to a solution faster [19].

2.6. Hierarchical brains

We will be looking at the creation of a multi-layer structure of vehicle control systems, which resemble the functioning of a brain. We have been inspired by recent neuroscience studies to structure the control systems as a hierarchical brain structure [20-21]. The result of such a structure has been shown to increase metacognitive ability [22].

3. Methodology & plan

We wish to create a project that can be understood and used by others, and create a code base that is easy to use, versatile, and scalable. This will allow further development for this project, but also a solid ground for further research.

3.1. Nomenclature

Here we will define our chosen nomenclature for special terms.

Model: a NARX network trained to represent the environment

Vehicle, **Agent**: A Braitenberg vehicle with a brain, in the simulation

Brain: a weighted network mapping sensors to motors

Control System: the controlling mechanism of a vehicle, in this case a brain

Predicting: using a model to get the next state of a system from the current state

World: simulated environment where the agent is present

Offline/online: Offline refers to a process that is executed outside of the simulated world, whereas online refers to a process that is executed inside the simulated world.

Small / large models: This represents the number of hidden layers constituting the model. The more layers a network has, the bigger its structure.

Real data / values: Used when describing the data a model is using to predict. A model will look back for a certain number of states (number of delays) when predicting the next state, if the past states contain data a vehicle has collected in the simulation, this data is referred to as real data.

3.2. Programming

The chosen language for this project was Python, for its large standard library, community-based development providing many custom modules, and it's code readability. Python is an increasingly popular language for data science and machine learning, and has multiple libraries for neural networks. To work with Machine Learning libraries, a common approach is using Numpy for data structures and management, and a neural network library such as Scikit-learn, TensorFlow, or Theano. We will be using a library developed by Dennis Atabay called Pyrenn, which has been developed for the purpose of Recurrent Neural Networks.

As a standard good practice the project uses Git, a Version-Control Software that allows simple change and progress logs, an easy way to backtrack, and a centralised version of the project for multi-user development. GitHub can also be used for documenting the progress of the project in steps, to help with the understanding of the implementation. For the development of the code, we chose to use JetBrain's PyCharm, a powerful Python IDE complete with many code tools and visual debugging, useful when dealing with different data structures in machine learning.

For the task of machine learning and neural network training, to help with the large network training times a cloud computing service can be used. Microsoft Azure is a cloud computing service where users can upload Jupyter iPython Notebooks to help with processing tasks. We have used this service to help with the long training time required for some big models of the environment in the testing.

The project code can be found on GitHub at github.com/lucasrijllart/Sleep-Wake.

4. System design

In this section we will give an introduction to NARX networks, an in-depth explanation of Sleep-Wake cycles, a short description of the simulation design, and an overview of the programming implementation.

4.1. NARX neural networks

An Artificial Neural Network is a set of connected artificial neurons. These networks can progressively improve their performance at tasks such as image recognition (including feature detection, e.g. numbers, faces), customer product recommendations, or new predictions for a system (e.g. stock trading, weather).

To further explain the concept of a neural network, we can consider the network to represent a function that transforms an input into an output. As an example, we can use the network in Figure 4-1 A, with 2 inputs and 2 outputs. To retrieve the inputs from the outputs we simply add the results of the inputs multiplied by the weights of each connection.

We are most interested in the ability to predict future states for our agent, and the way we can predict the next value is by using our output as a new input. The network can iterate through states, creating a prediction at timestep t+1, t+2, …, t+n. We assume these predictions can only be correct through Takens Theorem, as only in a deterministic environment would one state depend on its previous states.

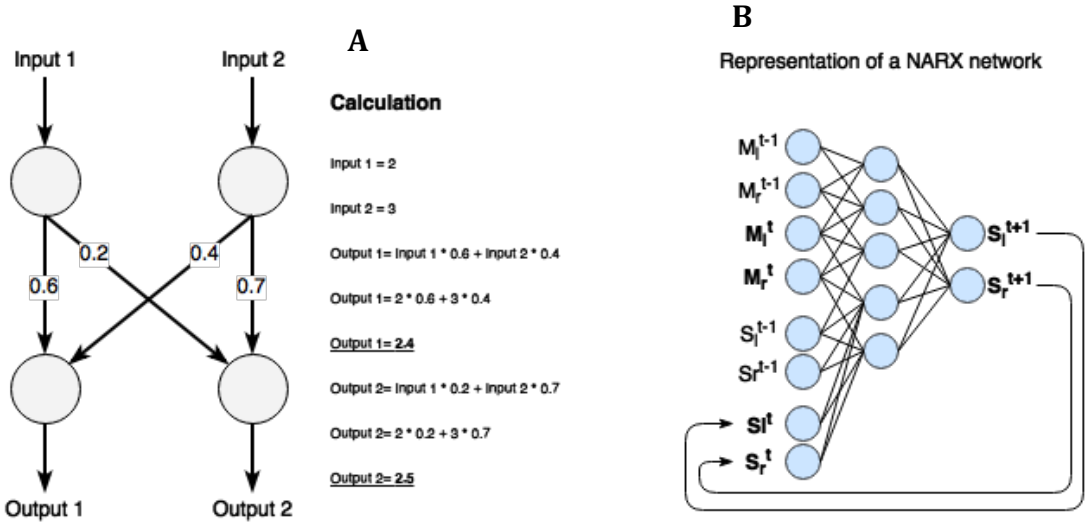


Figure 4-1: Representation of a simple neural netowkr and a NARX network. **A**) A simple neural network with 4 nodes and 4 weighted connections. The text aside details the functioning of it. **B**) A NARX network with 2 input and output delays.

We will now introduce the concept of input and output delays. The NARX network functions just like a regular Recurrent Neural Network, except we define how many steps into the past are considered. To predict the next state of system, we could choose to only use the system's current state, however this would not produce accurate results. By considering the previous states of the system, the network obtains more information and can increase its accuracy of prediction of the next state. Input and Output delays determine the number of previous states to consider. Delayed inputs represent the previous known states of the system, whereas delayed outputs represent the already predicted states to be used as information leading to the next state.

This delay mechanism is shown in Figure 4-1 B, where M^t and S^t represent Motor and Sensory values at time t, and (for simplicity) the network has access to 1 previous state of 2 Motor and Sensory values M^{t-1} and S^{t-1}. We use known previous Motor data and predicted Sensory data as the next input to the network. Once a prediction has been made, the network enters a cycle of feeding new data along predicted data to obtain the next prediction. The earliest Motor information available from the real world then gets removed to allow the new data to become the most recent at time t. For example, all data at time t-1 would become t-2, and t-2 would become t-3, etc.

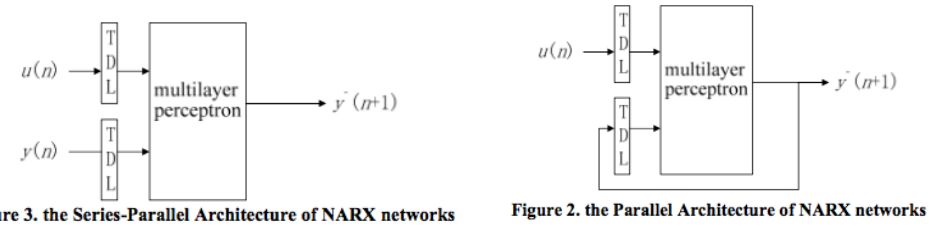


Figure 3. the Series-Parallel Architecture of NARX networks

Figure 2. the Parallel Architecture of NARX networks

Figure 4-2: NARX diagrams from Time series prediction based on NARX neural networks: an advanced approach, by Xie et al, p2.

The two NARX architectures seen in Figure 4-2 are two different methods of using the network. The first, Series-Parallel, is provided both inputs. The second, Parallel, uses one output as the next input. We use Series-Parallel when training a network, providing it with sensory and motor information to train the Multilayer Perceptron to match input to output. We use Parallel architecture when predicting sensory values for the prediction of trajectories, where the prediction gets fed back into the network to predict the next states.

4.1.1. What does modelling mean?

Our network is trained to accept 2 sensory and motor inputs and predict the next sensory values. This allows us to predict the next state (or next position) of our vehicle, with which we can iterate through many predictions and obtain a predicted trajectory.

We call this a model of the environment because it represents an understanding of the world by being able to predict what happens next. Even though the model is not visual or readable by a Human, the proof of its correctness lies in the accuracy of its predictions. However, our models have some room for errors, as they are only used to navigate in the general correct direction, and the positional precision of the prediction is less important.

4.2. Sleep-Wake cycles

To improve the performance of a vehicle, its behaviour needs to be organised efficiently. The Sleep-Wake cycles provide our agent with a structured behaviour. Cycles allow the following:

- Controlled exploration, to only collect data when in need
- Improvement strategy, where the behaviour is optimised for the situation
- Goal-driven approach, to try and efficiently reach the objective

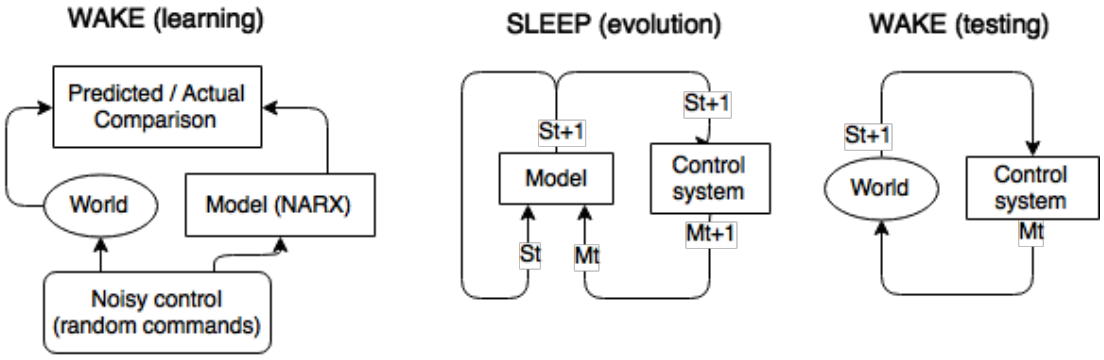


Figure 4-3: Diagram of the separate stages of one Cycle. This diagram shows the first wke stage, sleep stage, and second wake stage.

The general idea is the following: explore and learn the environment to build a model, evolve a new behaviour through predictions, and execute the best behaviour. We will explore each of these stages in detail below.

4.2.1. Wake – learning

This stage is to explore and learn the environment to train a model of said environment. Wake – learning can be summarised in the following steps:

- Random vehicle exploration
- Training of model from exploration data
- Accuracy measure through prediction error

To make the vehicle explore, we create a set of random, noisy commands that make the agent perform exploration in the world. The agent will record sensory and motor information continuously, and order it chronologically.

Once all of the information has been gathered, the data is passed to the NARX network and the model begins the training. This may take some time depending on the size of the model and the number of delays specified, as they impact the complexity of the data to learn from.

After the model has finished training, we can test its accuracy by comparing the accuracy of its predictions to real data. To perform this test we must first move the vehicle for a few time steps to gather some sensory and motor data. By only passing the motor information to the network, we get a set of predictions by the model of the missing sensory information. We can now compare the predictions to the real values gathered by the vehicle and measure the extent of the error of the model.

4.2.2. Sleep – evolving

This stage is responsible for the vehicle's evolution of behaviour. It can be summarised in these steps:

1. Run a Genetic Algorithm from the vehicle's position that uses the model to predict the trajectories of the GA individuals.
2. Obtain an optimised Control System from the GA

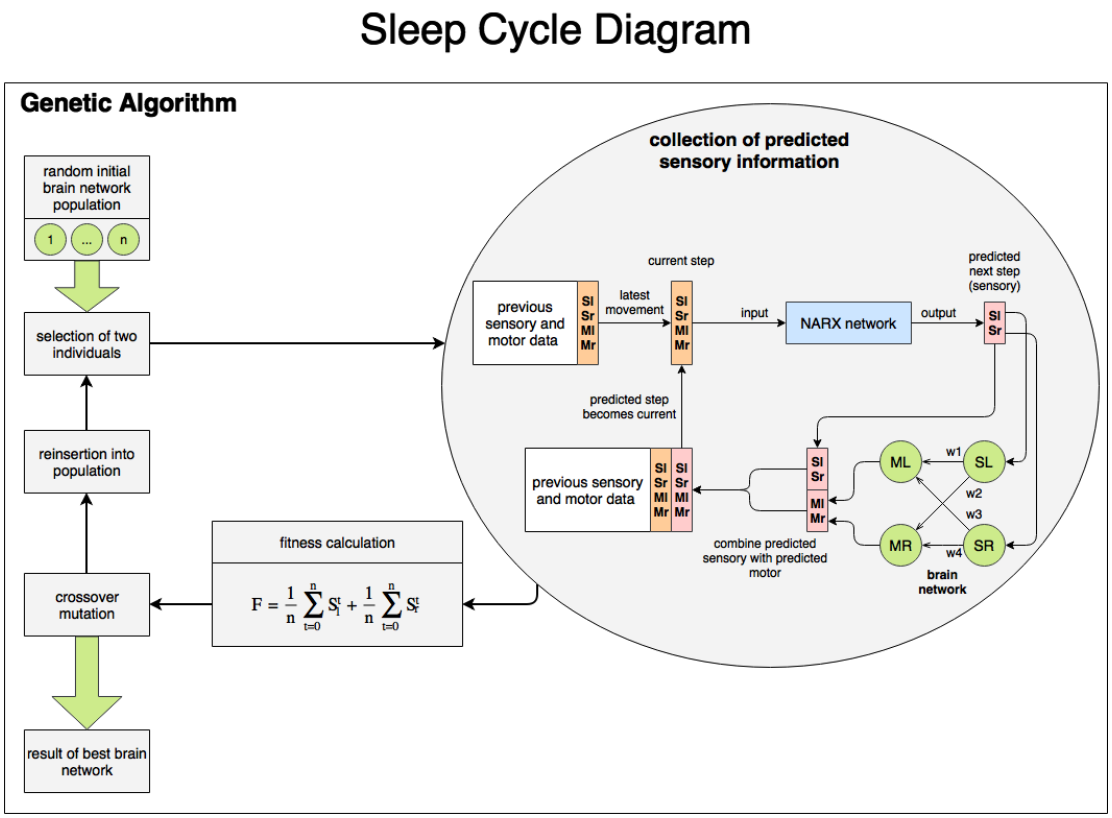


Figure 4-4: Sleep cycle diagram. This diagram shows the inner workings of the GA, from the creation of the population, to the mechanism of prediction, and fitness calculation.

The Genetic Algorithm uses tournament selection, crossover, and mutation to find the optimal brain network. The genotype of individuals consists of the 4 weights in the brain network; one gene being one weight. Tournament selection is the process of selecting two individuals at random and comparing their fitness. The individual with the lowest fitness (loser) is subjected to crossover with the higher fitness individual (winner). Crossover is the process of replacing one individual's gene with the gene of another, and in this case the winner's gene replaces the loser's. After crossover, the loser gets mutated, where there is a chance that a gene gets modified by 1% of the gene's scale.

Before calculating the fitness of an individual, we need to project the individuals' trajectory. Every individual uses the same past data coming from the real-world vehicle. We pass the last known state of the vehicle to the model as input, with a set of 2 sensory values and 2 motor values. The model will then predict the next state, which is a set of 2 sensory values. These two values then need to be passed in the brain network, that returns the two corresponding motor values. It is at this point that each individual will produce a different set of motor values. These values get added to the individual's list of sensory and motor data, and the mechanism is repeated with this latest set into the model. This process is repeated for a set amount called "look-ahead", that determines how many future states to predict.

Once the set of sensory and motor data has been built from predictions, we can calculate the fitness of that individual. The fitness equation is described as follows:

$$F = \frac{1}{n}\sum_{t=1}^{n} S_l^t + \frac{1}{n}\sum_{t=1}^{n} S_r^t$$

This equation depicts the addition of the average sensory value for left and right sensors. The n represents the total number of values for each sensor, and both sums represent the average of all the left and right sensor values. The resulting value is the fitness of the individual. The GA is set to iterate for a specified number of generations. One generation equals a tournament for every individual of the population, meaning that with a population of 10 individuals executed for 2 generations, we will have 10 * 2 = 20 tournaments.

After the GA has finished, it returns the individual with highest fitness to be the selected behaviour for the next stage.

4.2.3. Wake – testing

This stage simply consists of using the best behaviour returned by the sleep stage into the world.

Once the sleep stage is finished and the optimised brain has been given, we assign this brain to the current vehicle and execute it for a given number of iterations.

4.2.4. Multiple cycles

It would be possible to perform another cycle after the first one has ended, which could be beneficial for evolving an optimised solution. The second sleep-wake cycle would model the environment described by the first model and control system. This means it would learn what actions are possible under the command of the first brain.

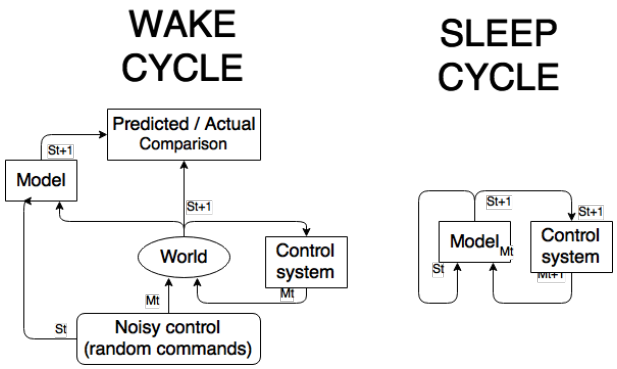


Figure 4-5: Diagram representing Sleep-Wake cycles when used in repetition

The second model would capture the environment of the first evolved control system. The training data given to this new model will be a collection of random exploratory trajectories, which already follow the movement of the first brain. We can visualise this training data as a noisy control system, which the model can learn to control with another control system.

The brain structure follows hierarchical structure, as observed in recent neuroscience research. In our case, we construct a list of brains where the sensory signal passes through one after the other until the last one, where it reaches the values of the motors. From one control system to the other, the motor values are added together to create the motor value that is impacted by every control system.

This mechanism might lead us to simpler modelling tasks, as it is divided across more than one model, and perhaps a better performance from the second model's predictions.

4.3. Code implementation

The following is an explanation of the implementation of this project. Let us first establish the functional requirements.

Functional requirements:

- Working Braitenberg simulation
- Genetic algorithm for vehicle brains
- NARX network used to model and predict
- Sleep-Wake cycle manager
- Controller of cycles
- Data collection and presentation of results

From this specification, we construct a flow chart of the different classes present in the implementation. The diagram is shown below, in Figure 4-1.

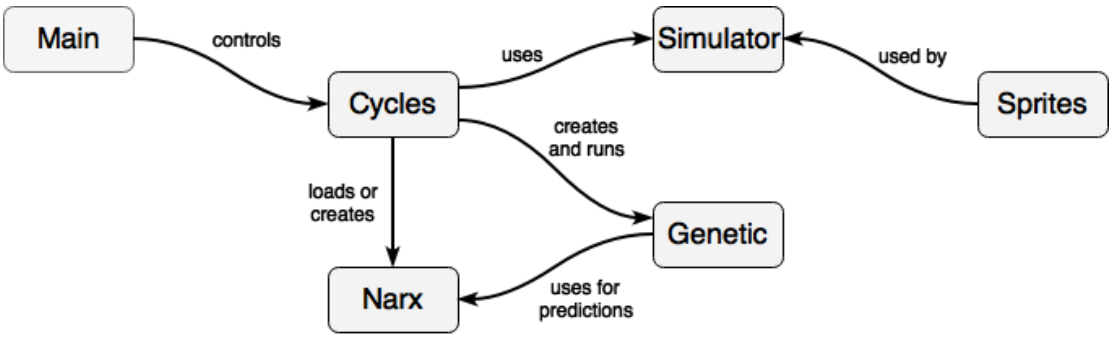


Figure 4-6: Flowchart of program classes' architecture and interaction logic

From the flowchart we can see that the Cycles class controls most of the system logic. The Cycles class manages the Sleep-Wake cycles of a vehicle, controlling which cycle is active and which step is executed next. Cycles uses the Simulator class to run the vehicle in the simulation, for example when the vehicle is collecting data to train the network, or after the best brain has been returned by the GA. The Main class can load networks, execute cycles, and collect data from tests, allowing us to generalise the behaviour from the vehicle.

In the sleep cycle, the Genetic class is used to start a Genetic Algorithm with the goal of evolving a brain to a desired behaviour. As the individuals of the GA do not have access to the simulated world, their trajectories are calculated using predictions from the Narx class, which will predict the next sensor values from a current state.

The Simulator's main goal is to simulate the environment and run a vehicle inside it. It renders the screen with the sprites, performs calculations on sensory and motor information, and updates the sprites at every iteration. The vehicles are represented by the Sprites class, which holds the different types of vehicles as well as the light object.

$$S_l = \frac{10}{\Delta l^{0.5}} \qquad S_r = \frac{10}{\Delta r^{0.5}}$$

The sensory values are calculated through the above equation, where Sl and Sr represent Sensor left and Sensor right, and Δl and Δr are the Euclidean distance of the sensor to the light.

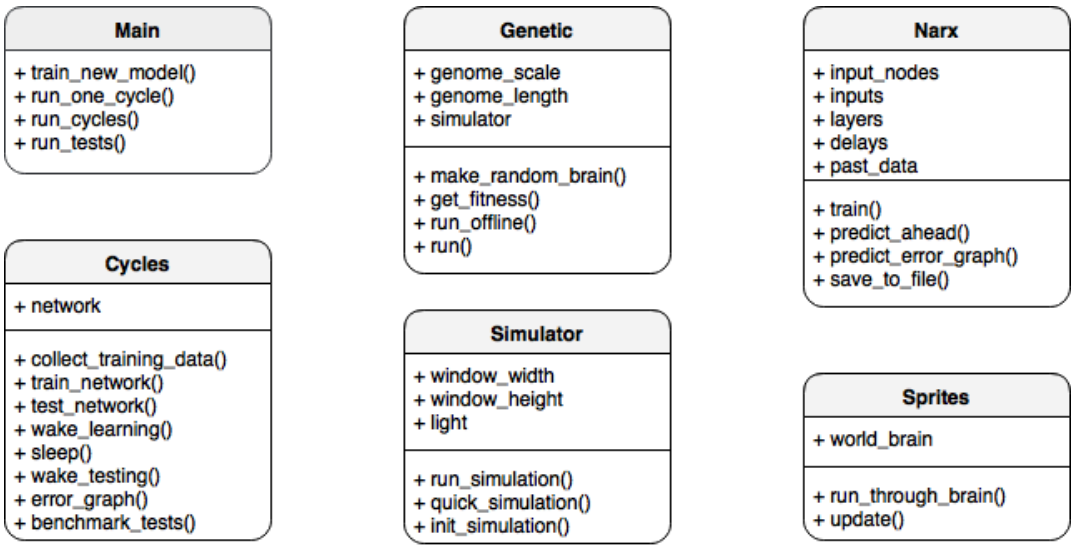


Figure 4-7: Class diagram of the proejct, with important methods and fields.

In this next section we will explain the functioning of the classes in detail from the class diagram above.

Sprites

This class contains all of the objects that will be used in the simulation. Firstly, we need to describe the main vehicles we will use: BrainVehicle and RandomMotorVehicle. The former is a vehicle that updates its wheels through the network of the brain, the latter uses random noise to create a random movement to explore the environment. The random vehicle is used in the collection of the sample data for the training of the network. We use a third vehicle called ControllableVehicle for the sole purpose of tracing the predicted behaviour of a GA-evolved brain on screen. It gets assigned a list of wheel velocities and re-enacts what they dictate. The class also controls the Light class, which only keeps its position for calculating its distance to a sensor.

All sprites have access to a variable called "world_brains", this is a list that represents the hierarchical brain structures present after abstracting the previous brain into the model. The Brain and Random Vehicles access the run_through_brain() method, which passes the sensory values through the brain network and return the wheel values. All vehicles access the 3 same methods for the calculation of sensor intensity, update of vehicle position, and update of vehicle image rotation.

Simulator

The Simulator class's only goal is to run a vehicle in a simulation. It creates an instance of PyGame that controls the screen and rendering. A vehicle is then run for a number of given iterations in the environment.

The main loop's logic is as follows: update all sprites (movement, values), draw all sprites on the screen, and finally draw extras such as sensor and motor values on the vehicle and the previous and current trajectories in appropriate colours.

Narx

The Narx class controls the NARX network used as model of the environment. This class allows us to create and train a new network from sample data, predict the next sensory values from a given state (with or without known wheel values), saving a network to a file for future use, and load such file into a network. The functions use the PyRenn library to control the behaviour of the network.

Genetic

The Genetic class controls the working of the GA. The are two possible ways of using the GA class: calling run_offline() will start a GA and use as fitness function the mean between the two sensor values which have been predicted by the NARX network, or calling run_with_simulation() which will use the fitness of a vehicle that has access to the real-world data.

The GA's individuals are lists of integers representing the vehicle's brain network connections. For example, the brain of a Braitenberg Aggressor is [0, 5, 0, 5]. The GA uses random tournament selection for two individuals, then crossover and mutation on the less-fit individual to create a new brain.

The main loop's logic follows this order:

- Create population and calculate the fitness of every individual
- Iterate for the number of generations given
- Pick two individuals randomly for tournament, highest fitness crosses over the loser, then the result gets mutated
- Place the new individual into the population and repeat until end of generations

The GA has 2 other functions that are used outside of the GA class: the creation of a random brain, and the calculation of the fitness of a real-world individual's fitness, both of which are used when running tests.

Cycles

The Cycles class contains all the functioning of the different Sleep-Wake Cycles. It manages the vehicle in 3 main stages: wake learning, sleep, and wake testing. These methods will run the different steps of every stage on a vehicle, managing the data flow and vehicle position.

Some other methods are present to help deconstruct the tasks such as: a method to collect sample training data to give to the model, a method to train a model with data and a number of configuration parameters, and some methods to format data into the correct structure. Some methods also allow one full cycle, or multiple cycles to execute.

Main

This class controls what test to run and the configuration of the cycles as well. It allows the modification of parameters for all methods of Cycles, from number of GA individuals to number of iterations in the wake-testing stage. There are also Booleans that can be toggle to run different tests, which will be covered in Chapter 5.

5. Results

In this section, we will evolve a Braitenberg vehicle in our simulation, explore the modelling of the environment, report on the evolution of a control system using prediction from the model, and try to iterate through another Sleep-Wake cycle.

5.1. Can we evolve a Braitenberg Vehicle? (Test 1)

The evolution of a Braitenberg vehicle to reach a light using sensors is a trivial task, as they were designed to achieve such a behaviour. However, we will recreate this test in our environment to establish the accuracy of our simulator for the next results. This will also allow us to hone the parameters of the evolution for this particular case of vehicle.

Goal:

1. Create a Genetic Algorithm that evolves control systems (brain network) with goal to reach the light.

To perform this test, we create a random initial condition, composed of a set of coordinates and angle, where the vehicle will spawn. The initial condition is excluded from a radius of a certain distance from the light, as we do not wish for the vehicle to be created too close to the light. The goal of the GA is to maximise the fitness function, which we have described as the average sensor values of the vehicle. In this way a vehicle will have a high fitness if it stays close to the light the longest. Once the vehicle is created, the GA is executed from that position and returns the best individual possible from the amount of generations specified. To prove the efficiency of the GA to find a successful brain, we can establish a benchmark test that compares the performance of evolved brains to that of random brains.

We can see from Figure 5-1 A that a vehicle has been successfully evolved to reach the light, maximising its sensory values. This may seem like a simple task of going forwards, however the resulting brain was a “lover” configuration, where the vehicle slows down when it gets closer to the light. The GA has evolved a brain that reaches the light from its given initial position, with a starting population of 20 random individuals for 40 generations. The result of the GA is the brain [1.9, 2.0, -0.7, -0.6] with a fitness of 3.3. The brain returned is a variation of Braitenberg’s Vehicle 3.A. the “lover” vehicle. It orientates towards the light at first, and slows down when it approaches, due to its inhibitory values on the second sensor.

The population’s maximum fitness increases drastically over time, with the average slowly increasing too, however the minimum fitness remains around 0.5 (from Figure 5-1 B). This stagnant minimum could be the result of using a wraparound mutation function, which forces a mutated gene that has reached a value greater than the gene scale to become the minimum scale value.

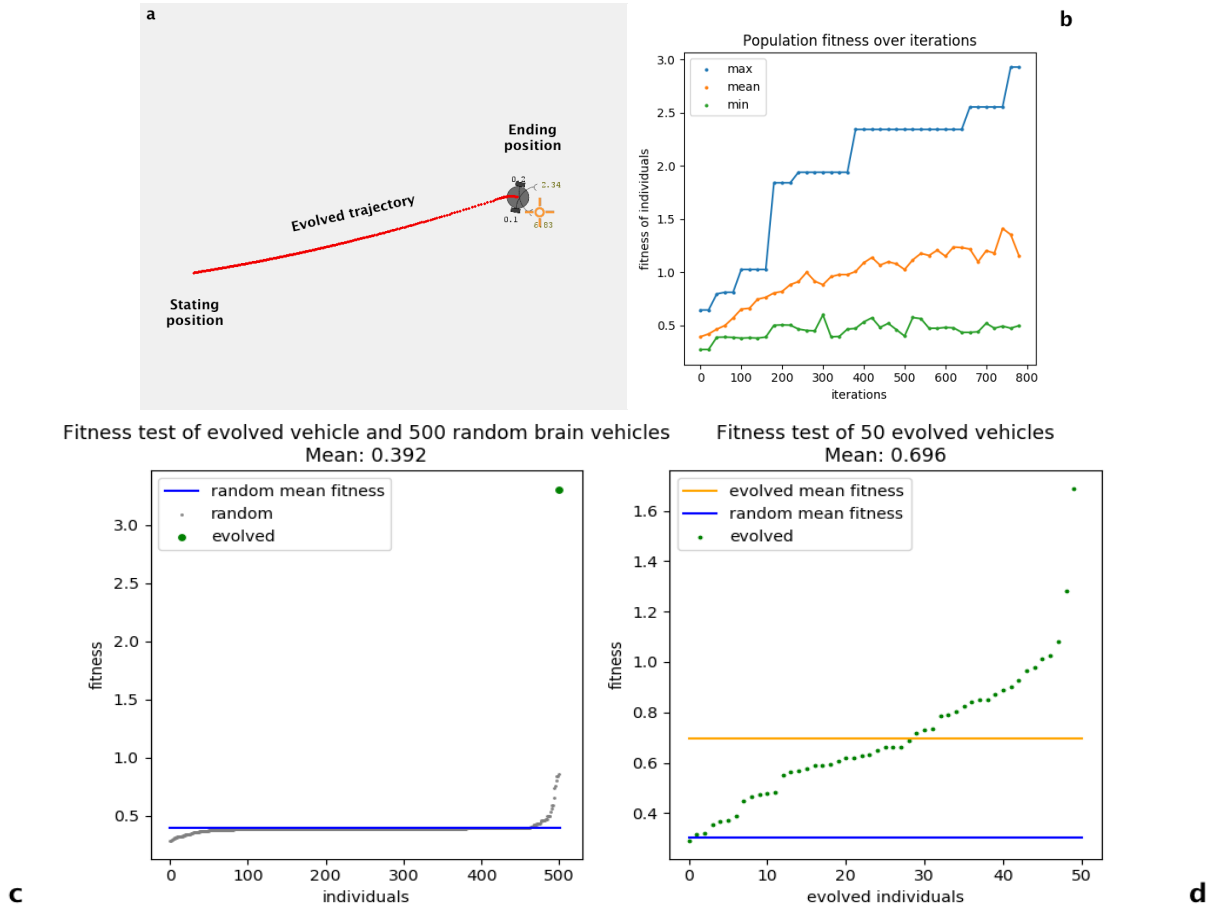


Figure 5-1: Example evolution of a vehicle and benchmark test for evolved vehicles. Example trajectory of an evolved vehicle from a random initial condition (**a**). Fitness of population over iterations with maximum, minimum and avearage (**b**). Fitness comparison between random brain vehicles and evolved vehicle from the same initial conditions (**c**). Benchmark test for 50 evolved vehicles with 10 random brains from each of the positions of the evolved, resulting in the average of 500 random brains (**d**).

Figure 5-1 C shows the resulting fitness of an evolved vehicle and of 500 random vehicles, all of which have the same starting position. This graph shows that the evolved vehicle performs much better than chance, with a fitness that has more than tripled the best of the 500 random brains.

We can now compare the average fitness of evolved vehicles from random initial conditions, to understand how much better these are compared to chance. This is represented in Figure 5-1 D, where 50 evolved vehicles are plotted and their average is compared to the average of 500 random vehicles. This graph shows that the average evolved brain will have a fitness of 0.67, whereas the average random vehicle only 0.35. We can see that the fitness of an evolved vehicle is greatly superior to chance, which suits the expected results of the task. The evolved vehicle's fitness varies significantly, which is due to initial conditions of varying difficulty, random initialisation of the GA population, and a restriction in allowed time to solve.

We now have set a benchmark of performance for evolved brains. Before evolving brains based on the predictions of their behaviour we need to understand the intricacies of environment modelling.

5.2. Can we build a model of the environment? (Test 2: Wake)

In this section we will test the predictive ability of NARX networks and their performance when applied to more complex, plausible scenarios.

5.2.1. Performance of NARX and RNN with identical initial conditions

To establish the baseline of accuracy of a NARX network, we want to create the simplest environment and compare its accuracy with a regular recurrent neural network, and the impact of delays on predictions. To train a network in a simple task, we create a training population of trajectories starting at the same coordinates and with the same angle. The only difference between them is their random trajectory, which is identically random for both networks. We refer to this as "identical initial conditions".

Goals:

1. Compare the accuracy of a NARX network and a recurrent neural network. The result will show which network is more accurate in its predictions.
2. Compare the accuracy of two NARX networks trained on the same data, but with different delays. We will compare both their accuracies with the same exploring vehicle, to see what difference the delays of the networks make in the predictions.
3. Compare how accurate two networks of different delays predict on average for different lengths of exploration. This test will allow us to see if a network that is trained to look further into the past can have better predictions than a network with a short sight, on a long exploration task.

To observe the accuracy of a network, we run a new random vehicle, determine a cut-off point and remove all future sensory values from this point. This set of data (sensors and motors up to a point, then only motors) is given to the network, where it will predict the missing sensory values based on the first missing state, while looking backwards in time for a certain amount. The delays of the network define how many steps into the past it will consider for the prediction of the next state. Once it predicts enough values, the historical data will consist of only predicted values. At this point, we can anticipate that the prediction error will increase, as there is no real-world data available. We can observe how accurate a network's predictions are by comparing the value predicted to the value the real vehicle recorded.

As expected, NARX networks are more accurate, and need less training than an RNN to reach good performance. From Figure 5-2 A and C we can see that the predictions of the NARX network follow the initial trend of the values, whereas the RNN cannot replicate the trend of sensory intensity.

In Figure C is shown the predictions of two networks, one trained on 10 delays shown in red, the other on 40 delays, shown in green. The continuous line represents where the predictions are based off of real-world data, whereas the dotted line represents where the predictions are made based off of previous predictions. Both networks finished training with an error function of 6.1e-6 for the 10 delays and 7.5e-6 for the 40 delays. As these numbers are very close, their predictions can be compared.

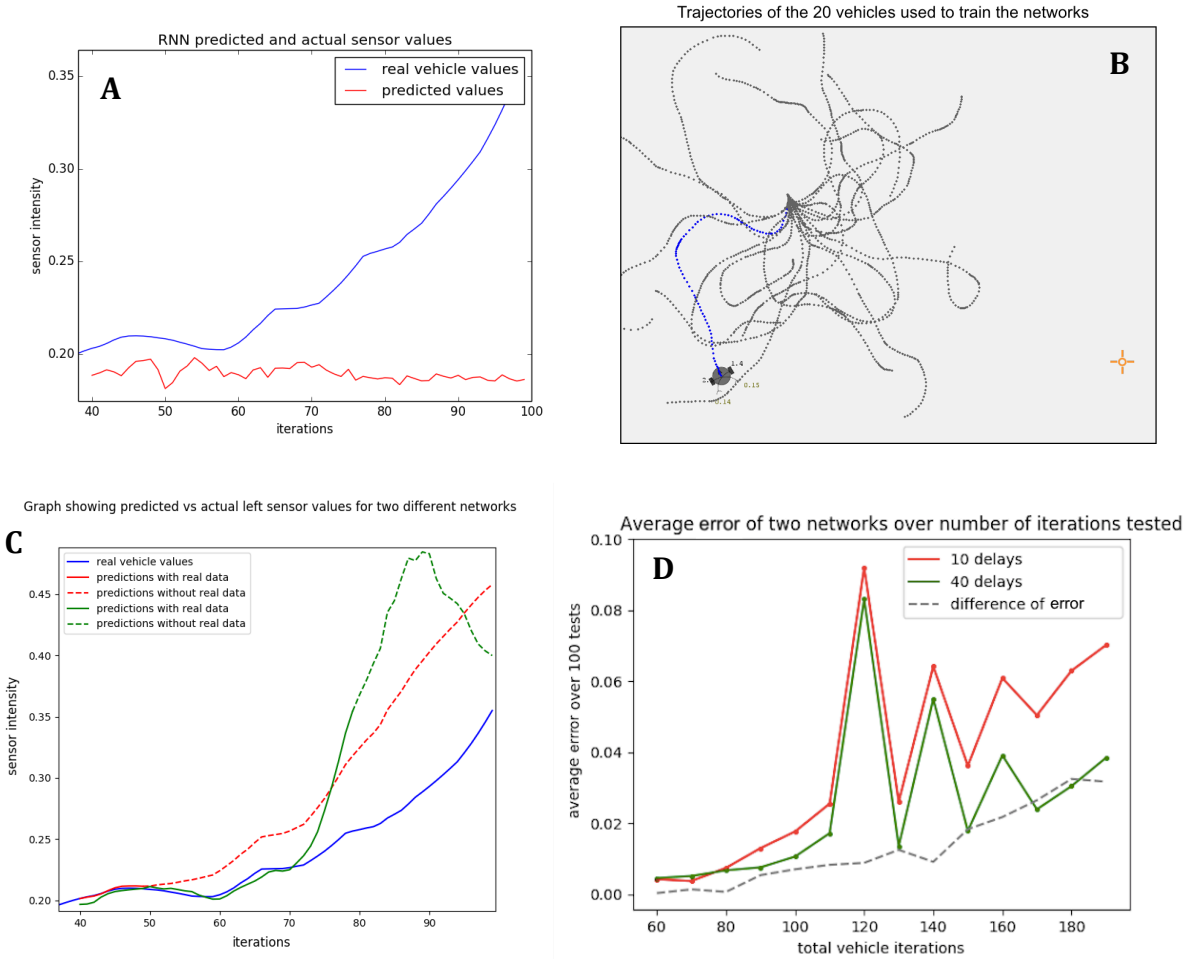


Figure 5-2: First environment models. Example predictions of the left sensor for an RNN (**a**). Trajectories used to train the networks, 20 vehicles starting at coordinates [300, 300] with an angle of 200 degrees (**b**). Left sensor predictions of two networks for a random vehicle (**c**). Both networks were trained for 200 epochs, with data of 20 trajectories of 100 time steps each. The red network was trained with a delay of 10, and the green of 40. Graph of average Mean Squared Error of a network for 100 random trajectories, for increasing trajectory length (**d**). Also present is a line showing the difference between the two network's accuracy.

We can see that the red network's line initially seems to follow the trend of the actual sensors accurately. The same can be said for the first half of the green network's continuous line, accurately predicting the sensors for 30 time steps, then diverging into high intensity. For the first 10 predictions, the network trained on 10 delays performs better, however for the first 40 predictions, the network trained on 40 is more accurate.

This result suggests that the delays of a network determine for how long the network will try and produce accurate results. We can also see that even though the red predictions start to diverge, they still follow the features of the real values, which suggests that if a network can very accurately predict the first few values, it can still perform decently when dealing with no real values.

We hypothesised that networks with longer delays could predict better for longer trajectories. From Figure 5-2 D we can see that the network with 10 delays has a smaller error for vehicles of 60 to 80 time-steps. After 80 time-steps the network of 40 delays consistently maintains a lower error than the red network, and while the error still increases with the number of iterations, the difference between the accuracy of the two networks also increases. The test was run for longer and the bigger-delay network always performed better for longer trajectories. The graph's lines aren't smooth because the initial conditions could be disadvantageous to the situation, however we are more interested in the difference between the two networks.
This result suggests that longer delays allow the models to learn longer patterns of trajectories, helping them match it to situations in the future, and maintain better predictions.

The NARX network's predictions were accurate to some degree, however the task is simplified by the identical initial conditions. We need a general model of the environment that is not dependant on the position.

5.2.2. Random initial conditions

Starting from the same position isn't biologically plausible; therefore we can simulate a diversity of information by randomising the initial conditions. The goal of this test is to train models with random data and understand whether a model's accuracy is affected by a training data of diversified initial conditions.

Goals:

1. Observe the predictive accuracy of a highly trained random-data model. We will compare the predictions of the model to the real-world values as seen in the previous tests.
2. Run many random vehicle trajectories and compare the error generated by the predictions to check whether the model can predict from random starting positions.
3. Compare the performance between networks trained with random initial conditions and identical initial conditions.

To perform the last goal accurately, we need to train both networks with vehicles having the same trajectory noise, and test them on the same set of randomly positioned trajectories. Then repeat for many models to get a generalised result.

The well-performing model was given 200 randomly positioned trajectories of 100 time steps, trained with a delay of 40 for 300 epochs, which reached a training error function of 6.7e-5. From the graph at Figure 5-3 A, the predictions were perfect for the whole time it had real data and the next iteration where it had all predicted values from the real ones. After these two sets of excellent predictions, the rest of them follow precisely the trend of the real values, but with some offset, generating an increasing error. However, the predictions remain rather accurate up until timestep 300, which demonstrates an incredible predictive performance from the model.

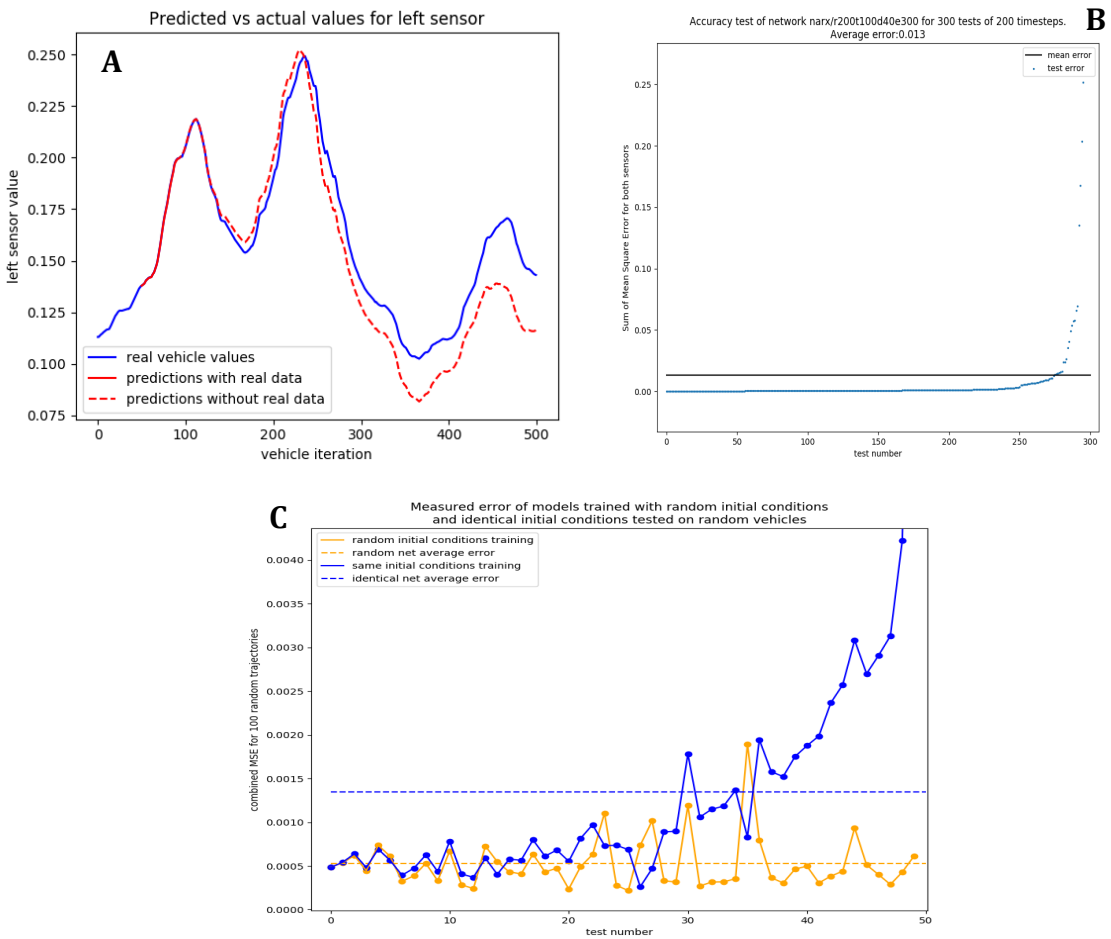


Figure 5-3: Random initial conditions tests. **a**) A graph of the predicted and actual left sensor values of an exploring vehicle, this was performed for 500 iterations, with predictions starting at iteration 50. **b**) Accuracy test on model, consists of 300 test trajectories of 200 time steps. The model used for both a) and b) was trained with 200 trajectories of 100 iterations, 40 delays, [4, 20, 40, 20, 2] layers, and 300 epochs. **c**) 50 tests where 2 models were trained, one with random initial conditions and the other with identical initial conditions, then both tested on the same 100 random vehicles, results sorted by difference of error

With predictions starting at iteration 50, and real values disappearing from delays at iteration 90, the model can predict from previous predictions for 5 cycles data before error becomes significant. In Figure 5-3 B, the same model is tested with 300 trajectories of 200 time steps. The average means squared error for both sensors is around 0.013, however around 250 of these tests are close to 0. A hypothesis of why some tests receive such a high error is that the training data never reaches the position of the light; therefore the model has never seen sensory intensity of that level and cannot predict how intense the light reading would be when reaching it with an exploring vehicle.

These two results confirm that it is possible to accurately generalise an environment through a model when the training data is collected from random trajectories.

Figure 5-3 C shows the average error of two models, in orange the model with random training and in blue the model with identical starting training. We will refer to them as the "random model" and "identical model" for short. The average error of the random model is of 5.2e-4, and 1.3e-3 for the identical. This proves the random model performs better when tested on random data, showing

that the other model predicts worse. All other values between the models are the same (training data noise, testing data); even the network's initialisation is generalised as we perform the test for 50 models of each.

This result puts forward that the model's performance was not determined by the identical conditions of the previous test, and that random training data only makes a model more accurate. However, in the case of a real robot, the data gathered to train the model would not be random trajectories, and we need a new way of collecting training data.

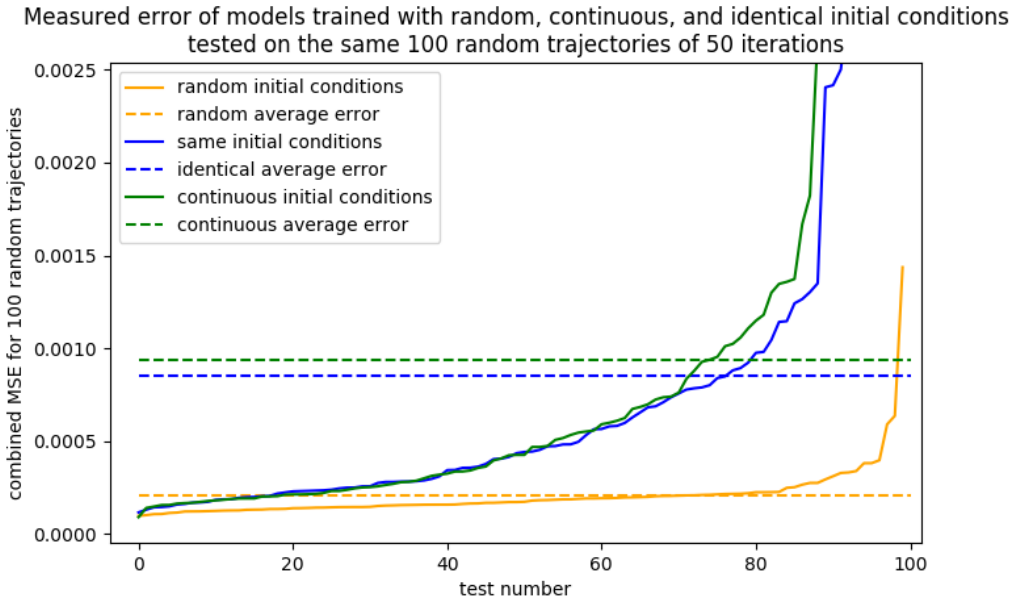
5.2.3. 3. Continuous initial conditions

For the training of a model to become more realistic, we can generate training data by running one vehicle for many time steps, segment that information and train the model with it. By following simple Machine Learning theory, we train the model with a random sample of 50% of the data.

Goals:

1. Compare the performance of a model trained on continuous data to the previous models seen.
2. Assess whether this realistic method could be successful with physical robots.

A



B

	From training positions	Random positions
Average error	4.751 e-05	2.538 e-03

Figure 5-4: **A)** Graph of combined Mean Squared Error for 3 networks trained with different methods. Random initial conditions describe training where the data is randomly gathered. Identical initial conditions maintain the trajectories' starting coordinates and angle identical. Continuous initial conditions follow the collection method of a real robot, by having one trajectory start at the end of the previous. Three networks are trained with data consisting of 30 trajectories of 50 iterations with a delay of 10 and 100 epochs for every test, and are tested on 100 random trajectories of 50 iterations with predictions starting at t=10. The error value is the combination of the MSE for the 100 predicted trajectories, and all 3 models are tested on the same data. **B)** Table showing average error for 10 models depending on the test positions. Both values are average error for 500 tests of 100 timesteps, models tested were small with 10 delays and trained with continuous data. The tests were from random positions from training data and random positions.

From the graph at Figure 5-4 A, the average error for the continuous model is higher than the random model, however it remains very similar to the identical model. This may be because the collection by continuous movement observes a similar area than the identical, and also may lead to the vehicle straying further from the light, never to come back. This could potentially create a worse model, as it would never experience intense sensory values from being close to the light, and therefore not be able to predict high values when testing.

As the average error of continuous training is similar to the identical training, and we previously have shown that the latter's predictions were still acceptable, we can assume that the continuous model's predictions are acceptable. For more than 40 tests, the error value of the continuous model was practically identical to the random model. From this result, we can confirm that the accuracy of models trained with continuous initial conditions is worse than the random training, but acceptable.

Furthermore, the networks are tested on random data, which is not a realistic method for testing the network. If we assume that a real robot collected the data and then uses that model for predictions, the robot will never be in a location it has never seen, unlike the random testing, and the accuracy should remain high. The table in Figure B shows 10 continuous-trained models being tested 2 different ways: one from the position it would be after training and random positions all around the environment. The models performs much better on the first set of test data, which shows that a network will perform better on the data it is trained on.

We conclude that a real robot gathering training data in continuous intervals would be an acceptable method of gathering training data for a model. Now that the environment has been successfully modelled, we want to explore whether the predictions can be used to evolve a vehicle's control system.

5.3. Can we evolve a Control System based on the model? (Test 3: Wake Sleep Wake)

Now that we can train accurate models, we need to use their predictions for the fitness of the individuals of the GA. Below are the early results of bad performance, and then a method. In these sections below we explore the failures and successes of these evolved brain networks.

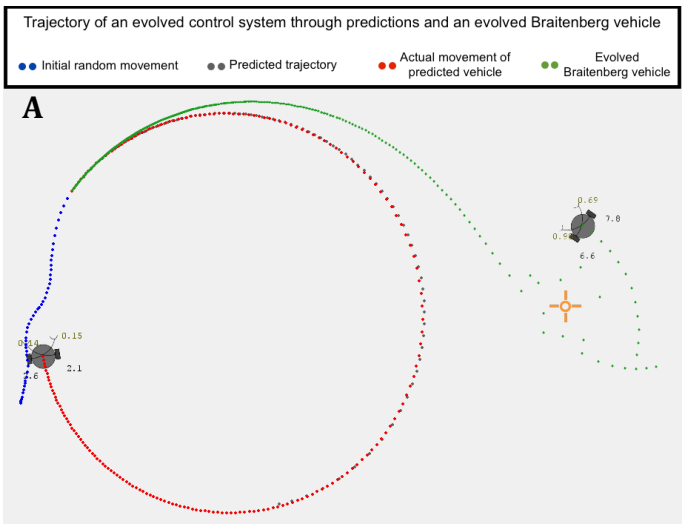
5.3.1. 1. Ballistic trajectory

We want to evolve a brain from a random initial condition to go towards the light, by only using a prediction of its trajectory as measure of fitness.

Goals:

1. Observe the evolutionary results of brain networks based on their predicted trajectories
2. Compare the success of the individuals evolved using predictions and real-world data.

The result of a GA using predictions of vehicles ("predictive GA" for short) will be extremely dependent on the result of the predictions from the model. For example, if the model does not predict that a vehicle's sensor intensity increase, but the real trajectory approaches the light, that individual's fitness will be wrongly low. To compare the success of the predictive GA to the real-world evolution of a vehicle ("world GA" for short), we can collect the real sensory data of the prediction vehicle, to assess how well the vehicle did in the simulation instead of its predictions.



B

EXAMPLE EVOLVED CONTROL SYSTEMS
[LL , LR , RR , RL]
[2.6, 0.8, 1.6, 3.7]
[4.0, 5.0, 2.9, 3.9]
[1.6, 5.0, 5.0, 3.5]
[3.2, 4.6, 5.0, 2.2]
[4.9, 3.7, 5.0, 2.0]
[4.2, 4.4, 3.9, 3.0]
[3.0, 4.7, 4.8, 2.1]

Figure 5-5: First results of evolved control systems. **A**) A representative example of a ballistic control system and a correctly evolved vehicle with access to the world as comparison. **B**) 7 example evolved control systems. The brain network is composed of 4 connections: left sensor to left wheel (LL), left sensor to right wheel (LR), right sensor to right wheel (RR), right sensor to left wheel (RL). The maximum value for motors is 5.

From the results of Figures 5-5, the vehicles get closer to the light but don't have a control system that is attracted by light. It seems the GA tried to create the best trajectory that would be closest to the light, not finding the control system of a generalised "aggressor brain". The evolved control systems describe ballistic trajectories from its starting position, meaning the control systems are not generalised, and would not work from a different position.

A generalised "aggressor" brain would be composed of two strong LR and RL connections, and two weak LL and RR connections. It is clear that in the example evolved control systems in Figure 5-5 B, the connections do not follow this trend, and instead are just optimised circular trajectories. This is why the vehicle in Figure A starts with a correct angle of approach, but does not turn towards the light when close, and instead turns away to complete the circle.

To help create more generalised control systems, the unused training data can be used as testing, and use those positions in the fitness function so that a fit brain will need to generalise across many different positions.

5.3.2. 2. Generalisation - Optimisation of the fitness function

To minimise ballistic trajectory, we calculate the fitness from many different starting points, forcing vehicles to be successful in more situations.

Goals:

1. Check if using test data for fitness forces control systems to generalise their movement.
2. Compare the performance of the evolved control system to the benchmark of random brains and a world GA.
3. Generalise the performance of predicted GA's compared to world GA's.

The evolution of a control system results in a vehicle that is attracted to the light and shoots through it, as seen in Figure 5-6 A. This is the desired behaviour of an "aggressor" vehicle, it's brain network being [1.6, 3.0, 1.5, 3.3]. From Figure B it is clear that cross-connections (LR & RL) are the strongest in every example. This confirms that the brains have lost their ballistic aspect and now maintain generalised, light-attracted behaviour.

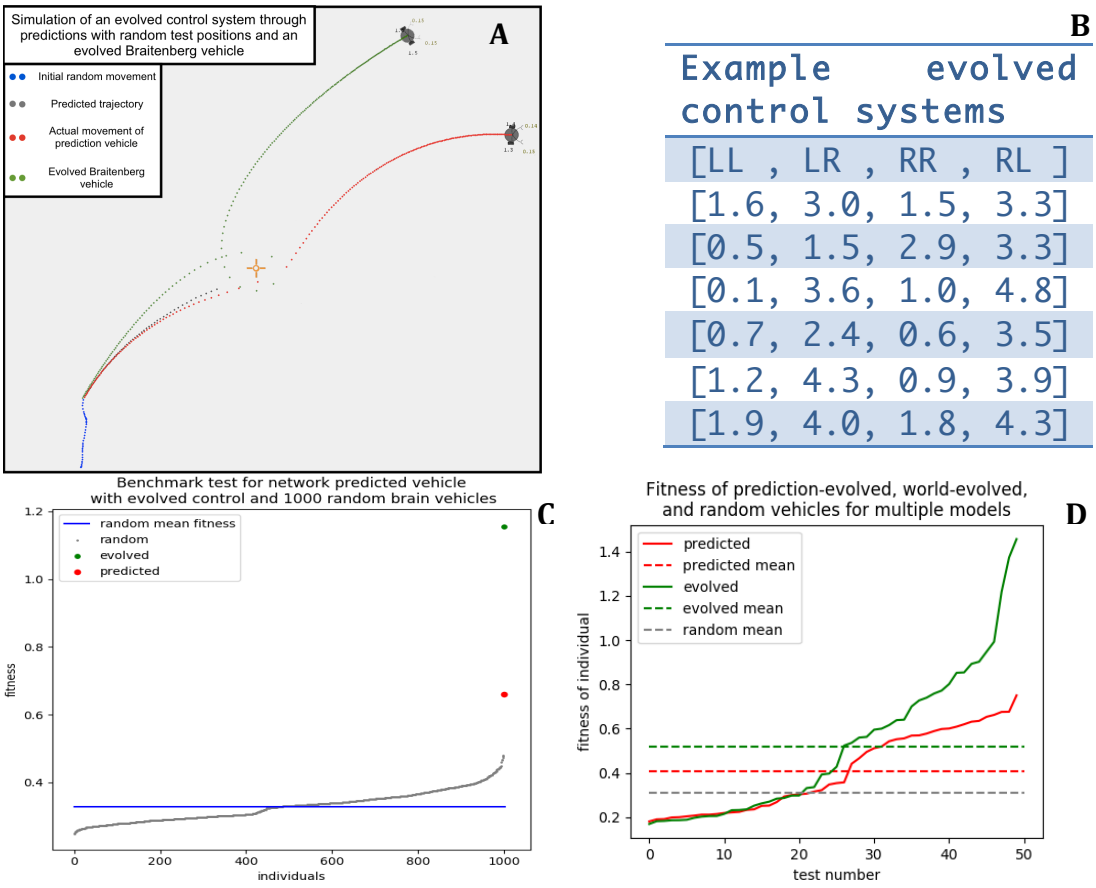


Figure 5-6: Results of generalised evolved control systems. a) Representative example of an evolved control system tested on multiple positions, resulting into a generalised behaviour. B) list of example control systems. c) Benchmark test for predicted vehicle in a), the evolved control, and 1000 random brains. d) Fitness comparison between prediction-evolved and world-evolved vehicles for 100 tests, and the average fitness for 100 random vehicles for every test. The model used to predict with is the same as sub-chapter 5.2.2, and the values are sorted by increasing difference in fitness.

Figure 5-6 C shows the evolved prediction vehicle is better than chance in this case, and as expected performed worse than the GA. We also notice that the aggressor individual has a lower fitness than the evolved and predicted, which is because both of them are slightly optimised for their situations and test data. This is a desired effect, as we want the most performing control system for the position of the agent.

We can now compare the average performance of the predicted brains compared to the world-evolved ones. The graph in Figure 5-6 D plots 50 prediction-evolved, world-evolved and random vehicles sorted by increasing difference. The average fitness for random vehicles was calculated by testing 100 random brains for every test. The world-evolved vehicles seem to perform poorly for many tests, the reason is because we decided to restrict the maximum number of iterations to exclude answers that would take too long. The average fitness for world-evolved agents is 0.5242, whereas predictive agents have an average fitness of 0.4127.

The difference of fitness between world-evolved and predicted brains is significant, however they do follow the same trends. When the evolved performs poorly because of restrictions, the predicted also performs poorly. Another reason why world-evolved brains achieve higher average fitness is because of those very high scores for the last few tests. These are produced because they can sometimes reach the light and circle around it, whereas the prediction brains cannot predict far enough to develop circling, and therefore only "shoot-through" the light. On average, the predicted brains' performance is exactly between chance and the optimal solution. This result is successful as it proves the brains using model perform on average better than chance.

Figure A also shows that the predicted behaviour (in grey) did not reach or circle around the light, meaning the behaviour generated by that predicted trajectory was extended successfully to satisfy the environment. The predicted behaviour represents the look-ahead of the GA when predicting vehicle trajectories. In the case shown in Figure A, the look-ahead nearly reached the light, however this is a special case, as most of the random starting positions are far enough that the predictions don't get close. This confirms that the behaviour generated is scalable.

Now that the prediction-evolved vehicles show an acceptable fitness, but are still ranked lower than world-evolved agents, we should explore the trade-off between fitness lost and time gained.

5.3.3. Time comparison between world and prediction vehicles

We now want to explore how much time has been spared from using predictions instead of the real world.

We can analyse the factors that increase the number of time-steps of a vehicle. Both vehicles add time-steps after the GA where they execute their optimised behaviours. For the world-evolved agent, the time-steps are accumulated by the trial and error, whereas for the predicted agent, the time-steps come from the training of the model. The following describes the trial and error and model training process:

- **Trial and error**. The GA calculates the fitness of an individual by executing that individual in the world for a number of time-steps. This represents the majority of the time taken by the world-evolved vehicle, as a fitness calculation is executed every iteration of the GA.

- **Data for model training**. This represents the majority of the time taken by the prediction-evolved vehicle, as the agent needs to collect a substantial amount of information before training the model.

Goal: Compare number of time steps used by prediction-evolved and world-evolved vehicles.

	World-evolved agent	Prediction-evolved agent
Time-steps per agent	100200	20200

Figure 5-7: Table comparing world-evolved and prediction-evolved vehicle time-steps' in the real world

The results in the table above are for the network and GA used in Figure 5-6 D. The world-evolved vehicle has accumulated the time-steps for 25 individuals of 200 time-steps for 20 generations. In fact, $25 * 200 * 20 = 100{,}000$. The prediction-evolved vehicle's time-steps are created by a training data of 200 samples of 100 iterations. Both results then add 200 time-steps of the post-GA execution.

We can see that the prediction-evolved agent uses 5 times fewer time-steps than the world-evolved agent. This is because the trial and error of the GA would take an unrealistic amount of time to execute in the real world. Moving the trial and error to offline processing was to minimise time spent in the world, and the results clearly show that by training a model, the time taken is inferior.

Now that we have proved the success of one Sleep-Wake cycle, could it make the task easier to execute two cycles?

5.4. Is it possible to have a second Sleep-Wake Cycle?(Test 4: Two cycles)

By running a second cycle, we might be able to rely less on the accuracy of one model, and share the complexity of modelling the environment and reaching the objective to multiple models. The following are the chronological steps executed:

1. Evolve a vehicle from a model and execute it in the simulation
2. Collect a new set of data from an exploratory vehicle subjected to the first control system
3. Train a new model based on the collected data
4. Test the model's prediction accuracy on random samples
5. Evolve a new control system based on the predictions of the new model
6. Execute the new control system in the simulation and check the prediction error

The following sub-sections are observations on the results of this test.

5.4.1. Downwards spiral

The first step of this test is to try and share the workload by training a moderately accurate model (model 1), then another in the second cycle (model 2). After training the first model, we tested its accuracy and checked the result of the evolution was acceptable. After collecting new data, model 2 was trained with less data, and fewer hidden layers than model 1. Once the model finished training, its predictions were tested on a few samples and the error generated was immense. No accuracy was conserved from model 1.

The same test was conducted, but with a large model 2. This new model had lots of training data, a large layer structure and number of delays, and intensely trained. Even with this much training, the second model's error remained as inaccurate as the previous model 2. Even the first 10 predictions weren't accurate.

We conclude that this issue is a problem of downwards spiralling. If the first model isn't accurate, all future models built on those inaccuracies will only generate more errors. Therefore, we must try with an accurate first model.

5.4.2. Second model error

Now that we have concluded the first model needs to be accurate, we will start with a guaranteed precise model, and train a smaller second model. The following is a collection of hypotheses and results that were conducted.

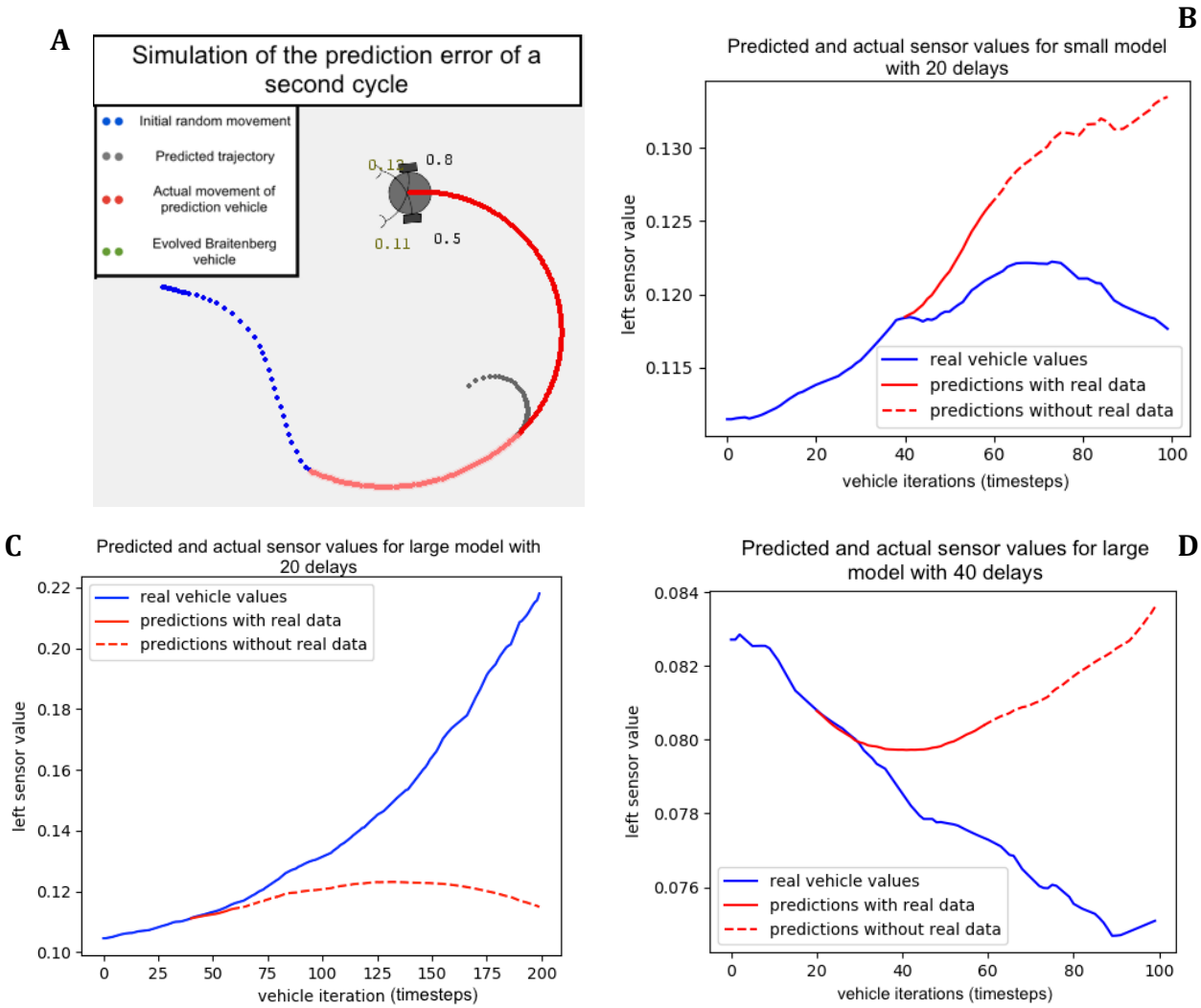


Figure 5-8: Graphs representing the accuracy of predictions for the second model, in the second Sleep-Wake cycle. **A)** Visual of the simulation representing the vehicle with predicted and actual trajectory for the second cycle. The trajectory coloured in pink is the result of the first cycle (correctly heading towards the light). **B, C & D)** Accuracy of the sensory prediction for a test with respectively, a small model and 20 delays, large model and 20 delays, and large model with 40 delays.

Hypothesis 1: A small model with little training data suffices for an accurate model.
Once the bulk of the modelling process is performed by the accurate model, perhaps only a small model is needed for the second cycle. Model 2 will have a restricted environment to model, therefore a better accuracy and low error accumulation. This will allow us to predict accurately far into the future and reach the objective.

Figure 5-8 B proves this hypothesis wrong, where a simple model's predictions for the first 20 time-steps (including real data) are not accurate. Perhaps a bigger model would maintain an acceptable level of accuracy.

Hypothesis 2: A model needs substantial training data and a large structure for maintaining accuracy in the second cycle.
Model 2 was trained with the same amount of training data as model 1, with the same structure, and same epochs. At worst, this model should perform as well as the model of the first cycle.

Once again, the accuracy of the second model drops significantly. However, from Figure C we see that the predictions with real data are considerably more accurate than the predictions without real data.

Hypothesis 3: Models in the second cycle need to be trained with more delays to maintain accuracy.
To test whether the second cycles makes predictions harder without real data, we create the same model but with 40 delays. If this hypothesis is true, the resulting model should maintain acceptable accuracy for all predictions with real data available.
Figure D disproves this hypothesis, as we can see that even when the model still has access to real data, the predictions have a high error. We have now trained larger models for the second cycle with no acceptable resulting accuracy.

Other tests performed include: creating a model that can deal with forwards and backwards movement, and a creation of GA individuals with negative values.

Conclusion:
After having evolved a vehicle on a good model, and obtaining a favourable behaviour, we are unable to train an accurate second model. We hypothesise that the second model isn't capable of learning trajectories when its training data depends on a first model, perhaps because the first model will always have inaccuracies that will be part of the data.

Without acceptable model accuracy, the predictions cannot be trusted when calculating the fitness of an individual; therefore the research of multiple sleep-wake cycles was halted.

6. Discussion

In this section we will discuss the implications and limitations of the results, then mention possible future research.

Our results show the successful building of a model of the environment using a NARX network. Although the model cannot be visually confirmed as correct, we have tested its accuracy by predicting test data and analysing the prediction error. Results show that the model can accurately predict a number of future states of the system. We conclude that NARX networks can successfully model an environment, delivering acceptable performance even when faced with long-term dependencies.

We performed research on the realism of the model training method and reported that the method could be extended to a robot in a real environment, where it would collect data continuously to train a model. The results show that a model trained on such data maintains an acceptable level of accuracy. This shows the technique can be extended to real applications.

From a correct model, we have evolved a control system to reach the objective with a performance significantly better than chance. This result was observed using a highly trained model in a simple environment. For a real world application, the environment would be more complex and therefore the model would also need to be highly trained and proven to be accurate. Extending this test to a real robot would allow us to prove the scalability of this technique to more complex environments.

The behaviour obtained from an evolved control system scaled to reach the original objective. The cycle generalised a control system for a simple problem, creating a brain attracted to light; however the success of this method could depend on the simplistic nature of the problem. Although the cycles created a generalised behaviour scaled from a short prediction, it is possible a more complex task would not provide such successful results. The individuals' genotype only contained 4 genes, which proves to be a simple task. A more complex problem would be more difficult to model and to correctly evolve an optimised control system.

It is clear that world-evolved agents perform better through their ability to test behaviours further into the future than the predicted agent's foresight. However, using predictions significantly reduces the amount of time needed to obtain a better-than-chance answer. If we were to extend this method to a real-world application, we will still have to consider the two factors of time increase. A complex environment would increase the amount of gathered training data needed, increasing the time taken by the Sleep-Wake cycles. However, this would also increase the size of the Genetic Algorithm to evolve an optimal individual, which increases the time of the world-evolved answer. In conclusion, the reduction in time experienced will probably be conserved correctly with more complex environments and tasks.

Our unsuccessful results on the repetition of Sleep-Wake cycles leave us with the question: If the accuracy of the second model could be improved, would the second cycle simplify the problem and provide better solutions? Our hypothesis is that the second model would learn to manipulate the control system appropriately, which would increase its success greatly. More research on multiple cycles is needed, in a different environment, to examine their potential.

6.1. Future work

This project's research can be extended in multiple ways, from exploring the limitations of one cycle to applying this technique to other problems.

To further the work on the success of one cycle, the problem could be modified to be more complex, with multiple light sources or moving lights. This algorithm could be transformed into a search problem, where the objective is to explore the whole environment in hopes of building the perfect model. The algorithm could also be used to create obstacle avoidance behaviour. This technique's robustness can be explored by removing a sensor or motor at an unexpected time.

It would be interesting to apply Sleep-Wake cycles to other known AI problems such as the mountain car problem [23]. Where the stuck mountain car needs to go backwards to gain momentum to overcome the next hill. The OpenAI Gym is a collection of AI problems where users can submit their algorithms [24]. It would be interesting to apply Sleep-Wake cycles to these problems to compare the performance.

7. Conclusion

This project focused on creating more intelligent and adaptable behaviours for robotics, which would perform well in a new environment or unexpected situation. Our method provides an organised optimisation-driven behaviour that aims to let the agent create its own solutions to problems. The Sleep-Wake cycles allow the agent to model its own environment, and then perform offline optimisation to find a suitable behaviour.

We have achieved to train accurate models of the environment with a realistic data collection method. A control system was successfully evolved with a generalised and scalable behaviour, performing better than chance. This method can now be tested on other problems to evaluate its performance on different environments and problems. More research is needed to experiment with the use of multiple Sleep-Wake cycles, which might improve the efficiency of its problem solving abilities.

8. References

[1] Bongard, J. (2010). The Utility of Evolving Simulated Robot Morphology Increases with Task Complexity for Object Manipulation. Artificial Life, 16(3), 201-223. http://dx.doi.org/10.1162/artl.2010.bongard.024

[2] Nolfi, S. (1998). Evolutionary Robotics: Exploiting the Full Power of Self-organization. Connection Science, 10(3-4), 167-184. http://dx.doi.org/10.1080/095400998116396

[3] Sciavicco, L., & Siciliano, B. (2013). Modelling and control of robot manipulators. London [etc.]: Springer.

[4] Arkin, R. (1998). Behavior-based robotics. Cambridge, Mass.: MIT Press.

[5] Clark, A. (1988). Whatever next?. Physics World, 1(12), 44-44. http://dx.doi.org/10.1088/2058-7058/1/12/35

[6] Clark, A (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. Behavioral And Brain Sciences, 36(03), 181-204. http://dx.doi.org/10.1017/s0140525x12000477

[7] Seth, A., Suzuki, K., & Critchley, H. (2012). An Interoceptive Predictive Coding Model of Conscious Presence. Frontiers In Psychology, 2. http://dx.doi.org/10.3389/fpsyg.2011.00395

[8] Friston, K. (2012). Predictive coding, precision and synchrony. Cognitive Neuroscience, 3(3-4), 238-239. http://dx.doi.org/10.1080/17588928.2012.691277

[9] Takens, F. (1980). Detecting strange attractors in turbulence. Groningen: Rijksuniversiteit Groningen. Mathematisch Instituut.

[10] MOZER, M. (1994). Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing. Connection Science, 6(2-3), 247-280. http://dx.doi.org/10.1080/09540099408915726

[11] Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal Of Uncertainty, Fuzziness And Knowledge-Based Systems, 06(02), 107-116. http://dx.doi.org/10.1142/s0218488598000094

[12] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions On Neural Networks, 5(2), 157-166. http://dx.doi.org/10.1109/72.279181

[13] Tsungnan Lin, Horne, B., Tino, P., & Giles, C. (1996). Learning long-term dependencies in NARX recurrent neural networks. IEEE Transactions On Neural Networks, 7(6), 1329-1338. http://dx.doi.org/10.1109/72.548162

[14] DiPietro, R., Rupprecht, C., & Navab, N., Hager, G. (2017). Analyzing and Exploiting NARX Recurrent Neural Networks for Long-Term Dependencies. Unpublished.

[15] Xie, H., Tang, H., & Liao, Y. (2009). Time series prediction based on NARX neural networks: an advanced approach. Machine Learning and Cybernetics, 8, 1275-1279.

[16] Mitchell, M. (1998). L.D. Davis, handbook of genetic algorithms. Artificial Intelligence, 100(1-2), 325-330. http://dx.doi.org/10.1016/s0004-3702(98)00016-2

[17] Harvey, I. (2009, September). The microbial genetic algorithm. In European Conference on Artificial Life (pp. 126-133). Springer Berlin Heidelberg.

[18] Braitenberg, V. (1987). Experiments in synthetic psychology. Cambridge, Mass.: The MIT Press.

[19] Salomon, R. (1999). Evolving and optimizing Braitenberg vehicles by means of evolution strategies. Int. J. Smart Eng. Syst. Design, 2(6977).

[20] Bardella, G., Bifone, A., Gabrielli, A., Gozzi, A., & Squartini, T. (2016). Hierarchical organization of functional connectivity in the mouse brain: a complex network approach. Scientific Reports, 6(1). http://dx.doi.org/10.1038/srep32060

[21] Meunier, D. (2009). Hierarchical modularity in human brain functional networks. Frontiers In Neuroinformatics, 3. http://dx.doi.org/10.3389/neuro.11.037.2009

[22] Fleming, S., & Dolan, R. (2012). The neural basis of metacognitive ability. Philosophical Transactions Of The Royal Society B: Biological Sciences, 367(1594), 1338-1349. http://dx.doi.org/10.1098/rstb.2011.0417

[23] Singh, S., & Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. Machine Learning, 22(1-3), 123-158. http://dx.doi.org/10.1007/bf00114726

[24] OpenAI Gym: A toolkit for developing and comparing reinforcement learning algorithms. (2017). Gym.openai.com. Retrieved 28 August 2017, from https://gym.openai.com