Simulating the behaviour of a Rat in an EPM using an E-Puck

Roshenac B. Mitchell
Heriot-Watt University
rm32@hw.ac.uk

Ioannis D. Papaioannou
Heriot-Watt University
idp2@hw.ac.uk

In this paper we attempt to mimic the contrasting exploratory and fear characteristics of an individual rat placed in an elevated plus maze. This is achieved by evolving the behaviour of an E-Puck robot using an artificial neural network and a genetic algorithm.

Keywords—Elevated Plus Maze, Rat, E-puck, Evolutionary Neural Network, Genetic Algorithm

I. Introduction

In 1955, Montgomery (MONTGOMERY, K C, 1955) set out to investigate the conflicting behaviour of fear and exploratory drive in rats. This is characterised by an approach-avoidance behaviour. Following on from his research there have been many experiments carried out in order to try and recreate his results. Several papers that aim to simulate this research have conducted their experiments in an elevated plus maze (COSTA, Ariadne A et al., 014).

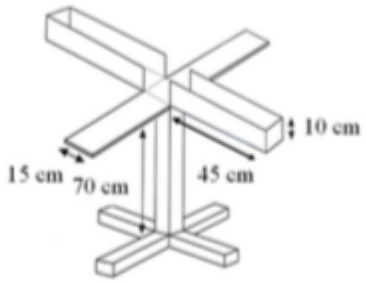

Figure 1: Set up of an elevated plus maze (CHAROENPONG, Theekapun et al., 2012)

An elevated plus maze gets its name due to the fact that it is a platform in the shape of a plus and it is elevated from the floor. As can be seen in *(Fig. 1)*, two of the paths are enclosed by walls in order to prevent the subject falling while the other two paths are open.

In order to conduct this experiment extensively it is possible to recreate this research using robots instead of rats. By creating a robot that mimics the rat's behaviour in this surrounding, it is possible to gain a deeper understanding of how a rat's brain work. The conflicting behaviours should be evident by the approach and avoidance behaviour (MONTGOMERY, K C, 1955) of the robot.

II. Literature Review

This research has been recreated multiple times with each researcher using their own variations in order to create the desired confliction behaviour. For each of the papers reviewed there are a number of key topics that will be focused on. These include the topology, fitness function, genetic algorithm and selection approach.

A. Topology

When creating an artificial neural network, one of the first things to decide on is the topology. This represents the number of nodes (neurons) that will be used. These include the input, hidden and output nodes. As well as this, the number of layers and node connections need to be defined. In both (COSTA, Ariadne et al., 2013) and (SHIMO, Helder K et al., 2010) the Elman architecture is used for their topology. This is defined as a recurrent network where the recurrent signals are passed from the hidden layer. The use of recurrence acts as a memory. These recurrent inputs, which are also know as context layers (SHIMO, Helder K et al., 2010) , allow the previous outputs to be stored. The final inputs are given by the sensor values and recurrent signals from the hidden layer.

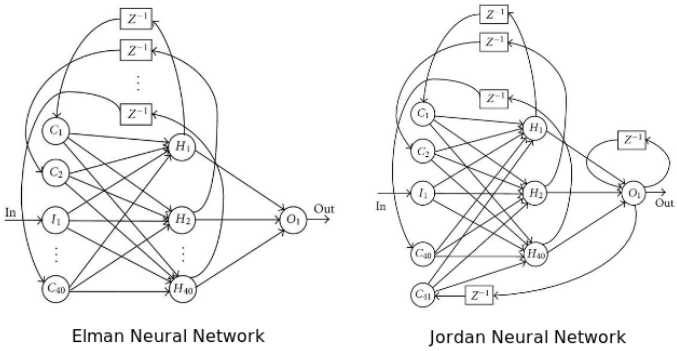

Figure 2: Diagram of Elman and Jordan Neural Networks (TURING FINANCE, 2014)

In both (COSTA, Ariadne A et al., 014) and (COSTA,

Ariadne et al., 2013) their topology results in 4 neurons in the hidden later and 4 neurons in the outputs layer. These output neurons correspond to the following:

- Forwards
- Rotate left 90°
- Rotate right 90°
- Stop

These outputs allow the robot to know what direction to travel in during the next time step. It can be seen from these output nodes that two time steps are needed in order for the robot to travel a full 180 degrees so that it is facing in the opposite direction.

The output of the neurons in (SHIMO, Helder K et al., 2010) are given by the following function where y_i is the output, and w_j is the synaptic weight associated to input x_j (or the output of the neuron j of the previous layer) and i is the current neuron:

$$y_i = \varphi\left(\sum_{j=1}^{p} w_j x_j\right)$$

where φ is the activation function. The activation function is usually a hyperbolic, signal or logistic function. (SHIMO, Helder K et al., 2010)

B. Fitness Function

A fitness function defines the given ideal conditions for an ideal controller. It tends to be split up into two terms. One part rewards the robot for correct behaviour while the other term punishes the robot (COSTA, Ariadne A et al., 014). In the example of the rat in the elevated plus maze (EPM), the reward is equivalent to the curiosity the rat (or robot) portrays. This behaviour will result in the value of the fitness function to increase. The punishment however, is equivalent to the fear the rat (or robot) displays, which in turn will decrease the fitness function (COSTA, Ariadne A et al., 014; COSTA, Ariadne et al., 2013). This fear is show when the rat is in the open arms and it has a higher risk of falling. One suggested fitness function equation can be seen below.

$$f(\mathbf{x}) = \sum_{t=1}^{n} r(\mathbf{x}, p_t) + s(\mathbf{x}, p_t).\beta \ ,$$

Figure 3: Suggested Fitness Function (COSTA, Ariadne et al., 2013)

The same fitness function is used in (COSTA, Ariadne A et al., 014) where t is used in place of p_t. An explanation of the given terms can be seen in the figure below *(Figure. 2).*

Parameter	Definition
$\mathbf{x}$	Evaluated chromosome (weights of the ANN)
t	Time step
$r(\mathbf{x}, t)$	Reward = $\begin{cases} 1 & \text{if } p(\mathbf{x}, t) \text{ was not visited for the agent in the last } \gamma \text{ timesteps} \\ 0 & \text{otherwise} \end{cases}$
$p(\mathbf{x}, t)$	Maze position occupied at time step t
γ	Parameter related to the agent's memory
$s(\mathbf{x}, t)$	Punishment = $\begin{cases} -1 & \text{if } z < \alpha(p(\mathbf{x}, t)) \\ 0 & \text{otherwise} \end{cases}$
z	Random number with uniform distribution in the interval [0,1]
$\alpha(p(\mathbf{x}, t))$	$\begin{cases} \alpha_o & \text{if } p(\mathbf{x}, t) \text{ is in an open arm} \\ \alpha_e & \text{if } p(\mathbf{x}, t) \text{ is in an enclosed arm} \\ \alpha_c & \text{otherwise} \end{cases}$
$\alpha_o, \alpha_e, \alpha_c$	Probability of punishment in an open arm, in an enclosed arm and at the center, respectively
β	Punishment weight (drug dosage balancer)

Figure 4: Explanation of parameters in the fitness function (COSTA, Ariadne A et al., 014)

In (SHIMO, Helder K et al., 2010) an alternative fitness function is used:

$$f(\mathbf{x}) = \frac{f_{\max}}{\sum_{i=1}^{m} a_i} \sum_{i=1}^{m} (1 - p_i(\mathbf{x}))a_i$$

where $p_i(\mathbf{x})$ is the penalty term, a_i is the significance of that term and f_{max} is the maximum fitness value. The fitness function in (SHIMO, Helder K et al., 2010) penalises an individual that keeps returning to the same position as this suggests it is not curious.

C. Genetic Algorithm

Each 'individual' is characterized by its own genotype. This tends to be an array or matrix of number that represent the individual weights for the different neuron connections for each layer of the topology. These weights are then used to define the behaviour of the individual.

The evolution of the genotype and weights of the artificial neural network are controlled by the genetic algorithm. As will be discussed later, there are number of alternative ways to pick the next generation population. In (COSTA, Ariadne A et al., 014), a combination of elitism and tournament is used. This process is then repeated for the number of set generations.

D. Selection Strategies

As discussed in (COSTA, Ariadne et al., 2013) there are three main selection strategies.

- Elitism
- Tournament
- Roulette Wheel

Elitism is achieved by selecting a set number of the best

individuals and copying them to the next generation without any change. In (COSTA, Ariadne A et al., 014) and (COSTA, Ariadne et al., 2013) the two best individuals are chosen.

In the same paper the rest of the new population is selected using *tournament selection*. In tournament selection, the population is paired randomly and each pair is compared according to its fitness score. The one with the highest score is selected to breed (to become one of the parents). Its offspring will then pass to the new generation.

The roulette wheel method increases the probability of the fittest genotypes to be chosen (SHIMO, Helder K et al., 2010). is a probabilistic selection, visualized as a wheel, where each individual covers a percentage of the wheel proportional to its fitness score. Thus as the wheel turns to select the individual for breeding, the ones with the highest score (or surface percentage in the wheel representation) are more likely to be selected as parents.

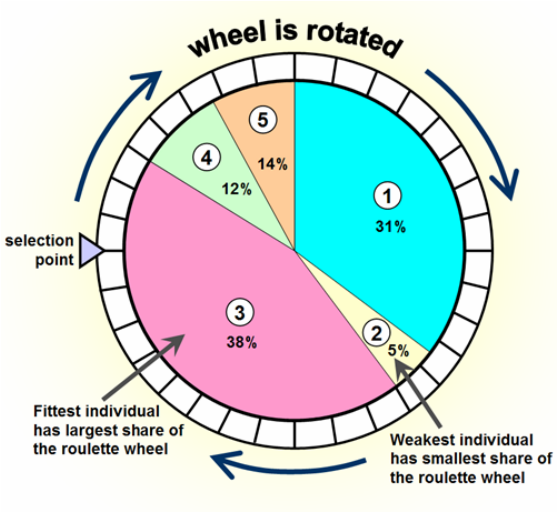

Figure 5: Roulette wheel approach (NEWCASTLE UNIVERSITY , 2016)

It is concluded in (COSTA, Ariadne et al., 2013) that elitism is necessary in order to keep the best individuals in the next population.

III. Design

A. Topology

We are using a Recurrent Neural Network (RNN) using the Jordan architecture. The topology consists of 11 input nodes (8 distance sensors plus 3 ground sensors) one hidden layer of 4 neurons, 2 output nodes, each controlling the speed of a wheel and 2 context units. The output nodes at each iteration are copied to these context units. These are then fed back into the hidden layer on the next iteration. As the network's purpose is to plan actions for the actuators of the E-puck, actions already taken (output) must be remembered to provide acceptable results. As previously explained, the context units act as a short-term memory for the network. All the network is fully connected, providing 52 (11 inputs nodes * 4 hidden nodes + 4 hidden nodes * 2 output nodes) weighted connections between the inputs, hidden layer and outputs, and 8 recurrent weighted connections from the context to the hidden layer.

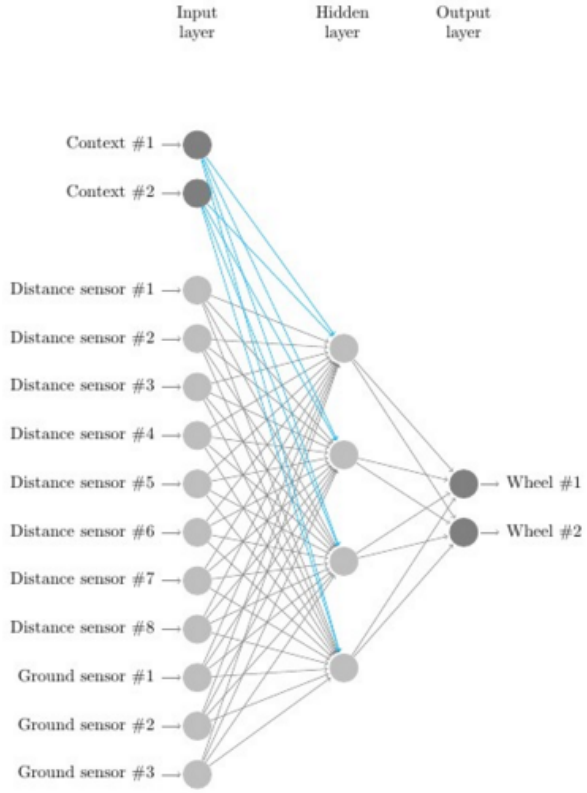

Figure 6: Chosen topology

In the above topology, the two output nodes provide the wheel speed at time *t*, while the context nodes hold the previous output of the network (speed of wheels at time *t-1)*.

The activation function used to fire all nodes is the hyperbolic tangent (*tanh*), as we want the output to receive negative values as well in order for the wheels to be able to turn backwards. The weights of the network are going to be evolved by the genetic algorithm (GA). The activation function for the neurons in the hidden layer is given by the function,

$$h_i^t = f(\sum_i w_{ij} x_i^t + \sum_k c_{ik} y_k^{t-1})$$

while for the output neurons,

$$y_i^t = f(\sum_i w_{ij} x_i^t)$$

where w_{ij} are the weights between the input and the hidden layer, c_{ik} the recurrent connections and $y^{t-1}{}_k$ the output of the network at the previous time step.

B. Genetic Algorithm

1) Genotype

The genotype is encoded as an array of 60 double values, each representing one of the network's weights (including the recurrent connections). The genotype can take values between –1 and 1.

2) Fitness Function

The fitness function is split up between reward and punishment. The robot is rewarded the further distance it travels and it is punished if it falls off the maze, gets and stays near the walls or continues rotating with small circle radius. The fitness score is represented by the following function:

$$f = aD - (bS + cC + dG)$$

Where D denotes the distance travelled reward, S the penalty of the summed values of the distance sensors, C represents the penalty for going in small circles and G is the penalty for going outside the maze. a, b, c and d are weights for each individual reward/penalty (e.g. falling off the edge of maze is penalized much heavier than standing near a wall). The way the fitness function is calculated is explained in more detail in the *"Implementation"* section.

3) Evolutionary Parameters

The genetic algorithm was setup using the following parameters:

Parameter	Value
Population size	50
Max generations	50
Crossover method	2 point crossover
Mutation probability	6%
Mutation deviation	20%
Selection method	Rank Selection
Elitism	10%

IV. Implementation

As with the discussed research it is planned to use an elevated plus maze. Instead of elevating the maze, ground sensors were used to detect whether the robot had fallen off the maze or not.

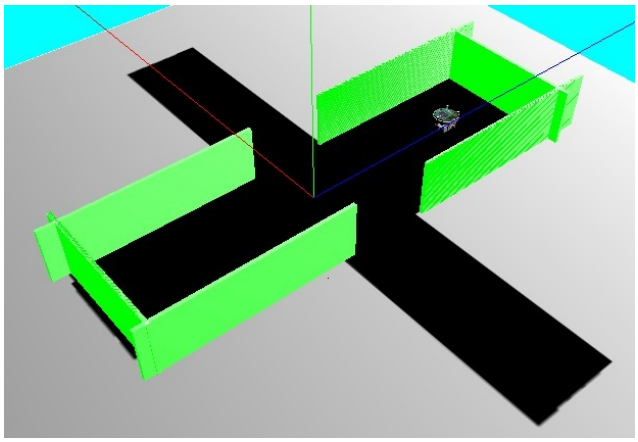

Figure 7: Elevated Plus Maze

Two controllers were created. One that controls the behaviour of the robot (from now on this will be known as the *controller*) and one supervisor controller that is responsible for resetting the position of the robot and running the genetic algorithm (GA) (called *supervisor* from now on). Communication between both the *controller* and the *supervisor* happens via two sets of *emitter/receiver* pairs, each communicating in a different channel.

Instead of transmitting the actual sensory values back to the *supervisor* for evaluation, we are using a reward/punishment system directly on the *controller* itself. Five (5) counters are being used, each representing a variable in the fitness function (getting stuck into a wall is calculated using two counters). Each time the robot follows an action that should be rewarded or penalized, the appropriate counter is increasing by one. Thus, only these counters are send back to the *supervisor* and weighted accordingly they calculate the fitness score of the evaluated genotype. The system works like this:

- Counter 0 (denoted as G in the fitness function and representing the existence of a cliff) is increased every time the robot leaves the black area of the maze.
- Counter 1 (denoted as S in the fitness function) represents the summed value of all distance sensors. If it exceeds a certain threshold, the counter is increased.
- Counter 2 is increased every time the wheels turn backwards, as while using Kinematics, and the robot gets stuck into a wall, the wheels turn back and forth. This counter along with counter 1 penalize this behaviour.
- Counter 3 (denoted as D in the fitness function) counts the times the robot moved between two subsequent time steps. This is done by using the *differential wheels encoders* of the robot. The differential wheel encoders calculate the rotation of the wheel. The value of the encoder continuously increases as the robot moves, but stop increasing when it collides with an object. When the "*kinematics"* property of E-puck is TRUE, then the

wheels are not sliding even if they are turning, making this property extremely useful for detecting whether the robot is stuck against a wall. By keeping track of this value at the previous time step, we calculate the movement of the robot, so every time it has actually moved more than a certain threshold, this counter increases.

- Counter 4 (denoted as *C* in the fitness function) counts the times the robot has moved in small circles. This is achieved by measuring the difference between the wheels. Every time step this difference is over a threshold the counter increases, while every time it is below that, it decreases. As while going in a circle the difference between the wheels will remain constantly over a certain value (according to the radius of the circle), the counter will increase in extremely high numbers, thus detecting that the robot is actually moving in a small circle.

The *supervisor* starts the evolution process where a new genotype (the weights of the network) is created and sent through the emitter to the *controller* in order to be evaluated. Every 120" a new genotype is send to the *controller.* The *controller*, while the simulation is executed, is constantly running, and constantly checking its receiver for new genotypes. The *controller* runs in time-steps of 32ms. As the emission / transmission is carried out using a buffer in a last in first out (LIFO) manner, synchronisation must be achieved between the two controller's transmissions. To achieve this, an internal counter in the *controller* is decremented on each time-step counting down until 120" have passed (120000 / 32 = 3750 steps). Once it reaches 0, the following things are happening in the *controller*:

1. The *reward / punishment* counters are transmitted back the *supervisor* for evaluation.
2. The wheel encoders, reward/punishment counter and sensory input variables are reset, in order to start measuring these values for the next evaluation cycle.
3. The synchronization counter resets to the maximum number of steps.

Once this data is received by the *supervisor*, the fitness function is applied and the score of that particular genome is calculated. Then the *supervisor* takes the next genome of the population and send it to the *controller* and the same procedure continues.

V. DISCUSSION

When trying to achieve a solution to this elevated plus maze task there are two main approaches that could have be applied. These are a behaviour-based or an evolutionary-based robotics approach.

A. Behaviour-Based Robotics Approach

Behaviour-based robotics relies on a set of fixed behaviours provided to the robot (FLOREANO, Dario and Mondada, Francesco). The theory, goals and methodology have to be thought of by the user as well as the layer architecture (FLOREANO, Dario and Mondada, Francesco).

B. Evolutionary-Based Robotics Approach

Evolutionary-based robotics relies on learning and adaption using genetic algorithms and artificially neural networks (FLOREANO, Dario and Mondada, Francesco). By applying a set of rules, these allow the program to evolve in a way to achieve the desired behaviour. By developing the controller, it allows the program to reproduce only the fittest chromosomes with regards to the set fitness criteria (FLOREANO, Dario and Mondada, Francesco).

As the program is not hard coded this may result in some unexpected behavioural side effects. However, in general evolutionary based robotic approach is often used as artificially neural networks are good at simulating artificial evolution (FLOREANO, Dario and Mondada, Francesco).

VI. EVALUTION

After evolving the best controller, it was then run for 5 minutes with the aim of completing the required task.

The amount of time it took to complete the maze can be seen in the table below. The vary times are due to slight repositioning of the robot before running the simulation.

	Time			
	First Run	***Second Run***	***Third Run***	***Total Time (Average)***
Task	2.14	2.20	2.16	2.17

As can be seen from the table, the robot successfully travelled the maze. However, the developed robot was seen to be rather fearful and never ventured on to the open fingers of the maze.

It was found that by adding the hidden layer in the topology allowed the e-puck to learn and evolve quicker and better than without this layer. The possibility of the sigmoid function was discussed for the activation function but as was stated previously it was found that the hyperbolic tan function allowed E-puck the ability to drive backwards. the maze.

Additionally, the fitness function was an extremely important factor when evolving the e-puck. Many sensor values were considered including the distance, ground and light sensors as well as the wheel encoding and speed. After analysing the performance of the different genotypes relative to the sensors we were able to narrow it down to the mentioned elements: ground sensor, sensor sum, wheel speed, whether it is going in a circle and encoder value i.e. distance. These results were then put in a graph and the weights were analysed in order to find optimum values the represented the performance of these genotypes.

Performance							Fitness
	fall	sum	speed	reward	circle measure	circle	
weights	1.5	0.6	0.5	0.3		0.5	
hit the wall	0	3388	1808	462	-3567	0	72.018
hit the wall	0	3415	1869	476	-3439	0	71.593
fell off the maze	7389	0	0	3747	-3749	0	0.406
the maze to the other while hitting walls	0	2385	1265	1475	-3505	0	83.79
driving in circles	0	132	69	3618	3479	3479	92.322
side of the maze to the other	0	189	71	3112	-3247	0	107.847

Figure 8: Measured punishment and reward values

By having the high weighting for when the robot fell, the controller quickly evolved to no longer fall off the maze. Additionally, while we didn't want to robot wheels going backwards while hitting a wall this resulted in the reverse functionality of the robot being punished which made the robot more fearful as it couldn't move backwards off an open maze platform.

VII. CONCLUSION

These results show that why it is possible to create a robot that simulates a rat, there are many different variations possible. In order to find the best method a number of different topologies, fitness function and activation functions would need to be tested in comparison to once another in order to see which one performs the best. During the course of this project it was made clear that designing the fitness function correctly was the most crucial part of the implementation as this is what dictated the behaviour of the robot. Additionally, synchronisation between the *supervisor* and *controller* was vital in order to update the fitness function correctly.

VIII. BIBLIOGRAPHY

CHAROENPONG, Theekapun, Yuttachon PROMWORN, Wongwit SENAVONGSE et al. 2012. An experimental setup for measuring distance and duration of rat behavior. *In*: *Biomedical Engineering International Conference (BMEiCON)*. IEEE.

COSTA, Ariadne A, Silvio MORATO, Antonio C ROQUE, and Renato TINÓS. 014. A computational model for exploratory activity of rats with different anxiety levels in elevated plus-maze. *Journal of neuroscience methods*. **236**, pp.44-50.

COSTA, Ariadne, Patrícia A VARGAS, and Renato TINÓS. 2013. Using explicit averaging fitness for studying the behaviour of rats in a maze. *Advances in Artificial Life, ECAL*. **12**, pp.940-946.

FLOREANO, Dario and Francesco MONDADA. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot.

MONTGOMERY, K C. 1955. The relation between fear induced by novel stimulation and exploratory drive. *ournal of comparative and physiological psychology*. **48**(4), pp.254-260.

SHIMO, Helder K, Antônio C ROQUE, Renato TINÓS et al. 2010. Use of evolutionary robots as an auxiliary tool for developing behavioral models of rats in an elevated plus-maze. *In*: *InNeural Networks (SBRN), 2010 Eleventh Brazilian Symposium*. IEEE, pp.217-222.

TURING FINANCE. 2014. *10 misconceptions about Neural Networks*. [online]. [Accessed 30 March 2016]. Available from World Wide Web: <http://www.turingfinance.com/misconceptions-about-neural-networks/>