Coursework Intelligent Robotics (2022)

Lia Perochaud, Victoria Giner, and Vikram Singh

***Abstract*—This document outlines the implementation of BBR (Behaviour Based Robotics) and evolutionary approach to create a controller for an e-puck robot in a T-Maze. The evolutionary approach consists of a Genetic algorithm which is used to train a feed forward neural network which ultimately controls the movement of the robot across the maze. The simulations have been done using Webots software (R2022b).**

Based on the simulations, a comparision between the methodologies have been done which revolves around ease of construction, perfromanfe and processing power requirements.

***Index Terms*: behaviour-based robotics, evolutionary robotics, genetic algorithm, feed forward neural network, robot-environment interactions.**

I. INTRODUCTION

THIS document discusses the usage of BBR (behavioural based robotics) and evolutionary robotics using genetic algorithm in a T-Maze path search algorithm.

BBR behavioural robotics represents robots capable of exhibiting complex behaviours by correcting their behaviours thanks to the numerous sensors and actuators they possess such as Ground sensors, proximity sensors, left wheel motors and right wheel motors in our case. Most behavioural robots like the ones we are using in this project have predefined basic functions such as recharging their battery or avoiding obstacles. They then acquire new behaviours by learning from their past experiences. The information is extracted from the robot sensors who use this information after to correct their behaviour according to the environnement.

For the task 2, we use the evolutionary approach which is another style of approach based on selection and heredity. Evolutionary robotics assumes that robot samples are randomly generated. Each robot has individual characteristics. However, the worst performing robots are replaced by robots with better performance (mutations or combinations of old robots). This evolutionary composition system allows the robot to find the most optimal solutions to a given problem that a human would not necessarily find. The number of robots can also come from a subset and then rejected or not thanks to the fitness function.
Indeed, the desired response would be an action to be performed. However, in the evolutionary algorithm, the fitness curve simply informs about the failure or success of the controller in question. The time taken to converge to the optimum solution highly depends on parameters such as mutation rate, population, and crossover selection of the initial pollution. [1]

Problem Statement

In this simulation, an e-puck robot will start in the T-maze, and it must reach the reward zone. The reward zone is different based on the occurrence of the “black mark” in the path avoiding obstacles. To complete this task, 2 different controllers are create based on Behaviour Based Robotics and evolutionary learning approach.

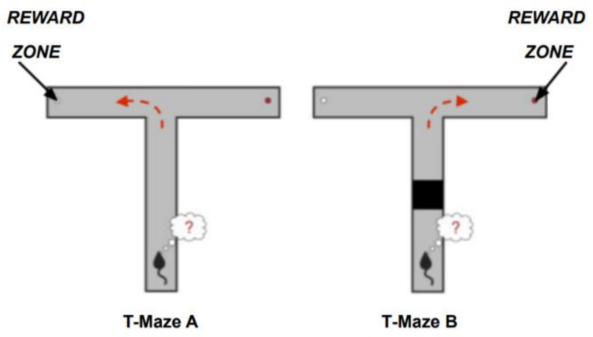


Figure 1 - T-Maze diagrams (from Coursework_Description2022_Final.pdf)

.

II. METHODS AND IMPLEMENTATION RATIONALE TASK 1

Introduction

The goal of the task 1 is to implement a Behaviour Based Robotics controller on the e-puck robot.
To accomplish this task, we used a subsumption architecture.
“*The key idea of levels of competence is that we can build layers of a control system corresponding to each level of competence and simply add a new layer to an existing set to move to the next higher level of overall competence*” …” *We call this architecture a subsumption architecture.”* [2]
In this architecture, the lower layers do not need the upper

layers to function, while the upper layers are applied on top of the lower layers and need them. The goal is to rank behaviours in order of significance. Higher layers can also affect lower behaviour. With this knowledge, we developed our own subsumption architecture, defined several behaviours for our robot, and assigned different priorities to the behaviours.

Methods

In first here is the summarized of what the robot needs to do:

- Go to the end of the T-maze without touching the walls
- Turn right our left at the junction after detecting black mark

And what the robot has for interact with its environment:

- Ground sensors
- Proximity sensors
- Left wheel motors
- Right wheel motors

With this, we can describe the fundamental robot behaviours to: Moving forward

Since we don't want our robot to get stuck in a wall, the next behaviour it needs to learn is to recognize and avoid obstacles.

It was difficult to categorize the next two behaviours in order of importance.

On the one hand we have the behaviour of turning at the intersection, and on the other hand we have the behaviour of detecting the black mark on the ground.

Technically, the robot can turn at the junction without knowing whether there is a black mark or not, because it has a safety behaviour -> always turn left.
So the behaviour at the intersection would be more "important".

Detection of the ground mark is more crucial in our situation since, to turn, we must first determine whether there is a black mark, so detecting the ground mark is more important

And the last behaviour is to stop the robot at the end of the maze.

The entire architecture (See *Figure 2*) is designed in the to summarize:

0. Go forward.
1. Avoid Obstacles.
2. Detect the mark on the floor.
3. Turn at the intersection.
4. Stop at the end.

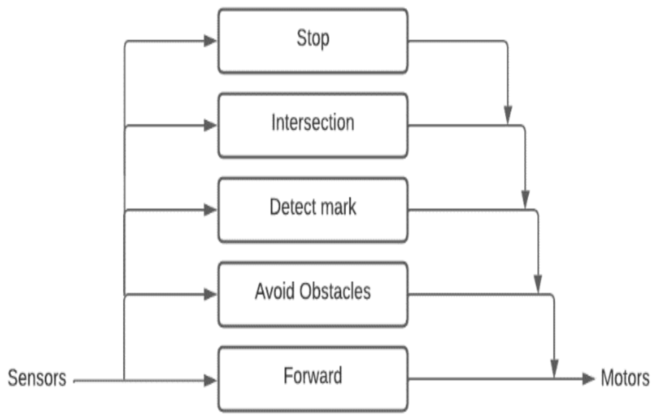


Figure 2 - Subsumption architecture of our robot

Implementation

The first step to implement this part on a controller in the software Webots, is to get the input of all the sensors of the robots, to retrieve all the information that the robot has on its environment.

Next step is to implement the comportment, we write one function for each behaviour.

0. Go forward:

For this comportment, the right and left motors wheels are set to the maximum speed of the robot.

The maximum speed of the e-puck robot is 6.28 rad/s [3]

1. Avoid Obstacles

We concentrate on using the robot's proximity sensors ps1 and ps6 (*see Figure 10*) to identify obstacles in the case of task 1's wall. We are unable to use the ps2 and ps5 sensors because of how close the walls are, which causes the robot to become "afraid of walls" and stop or turn inward as a result.

A proximity sensor returns a value between 0 and 1500, so we decided to detect a wall when its return is 200 or more.

Therefore, when the robot detects a wall, it should move to the opposite side, so if the wall is on the right, the speed of the left motor is set to -1.5 and the speed of the right motor is set to 1.5, and the opposite if the robot detects a wall on the left. The speed is set to only 1.5 and not the maximum speed to prevent the robot from spinning around, as it doesn't have much room to move, so it quickly detects another wall.

2. Detect the mark on the floor (see *Figure* 3)

The three ground sensors that positioned beneath the robot must be used for this stage. The ground sensors always return a reading of more than 800, as the map's floor is entirely white. The robot's "mark" index value is set to 1 when the three sensors are equal to or below 400(see

Figure 4).

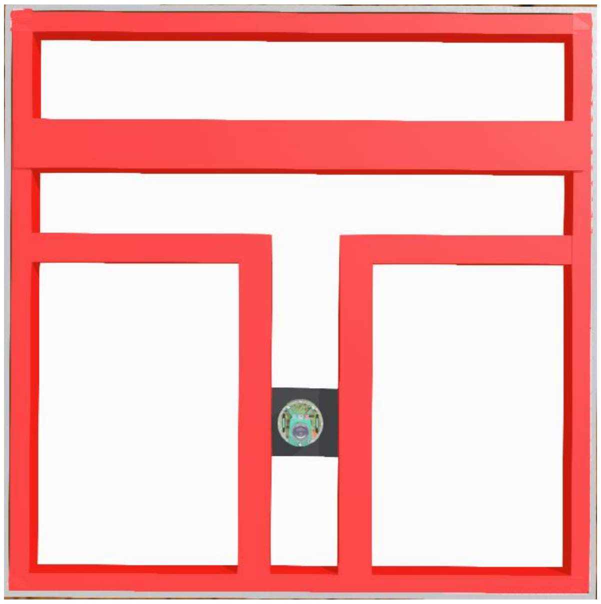
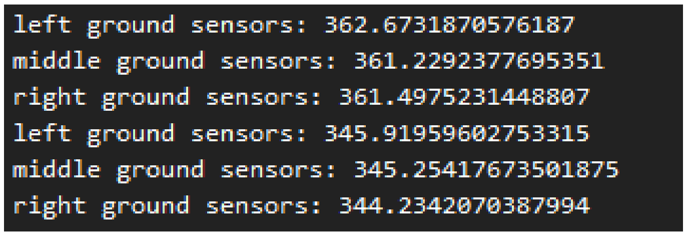


Figure 3 - Robot on the black mark



```
left ground sensors: 362.6731870576187
middle ground sensors: 361.2292377695351
right ground sensors: 361.4975231448807
left ground sensors: 345.91959602753315
middle ground sensors: 345.25417673501875
right ground sensors: 344.2342070387994
```

Figure 4- Values return by the ground sensors when robot is on the black mark

3. Turn at the intersection (see *Figure 5*)

The "turn at the intersection" behaviour is the "most" complex, it uses three steps:

- In first, detect of there is any walls on the side with ps2 and ps5 (see *Figure 10*), we check that the two proximity sensors return a value under than 70 (see *Figure 7*)
- After that we check the "mark" index value, if this value is equal to 1, the robot must turn to the tight, otherwise it must turn to the left.
- To turn to the right, the speed of the left motor is set to 1.5 and the speed of the right motor is set to -1.5. As for the "avoid obstacles" part, we do not want the robot to go into the wall.

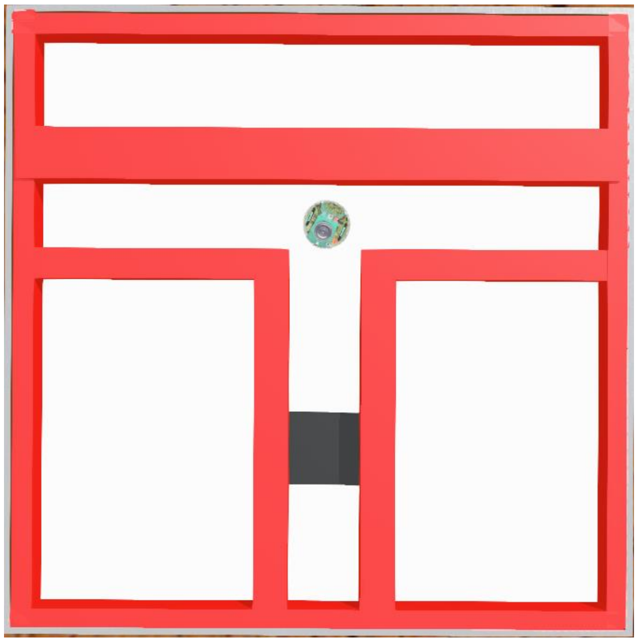
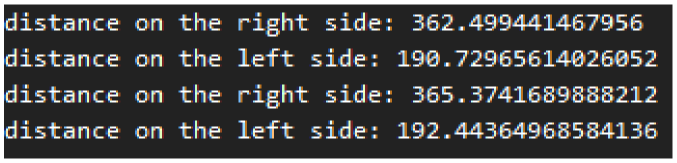
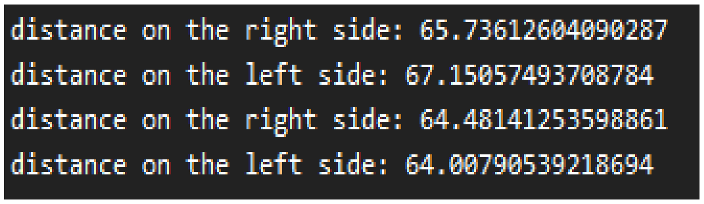


Figure 5 - Robot in the intersection



```
distance on the right side: 362.499441467956
distance on the left side: 190.72965614026052
distance on the right side: 365.3741689888212
distance on the left side: 192.44364968584136
```

Figure 6- Distances display with a wall on each side (ps2 is right and ps5 is left)



```
distance on the right side: 65.73612604090287
distance on the left side: 67.15057493708784
distance on the right side: 64.48141253598861
distance on the left side: 64.00790539218694
```

Figure 7 - Distances display when the robot is in the middle of the intersection (ps2 in right and ps5 is left)

4. Stop at the end (*see Figure 8*)

This is the final behaviour of the robot, the goal being to detect the end of the maze. It is defined by the fact that the robot is inside a corner, so for this we focus on the proximity sensor values ps0 and ps7 to check what is in front of the robot, and we check the ps2 and ps5 values to make sure the robot is in a corner.

So, when the robot as the wall in front of it the value for ps0 and ps7 need to be higher than 900. And the value for both sides needs to be higher than 200 (see *Figure 9*)

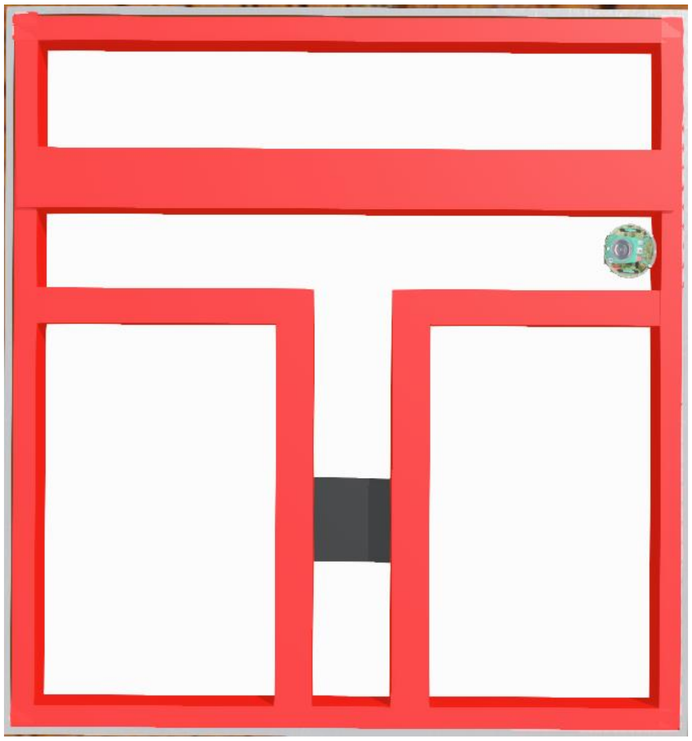
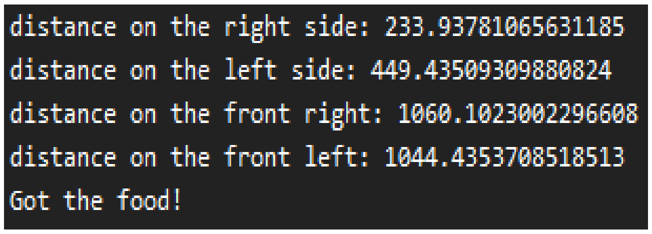


Figure 8- Robot in the corner at the end



```
distance on the right side: 233.93781065631185
distance on the left side: 449.43509309880824
distance on the front right: 1060.1023002296608
distance on the front left: 1044.4353708518513
Got the food!
```

Figure 9 - Distances value when the robot is in the corner (ps2 is right side, ps5 is left side, ps0 is front right and ps7 is front left)

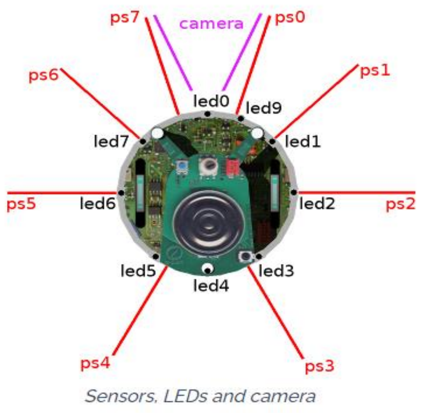


Figure 10 - Robot's sensors from https://cyberbotics.com/doc/guide/epuck

III. TASK 2

Introduction

This task uses an evolutionary technique of automated learning of the e-puck robot. It involves using a feed-forward multi-layer perceptron to control the movement of the robot (velocities of the left and right motors). In order to train the neural network, genetic algorithm is implemented. The weights and bias of the neural network are modified/updated based on the optimization done by the genetic algorithm. [4] The weights and bias are coded as real parameters vectors (using lists in python). The sensor's data is passed on to the multi-layer perceptron and the resulting output is multiplied by 3 and given as the speeds of the left and right motor of the e-puck robot.

The robot starts from an initialized position and then starts moving. During the run, as per the movement of the robot, based on output of MLP, its fitness is measured after every timestep (32 ms). After the run, *reward* is added to the fitness based on the final location, resulting in the final fitness of the genotype in that generation.

MLP Architecture

The architecture used for the multi-layer perceptron is 11-8-6-2. (See *Figure 11*)

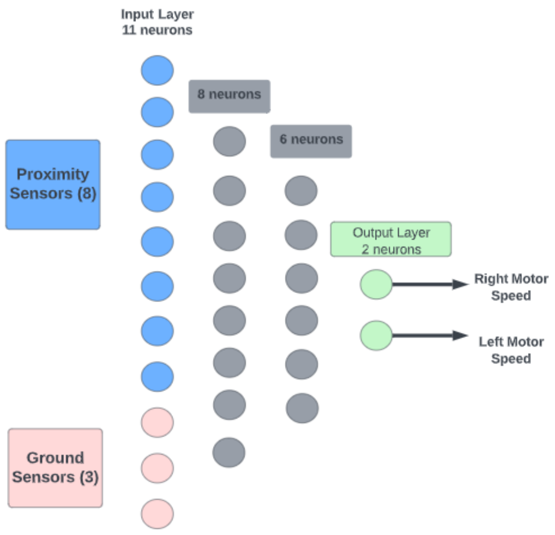


Figure 11 - MLP architecture

The network architecture has been designed based on the sensors and actuators of the e-puck robot. For the simulation, values of 8 proximity sensors and 3 ground sensors are passed to the input layer of the neural network and the feed forward output of 2 output nodes is used to set the speed of the bot. The network is using tanh as activation function among layers.

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

The network has 2 hidden layers containing 8 and 6 neurons respectively. The simulation was first performed with a single hidden layer, but to get acceptable performance, at least 25 nodes were required, hence decided to use more than 1 hidden layers. [5]

Genetic algorithm architecture

The genetic algorithm used to train the neural network is defined using below mentioned characteristics:

- Real valued coding of the data (weights of the neural network) is done.
- Tournament selection of genotypes for creating new generation. Selection of parent is done among 5 genotypes and the best one is selected.
- One point crossover is implemented by bisecting the chromosomes at the middle and generating new child.
- Uniform Mutation has been used as an operator which modifies the value of the gene with a uniform random value selected between the upper and lower bounds. As the activation function is tanh, output of the neural network is bounded between [-1, 1], so the minimum and maximum bounds of the gene values is also [-1,1].
- Population size is 50.
- Number of elites is 5 (10% of the population). This value is chosen to adequately take the best performing genotypes to the next generation. [4]
- Crossover rate used of 0.8.
- Mutation rate used of 0.1.
- Number of generations to be trained = 80.

Standardization of sensor values

The sensor outputs of the e-puck are sensitive, hence are required to be bounded and standardized.

Default range of proximity sensors = [0, 4095]

Default range of ground sensors = [0, 4095]

To facilitate training of the neural network, minimum and maximum output of the sensors are bounded as follows:

Proximity sensors = [200, 600]

The proximity sensor sensitivity increase is very steep for the bot after 700, when reaching near an obstacle. Based on the sensors' values near the turn and to provide ample time for the robot to rotate/ change its direction, a slightly aggressive bound has been put at 600. The upper bounds of 700,750,800,900 were also tested and bound of 600 resulted in better movement (less wobble and turning) of the bot. If using an upper bound of 800, the bot takes more generations to converge to the solution, and the final time taken to complete the task was higher by 8 seconds.

Ground sensors = [0, 800]

As for the ground sensors, when the bot is not on the black mark the sensor output value around 290, and on the mark, it fluctuates between around 800.Hence, the max value is set to be 800 so that after scaling it comes around 1.

After bounding the minimum and maximum value of the bot, the output of all the sensors is scaled to [0,1].

Fitness Function

The movement fitness function is divided into 3 sub-parts namely forward fitness, avoid collision fitness and spinning fitness. At the end of the turn, reward is added to it.

Forward Fitness

This aspect of the total fitness depends on whether the robot is not stationary and is moving in the forward direction.

$$Forward\ Fitness = \frac{velocity\ left + velocity\ right}{2}$$

velocity left and *velocity right* refers to the velocity of the corresponding motors of the robot. As the velocity is the output of the multi-layer perceptron (bounded between -1 and +1).

The fitness will be maximum when both wheels are moving forward and are at full speed. This fitness function will ultimately force the bot to move and increase the velocity. Max value = 1, Min value = -1

Avoid collision fitness

This function is used to stop the bot from colliding with obstacles, in this case side and front walls.

$$Avoid\ collision\ Fitness = 1 - 1.7 * (\max(proximity\ sensors\ values))^2$$

like

$$y = 1 - \alpha x^2$$

This is convex function which decrease exponentially as sensor value is increased (showing that obstacle is in proximity).

The value of 1.7 as α is chosen because it puts the value 0 at almost x = 0.5, which will correlate with the choice of bounding of proximity sensors. Max value = 1, Min value = -0.7

Avoid Spinning fitness

This function is used to stop the bot from spinning at a place. Absolute spinning happens when the velocities of the bot are as follows:

$$velocity\ left = -(velocity\ right)$$

i.e., both the wheels are rotating in opposite direction with equal velocity.

The function used in the controller is:

$$|(velocity\ left - velocity\ right)| > 1.8\ , \quad fitness\ - 1$$

$$|(velocity\ left - velocity\ right)| < 1.8 \quad fitness\ + 1$$

This is the absolute difference of the left and right wheels of the bot. By using this constraint, the bot will not rotate without moving forward, and this is not too strict, which can impact the movement of the bot at the turn. Max difference between speed is 2.0. Max value = 1, Min value = -1

Now to combine all 3 fitness, below mentioned function is used:

$$Combined\ Fitness = 5 \times Forward\ fitness + 7.5 \times Avoid\ collision\ fitness + 7.5 \times Avoid\ spinning\ fitness$$

The Rationale is to balance out all the movement fitness functions, so that it is not susceptible towards collision and spinning ultimately leading to a more reliable and robust bot.

Reward Function

At the start of the run, the bot is placed at an initial location and as the bot moves, based on the distance and direction travelled by the bot, a reward is added to it.

As per the axis of the T-maze, positive x direction is in the right and positive z is down. Centre of the axis is on the long side of T-maze.
Reward for map with black mark:

$$Reward = 25 \times displacement\ in\ xdirection + 75 \times displacement\ up$$

The idea behind is every movement towards "Up" (negative z axis) increases the reward of the bot as it is moving closer to the target zone.

Reward for map without black mark:

$$Reward = -25 \times displacement\ in\ xdirection + 75 \times displacement\ up$$

Fitness after adding reward and then final fitness:

$$fitness = (combined\ fitness + reward)/10$$

$$Final\ Fitness = e^{fitness}$$

This is the final fitness of the genotype for the run. The reason why displacement in x direction is multiplied by 25 (less than the coefficient of direction z (up) is during testing, if the priority of x-displacement is higher in reward function, the bot will get stuck at the turn, as it tries to turn too quickly into the corner and collides with the side wall.

Results and Discussion

Below is the final table of the best run timings for both the tasks (Time is in seconds). For the evolutionary approach, there is a single collision while turning, but the robot recovers from it:

	Time in **seconds**			
	First Run	Second Run	Third Run	Average
Task 1 T-Maze A	9	8.7	8.8	8.8
Task 1 T-Maze B	9	9	8.8	8.9
Task 2 T-Maze A	20	20.7	20	20.2
Task 2 T-Maze B	19.3	19.8	19.6	19.56

Table 1 Final Output

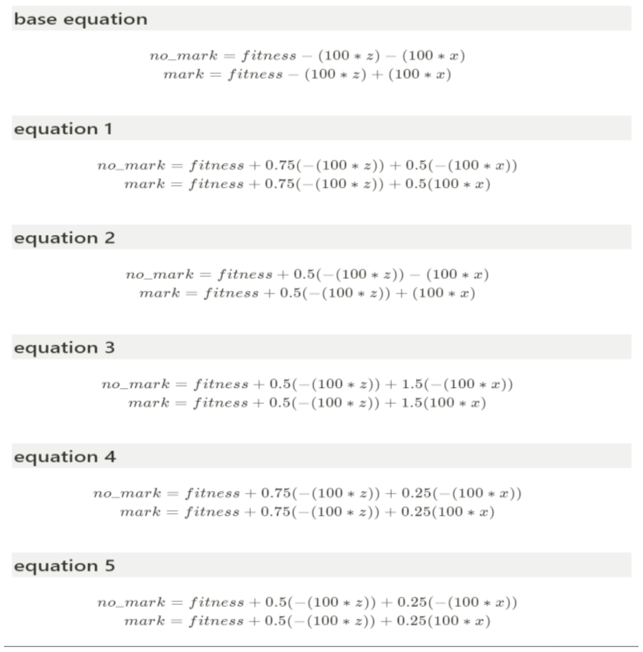
BBR Results

BBR results are automatically easier to obtained due to the type of architecture, so we just need to adjust the value of the detectors to generate the good behaviour at the right time. To adjust it we just printed the value like in *Figure 4, Figure 6, Figure 7, Figure 9*. After that we were able to detect the information at the right time.

Fine tuning of Fitness Parameters

To fine tune the reward and fitness function of the Task 2, different combinations of collision fitness and reward for moving "up" and in x-direction have been tested and documented.

Below mentioned are the reward function testing performed, with "*mark"* signifying reward of maze which has a black mark on the path. "fitness" mentioned is the combined fitness before adding the reward.



base equation

$$no\_mark = fitness - (100 * z) - (100 * x)$$
$$mark = fitness - (100 * z) + (100 * x)$$

equation 1

$$no\_mark = fitness + 0.75(-(100 * z)) + 0.5(-(100 * x))$$
$$mark = fitness + 0.75(-(100 * z)) + 0.5(100 * x)$$

equation 2

$$no\_mark = fitness + 0.5(-(100 * z)) - (100 * x)$$
$$mark = fitness + 0.5(-(100 * z)) + (100 * x)$$

equation 3

$$no\_mark = fitness + 0.5(-(100 * z)) + 1.5(-(100 * x))$$
$$mark = fitness + 0.5(-(100 * z)) + 1.5(100 * x)$$

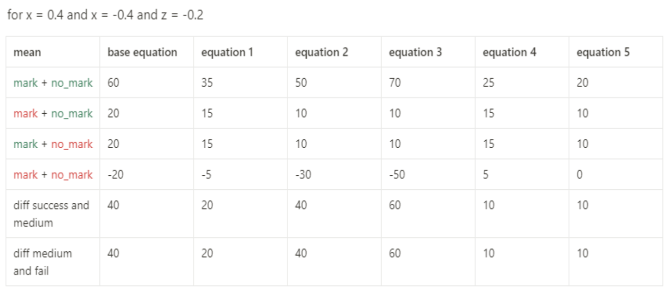
equation 4

$$no\_mark = fitness + 0.75(-(100 * z)) + 0.25(-(100 * x))$$
$$mark = fitness + 0.75(-(100 * z)) + 0.25(100 * x)$$

equation 5

$$no\_mark = fitness + 0.5(-(100 * z)) + 0.25(-(100 * x))$$
$$mark = fitness + 0.5(-(100 * z)) + 0.25(100 * x)$$

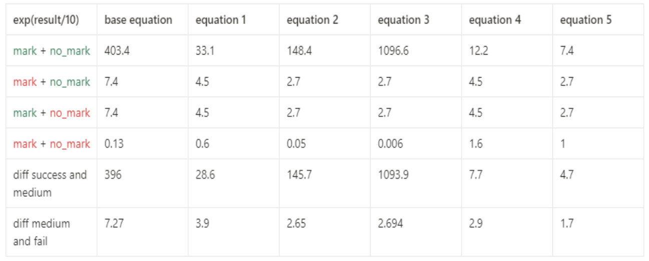
We tried to avoid the local minimum where the robot turns always on the same side, so test the different equation with the maximum value for x and z and try to avoid the local minima:



for x = 0.4 and x = -0.4 and z = -0.2

mean	base equation	equation 1	equation 2	equation 3	equation 4	equation 5
mark + no_mark	60	35	50	70	25	20
mark + no_mark	20	15	10	10	15	10
mark + no_mark	20	15	10	10	15	10
mark + no_mark	-20	-5	-30	-50	5	0
diff success and medium	40	20	40	60	10	10
diff medium and fail	40	20	40	60	10	10

Application of the exponential function on the result / 10 to "punish more the local minima":



exp(result/10)	base equation	equation 1	equation 2	equation 3	equation 4	equation 5
mark + no_mark	403.4	33.1	148.4	1096.6	12.2	7.4
mark + no_mark	7.4	4.5	2.7	2.7	4.5	2.7
mark + no_mark	7.4	4.5	2.7	2.7	4.5	2.7
mark + no_mark	0.13	0.6	0.05	0.006	1.6	1
diff success and medium	396	28.6	145.7	1093.9	7.7	4.7
diff medium and fail	7.27	3.9	2.65	2.694	2.9	1.7

Test of the equations on the robot:

	Eqn 2	Eqn 3	Eqn 4	Eqn 5
Average time on 5 tests	36	27	23	20

So, we kept equations 4 and 5 because best timing was achieved.

At this time good speed and good accuracy were found on 20 tests but robot hits the back wall all the time and slide across it, so try to change combined fitness:

- $Combined\ Fitness = 5 \times Forward\ fitness + 10 \times Avoid\ collision\ fitness + 7.5 \times Avoid\ spinning\ fitness$

Results	Eqn 4	Eqn 5
Time in sec	43	25
Accuracy on 20 tests	60% to the right 0% to the left	85% tot the right 100% to the left

- $Combined\ Fitness = 4 \times Forward\ fitness + 8 \times Avoid\ collision\ fitness + 6.5 \times Avoid\ spinning\ fitness$

Results	Equation 4
Time in sec	40
Accuracy on 20 tests	100% to the right 0% to the left

- $Combined\ Fitness = 7.5 \times Forward\ fitness + 12.5 \times Avoid\ collision\ fitness + 7.5 \times Avoid\ spinning\ fitness$

Results	Eqn 4	Eqn 5
Time in sec	24	50
Accuracy on 20 tests	100% to the right 65% to the left	100% to the right 0% to the left

- $Combined\ Fitness = 7.5 \times Forward\ fitness + 20 \times Avoid\ collision\ fitness + 7.5 \times Avoid\ spinning\ fitness$

Results	Eqn 4	Eqn 5
Time in sec	-	-
Accuracy on 20 tests	0% both side Never finishied	0% both side Never finished

- $Combined\ Fitness = 3.75 \times Forward\ fitness + 7.5 \times Avoid\ collision\ fitness + 3.75 \times Avoid\ spinning\ fitness$

Results	Eqn 4	Eqn 5
Time in sec	25	24
Accuracy on 20 tests	100% on both side	90% to the right 95% to the left

So, we kept the 5th changes for combined fitness function with equation 4 because it obtained 100%

accuracy on 20 tests, but it continues to hit the back wall and slide against, so that is not really what is required.

Fine tuning of robot sensors' sensitivity

After modifying the reward and fitness functions, the sensitivity of the robot sensors is optimized.

- Result for distance min = 200, distance max = 600, ground min= 0, ground max = 800

Results	**Equation 4**	**Equation 5**
Time in sec	34	30
Accuracy on 20 tests	100% to the right 100% to the left	95% to the right 95% to the left

- Result for distance min = 200, distance max = 800, ground min= 0, ground max = 800

Results	**Equation 4**	**Equation 5**
Tim in sec	29	31
Accuracy on 20 tests	100% to the right 100% to the left	95% to the right 95% to the left

Inference after these runs is to abandon eqn 5 of fitness and reward combination and finalise eqn 4.
Then we change ground sensor minimum value to 100 and maximum value to 800, and we retried test on distance sensor, with equation 4

Results	**Min 200 Max 600**	**Min 200 Max 700**	**Min 200 Max 800**
Time in sec	-	28	22
Accuracy on 20 tests	0% both side	100% to the right 100% to the left	95% to the right 100% to the left

Results	**Min 270 Max 600**	**Min 270 Max 700**	**Min 270 Max 800**
Time in sec	24	-	26
Accuracy on 20 tests	95% to the right 85% to the left	0% both side	95% to the right 100% to the left

Only good accuracy of 100% on distance sensor 270/700 with eqn 4 and ground sensor at 100/800

So, we rerun all the tests with 100% accuracy since the beginning of the tests and choose the one with least collision i.e equation 4, proximity sensor limits of 200/600 with ground sensor limits of 0/800.

These robot parametrs are used to test the genetic algorithm parameters. The idea was to find a parameters set where the robot is constantly getting good performance and the number of generations is lowest.

After all the testing, GA final Parameters are:
- Crossover rate = 0.8
- Mutation rate = 0.1
- Population = 50
- Generations = 80
- Number of elites = 5

Difference between BBR and ER implementation

Both of these methodologies in their traditional form share some common traits like both cannot perform active learning (GA learns over the iteration but can be implemented after it has completed running) and both are highly effective in robot path planning problems. But, there are a lot of differences in terms of implementation as well.

Behaviour based techniques require the creation of different control/behaviour layers and in contrast ER relies on calculating the benefit(fitness) of a particular position and orientation of the robot.

By running GA, some other pathways/solutions can be discovered while BBR needs to setup all the types of interactions beforehand.

For simple problem statements like moving in a T-Maze, following a line, on a small scale, BBR can perform better and faster. Like in this experiment, BBR performs better because the choice of decisions and behaviours are manageable. All the behaviours are easily incorporated in the BBR controller. On the other hand, ER requires a lot of parameters tuning and testing to find an optimum solution. The time and processing power required to complete all the generations and testing, is very large compared to creating a behaviour-based controller.

The difficulty of using BBR increases dramatically with the increase in different interactions of the robot and the increase in complexity of the environment. All the interactions/behaviours will be required to make into layers and the communication between the layers needs to be properly setup. This also leads to the imagination of the person designing the controller. The difficulty of using ER on a large-scale problem statement also increases but it makes more sense because there is high probability that a new kind of behaviour can come out of the simulations.

Impactful Parameters of GA Implementation

The performance of the genetic algorithms highly depends on the parameters used (encoding, fitness, crossover rate, mutation rate, population size, etc).

Encoding of the search space [6] is the first step of defining a problem. In this experiment, we encoded the search space as a real valued vector, and bounded the output of ML to [-1, 1], resulting in maximum speed during the simulation be to 6 rad/s, well within the maximum feasible speed of the robot of 6.28 rad/s.

Fitness will play a major part in deciding whether the algorithm will ultimately find the optimum solution or not. The fitness function must clearly and accurately define the value of every point in the search space, based on the expected functionality of the robot. As seen in last section, modifying the fitness function vastly affects the performance. [1]

When a GA is used to train a neural network which controls the movement of a robot, the sensitivity of the sensors becomes a major factor while defining the fitness function. So, the sensor readings must be bounded in a way that they do not impact the learning of the neural network.

The combination of mutation, crossover rate, elites and size of population affects the efficiency of the learning process. If the number of elites is low, the chances of passing best information from one generation to another decreases. Mutation and crossover cause new points in the solution space to be searched and paves the way for improvement over the generations. Improper values of these parameters will result in slowing down of the increase of fitness. A balance of these parameters must be identified for the problem statement which ultimately results in less generations for the algorithm to complete.

Conclusion

Both mythologies have their pros and cons in their implementation as well their results.

As per the simulations, it is clearly evident that both the methodologies are efficient in creating the controller and at the same time differ with each other with their implementation in a considerable way. For small-scale problem statements, where the goal is known, and the behaviors can be categorized and implemented easily, behaviour based controller will be very fast and easy to test as compared to GA based evolutionary methods.

Genetic algorithm based evolutionary techniques are highly effective and versatile but require much more computation power and time. Hence, the use of the methodology must be decided based on the requirements of the problem.

References

[1] D. Goldberg, Genetic algorithms in search, optimization and machine learning, Massechusetts, 1989.

[2] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal of robotics and automation*, vol. 2, no. 1, pp. 14-23, march 1986.

[3] "https://cyberbotics.com/doc/guide/epuck," [Online].

[4] B. N. M. J.-B. E. A. E. (. Doncieux Stephane, "Evolutionary Robotics: What, Why, and Where to," *Frontiers in Robotics and AI*, vol. 2, 2015.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems," 1989.

[6] J. C. G. Randall D. Beer, "Evolving Dynamical Neural Networks for Adaptive Behavior," 1992.

[7] J. N., "Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis. Adaptive Behavior.," 1997.

List of Tables