Behavior Based Robotics (BBR) and Evolutionary Robotics (ER) approaches analysis on a e-puck robot using Webots simulator

Professor: Dr. Patricia Vargas

Jiancheng Zhang
MSc Robotics
School of Electrical Electronic & Computer Engineering
jz76@hw.ac.uk

Janany Kamalanadhan
MSc Robotics
School of Electrical Electronic & Computer Engineering
jk2023@hw.ac.uk

Haoqin Zhao
MSc Robotics
School of Electrical Electronic & Computer Engineering
hz2022@hw.ac.uk

Abstract

This paper describes the implementation result of the BBR approach and ER approach bio-inspired algorithm developed to control an e-puck robot using Webots Simulator.
In order to do this, we implemented different robot controller approches and tested them in a simulated arena in Webots simulator. First, we added an additional obstacle avoidance layer to our robot using BBR approach whilst following a line, then we created an arena and added light in the corners that would "switch on" on a random time sequence for the robot to perform phototaxis and finally we added an ER approach to our robot to avoid obstacle, follow the line and getting back to it when off-track. After successfully met the objectives, we tested them by adding obstacles in our arena and analyzed our results. Although our experiment only consists mainly in navigating and avoiding obstacle, we did observe some characteristic for both approaches and sorted a better understanding of these.

Key words: Behavior Based Robot, Evolutionary Robotics, Robot Controller, , Finite State Machine, Intelligent Robotics.

Introduction

Robotics could be controlled in various ways, which includes using manual control, wireless control, semi-autonomous (which is a mix of fully automatic and wireless control), and fully autonomous (which is when it uses AI to move on its own, but there could be options to make it manually controlled). Here, we are using semi-autonomous controlled robots and there are different approaches to develop those.

Traditional Approach

[1] In the traditional approach, the model block stores a complete and accurate model of the real world that the robot operates within. This may be accomplished in a variety of ways, but there is usually some type of geometric coordinate system involved for mobile robots to establish the current state and predict the future state. Sensor data must be calibrated and accurate for the state to be precisely determined. Ideal state information is either stored or computed based upon the sensor data set. Deviations from the ideal state are errors that are passed onto to the plan block in

order that appropriate control measures can be taken to minimize these errors. Controllers or actuators must also be precise and accurate to ensure that the corrective motions are done in strict conformance with the commands created by the plan and action blocks.
The traditional approach often involves a lot of computer memory to store or compute the necessary state information. It is slower than the BBR approach, which in turn makes the robot less responsive.

Behavior-Based Robotics Approach

Behavior-based robotics (BBR) is an approach to control robots. Its origins are in the study of both animal and insect behaviors.

real-world robots can use a limited real-world representation or model, in order that it should be adequate to carry out the desired requirements without reliance on an overly detailed world model.

Instituting a behavior prioritization scheme has an unintended positive outcome. Robots usually operate with the same behavior, such as moving forward. All behaviors acting in sequence should strive to maintain this normalcy. Only when environmental conditions warrant should a behavior direct the robot to deviate from the normal. When deviations happen, the higher-priority behaviors take control and try to restore normalcy.

BBR also incorporates long-term progress indicators to help avoid a "looping" situation, as might be the case where the robot continually bounces between two barriers or is locked into a wall corner. These long-term progress indicators effectively generate a strategic trajectory in which the robot moves in a general direction or path. When progress is impeded, a different set of behaviors is selected to return to the normal state.

Evolutionary Robotics Approach

[2] The idea of using evolutionary principles in problem solving dates back to the dawn of computers (Fogel, 1998), and the resulting field, evolutionary computing (EC), has proven successful in solving hard problems in optimization, modeling, and design (Eiben and Smith, 2003). Evolutionary robotics (ER) is a holistic approach to robot design inspired by such ideas, based on variation and selection principles (Nolfi and Floreano 2000; Doncieux et al., 2011; Bongard, 2013), which tackles the problems of designing the robot's sensory apparatus, morphology, and control simultaneously.

The evolutionary part of an ER system relies on an evolutionary algorithm. The first generation of candidate solutions, represented by their codes, the so-called "genotypes," are usually randomly generated. Their fitness is then evaluated, which means translating the genotype – the code – into a phenotype – a robot part, or its controller, or its overall morphology; putting the robot in its environment; letting the robot interact with its environment for some time and observing the resulting behavior; and computing a fitness value on the basis of these observations. The fitness value is then used to select individuals to seed the next generation. The selected would-be parents undergo randomized reproduction through the application of stochastic variations (mutation and crossover), and the evaluation–selection–variation cycle is repeated until some stopping criterion is met (typically a given number of evaluations or a predefined quality threshold).

1. Experiments: tasks, results and discussion

Task 1 : BBR - Obstacle avoidance and light following.

For task1, we have to use Behaviour-Based Robotics approach to implement behaviour that avoids obstacles while following the line. [3] Brooks (1986) paper shows that each module could be a finite state machine. Based on this idea, we can append the requirements into different states, each state represents a certain action. At the same time, the robot can only have one state. There are several advantages with using a finite state machine; the robot will keep one state for a specific time and will transfer from one state to another under certain conditions. We can add more behaviour layer by adding states and easily analyse the robot's behaviour by looking at the states graph.

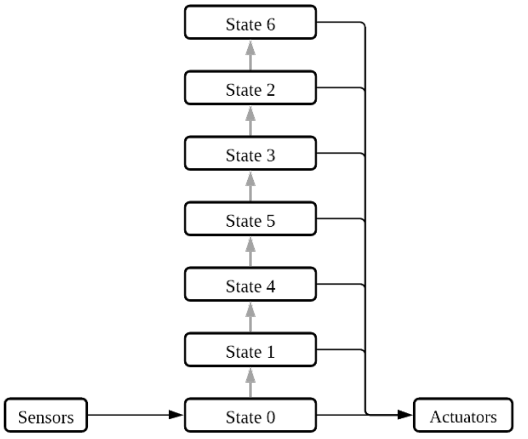


Figure 1 : states description

For this task, we can part it into seven states (figure 1). Each state relates to a specific velocity behaviour. Although the different state has a different priority which is suppression, there is no inhibition in velocity control.

State 0	Detecting an obstacle in the front
State 1	Spinning in-place to adjust moving direction parallel with the obstacle, which is following a wall.
State 2	Following the wall
State 3	Reached an external corner of the wall, has to turn in order to follow the wall again
State 4	Following the line
State 5	Spinning in-place to find the line
State 6	Wandering

Figure 1: the priority of states, note that the lower level, the higher priority.

Thought Process on the experiment:

We have to make sure each state contains one action, if there are more actions in one state, the major problem is when some unexpected situation happened during these sequential actions then some of them cannot be done properly. When decided the priority of each state, we first group actions together. After detecting an obstacle, the robot must spin in-place, otherwise it will collide with the obstacle; state 0 and 1 form a group and 0 comes first. State 2 and 3 is a group due to that the robot must meet a wall first, and then a corner of the wall. State 4 and 5 is a group since if there is no line, the robot does not need to spin to find a line again. State 6 is wandering if there is no obstacle and no line detected. State 0, state 1 and state 6 respectively are the highest and lowest priority and can be easily decided. For each group: states 4, 5 and states 2,3, if the latter group has higher priority than the former group when the robot detects a line during following the wall, it will still follow the wall rather than the line which doesn't meet the requirements. Hence, the correct priority order is shown in Figure 1.

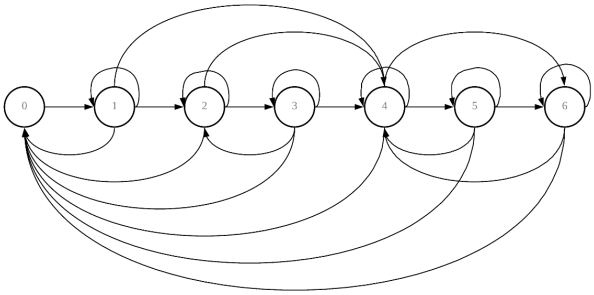


Figure 2: finite state machine, the robot can start from state 0, 4, 6

Figure 2 shows the transfer routes between states. Every state iterates themselves, except state 0. The reason being: the robot will stay in a state for a time period.
- State 0 has the highest priority, no matter the current action, if the robot meets an obstacle in the front, it must avoid it immediately. Consequently, every state can transfer to state 0 except itself.

```
If max(self.inputs[3],self.inputs[4],self.inputs[7],self.inputs[8])>=0.15
```

Inputs [3], [4] are the right-front proximity sensors, inputs [7],[8] are the left-front proximity sensors. The threshold is 0.15 which means the output value of the sensors is 150. The reason why we chose 0.15 as a threshold is that after several experiments, we found that the robot cannot be too close or far from the obstacle.
If the obstacle surface is perpendicular to the robot movement vector and with a higher threshold, from figure 3 we can see that ps0 and ps7 has a small angle against the real front, therefore the true distance between the robot and the surface is greater than what we got from the output value of proximity sensors. Due to the fact that the output value of sensors is very sensitive, and after spinning in-place (figure 4), the ps2 or ps5 will be perpendicular to the surface, the output of ps2 or ps5 will be much greater than the threshold. This will cause the problem that it is hard to follow the wall with a relative same distance because slight distance changes lead to a huge sensor output value change.
If we have a lower threshold, the robot will lose the wall easily when following it due to the noise of sensor outputs. We will record which side has an obstacle in this state.

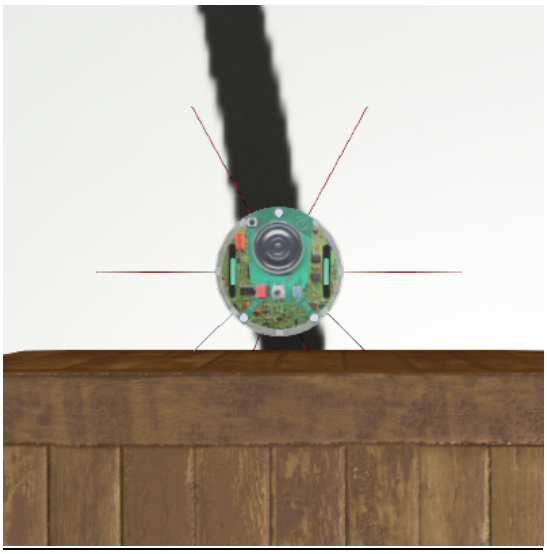


Figure 3: face at an obstacle

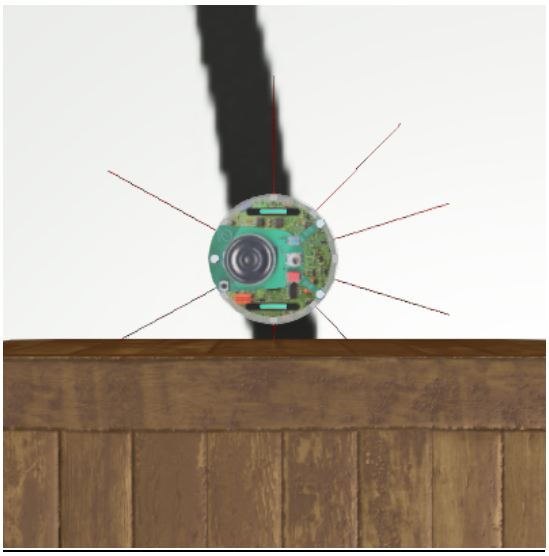


Figure 4: parallel to an obstacle

- State 1: When the robot transfer into state 1, the basic idea is that with the robot spinning, the distance between ps2 or ps5 and the obstacle will increase first and decrease, there will then be a peak of sensor output value. State 1 is the output value increasing stage, once the output value starts decreasing, the robot will transfer to other states.

```
elif self.current_state == 1 and (self.inputs[5] > self.distance_max or self.inputs[6] > self.distance_max):
    # if the obstacle is on the right-hand side
    if self.side == "right":
        if self.inputs[5] > self.distance_max:
            self.current_state = self.FSA[1]
            # updating the maximum value of the distance between proximity sensor 2 or 5 with the obstacle
            self.distance_max = self.inputs[5]
```

if the new output value is greater than the old one, we can stay in state 1. We will focus on the side that has an obstacle and ignore another side. The reason is output value has noise, although there is no obstacle, the output value still slightly greater than the previous from time to time. We will update the maximum distance each time.

- State 2: Once the output value of ps2 or ps5 decreases, the robot will move forward while keeping the output value of ps2 or ps5 stay the same as the maximum distance by controlling the velocity of left and right.

```
if self.inputs[5] < self.distance_max:
    self.velocity_left = self.rotation_speed / 1.8
    self.velocity_right = self.rotation_speed / 2
elif self.inputs[5] >= self.distance_max:
    self.velocity_left = self.rotation_speed / 2
    self.velocity_right = self.rotation_speed / 1.8
```

The robot always goes forward with a winding line, and this is why the threshold cannot be too low in state 0.

- State 3: when the output of ps2 or ps5 is suddenly below 80, probably the robot reaches an external corner. In order to keep following the wall again, the robot has to turn to the same side as the wall. Here we use the self.side variable to indicate turn left or right.

```
elif (self.current_state == 2 or self.current_state == 3) \
        and ((self.side == "right" and self.inputs[5] < 0.08) \
            or (self.side == "left" and self.inputs[6] < 0.08)):
    self.current_state = self.FSA[3]
……
elif self.current_state == 3:
    # turn right immediately
    if self.side == "right":
        self.velocity_left = self.rotation_speed / 4
        self.velocity_right = 0.1
```

when the ps2 or ps5 detect the wall again, the robot will transfer into state 2.

- State 4: If any of the output values of the ground sensor are smaller than 380, the robot will start following the line. If the left ground sensor has the smallest output value, the robot will go left. If the middle ground sensor has the smallest value, it will go forward. If the right ground sensor has the smallest value, it will go right.

- State 5: When all the output values of ground sensors are greater than 600 and the robot is following the line, the robot will spin in place trying to find the line again. The way of spinning depends on which side there is an obstacle or not. In figure 5, if the robot spins clockwise, it will face the obstacle again. So, the robot has to turn to the opposite side of the obstacle. Again, we will use the self.side variable here.

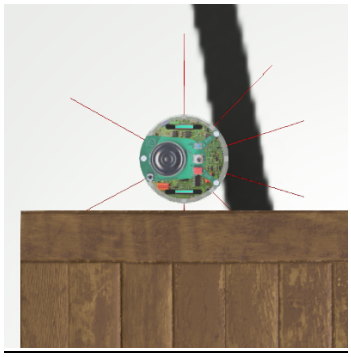


Figure 5: one possible case

- State 6: if there is no obstacle, the e-puck robot is just wandering.

```
elif self.current_state == 6:
    self.velocity_left = np.random.rand()
* 2
    self.velocity_right = np.random.rand()
* 2
```

Results:

the experiment will run in the environment shown in figure 6. The surface is 2.20m × 1.80m in size with 0.05m high and 0.025m thick side-walls. There are two obstacles, one is a bigger cube, the other is a smaller cylinder. The robot will run for one round then get back to where it started.

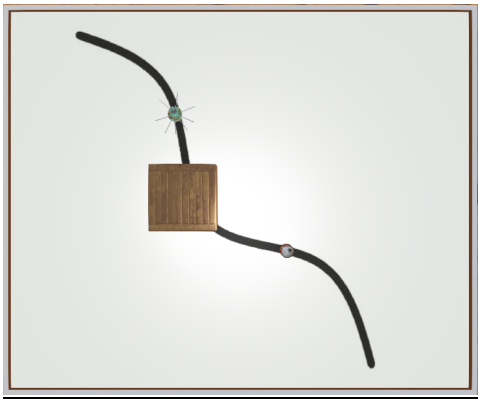


Figure 6: test area (top view)

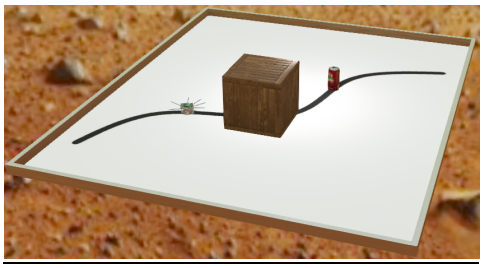


Figure 7: test area (3D view)

The robot finishes the task successfully, and there is 2474 number of couple of iterations average results in total, with timestep = 32ms and duration = 158 second. From figure 8, we can know see that the minimum output value of the ground sensor is approximatively 300, although there is noise on the upper bound which is around 340. We noticed that the vibration of the ground sensor value is larger when the robot is navigating a white floor than on a black one. When the robot is following the obstacle, the velocity will change in a large range which causes the robot to have a pitch angle, and this makes the ground sensor receive darker or whiter colour than normal.

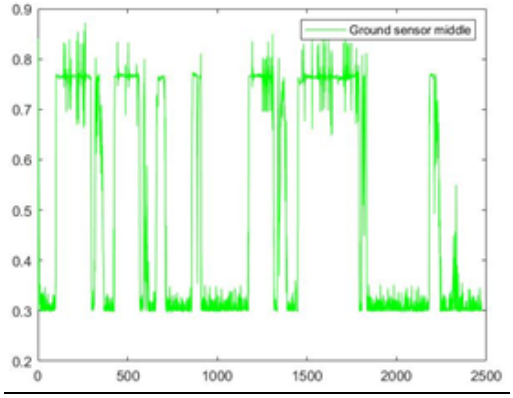


Figure 8: middle ground sensor

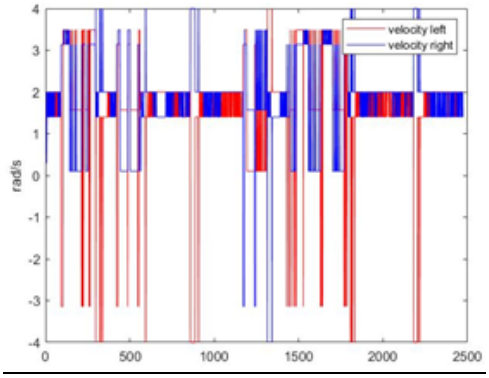
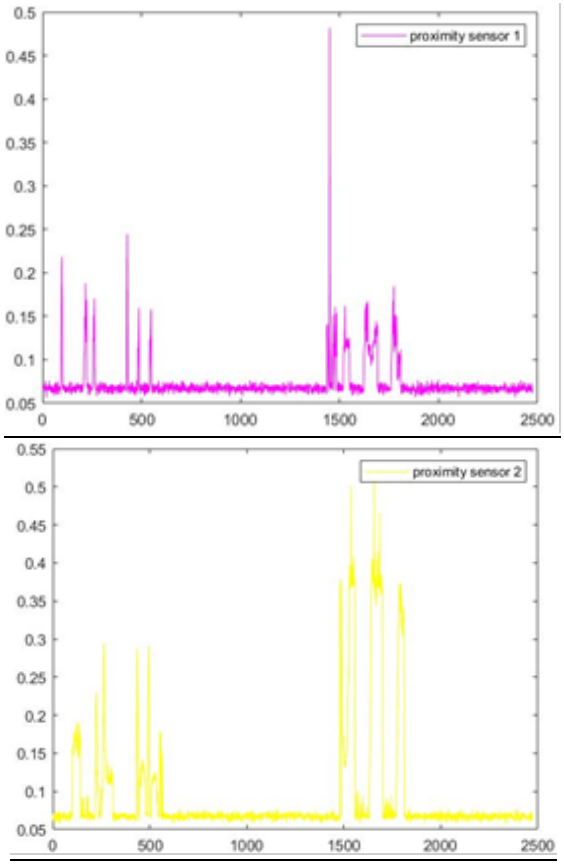


Figure 9: left and right velocity

From figure 10 and 11, we can see that the upper bound of the output value with the noise of the proximity sensor is near 80 when there is no obstacle. There is a pulse output value near 1500 iteration, that is caused by the robot touching the corner of the cube when it follows the line. Since there is no proximity sensor in the right front, when ps1 receive the data, the robot already collided head-on with the cube. As we can see, we can get a sensible output value range of ps2 or ps5 by adjusting the threshold in state 0.



[4] Brooks, R. (1986)

Task 2 : BBR - Obstacle avoidance and light following.

In this task, we have to implement a function of following lights and avoiding obstacles. Noted that for the given floor texture, there is a black circle near the wall, which indicates that the robot cannot move in black area. The major difference between this task and previous task is how to decide when the robot should stop avoiding the obstacle and start following the light. The original idea is that the obstacle can shade of the light source, in such way that thelight sensors cannot detect light when the robot is behind the obstacle; and when the light sensors receive light again, the robot can stop avoiding the obstacle. However, in this simulator, as the objects are transparent, the light sensor can still receive lights when the robot behind the obstacles. To solve this problem, we have to divide following lights state into two parts. One is for no obstacle situation following lights, another is for deciding when can the robot stop following the wall and start following lights.

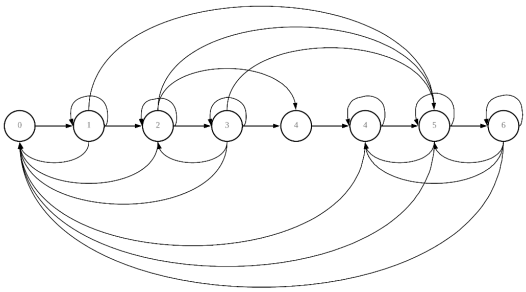


Figure 10: finite state machine, the robot can start from state 0,4,5,6

Tought Process on the experiment :

For this task, we can reuse most state from previous task, but we have to change the meaning of state 4 and 5 as well as the priority order of each state. We found the state 4 should has two different priorities, although the actuator behaviour is the same. For the higher priority state 4, it will stop the robot following the obstacle, since it's

priority higher than state 2 and 3. For the lower priority state 4, it only works when there is no obstacle, therefore its priority is only higher than state 6.
We noticed that the state 5 has the highest priority, which means no matter what the robot is doing, avoiding an obstacle or following lights etc., if it touches the black area, the robot has to spin in-place until the ground sensors stop detecting black.

State 0	Detecting an obstacle in the front
State 1	Spinning in-place to adjust moving direction parallel with the obstacle, which is following a wall.
State 2	Following the wall
State 3	Reached an external corner of the wall, has to turn in order to follow the wall again
State 4	Stop avoiding the obstacle, start following the light source
State 4	Following the light source
State 5	Spinning in-place to avoid step into black area
State 6	Wandering

Figure 11 : sensor description

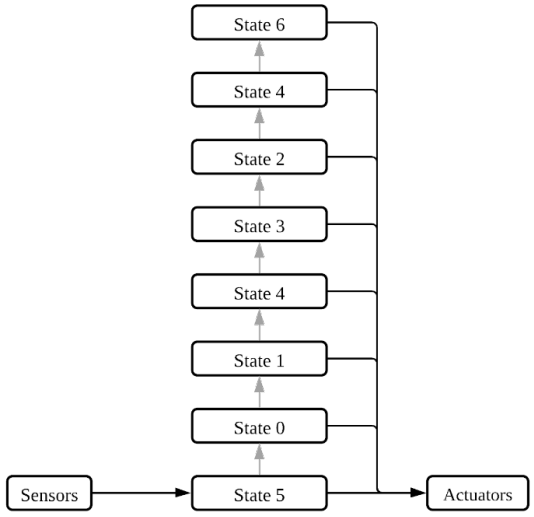


Figure 12: the priority of states, note that the lower level, the higher priority.

Most of the states follow the same steps, we are going to introduce the states that differs.

- State 4: Considering one possible case, figure 11, the light source located in right bottom corner. We can see that there are four light sensors that detects the light which the output value is zero, although the robot is behind the box. Even though the objects are transparent, this does not mean we cannot use the output value of light sensor as condition and this is the only way that the robot detects the lights. We need to find a connection between the light sensors' outputs and the position of the obstacle.

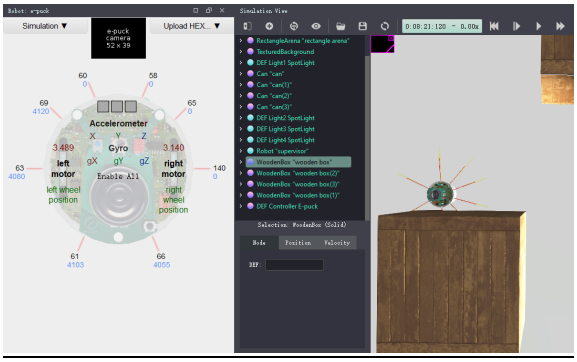


Figure 13: one possible case

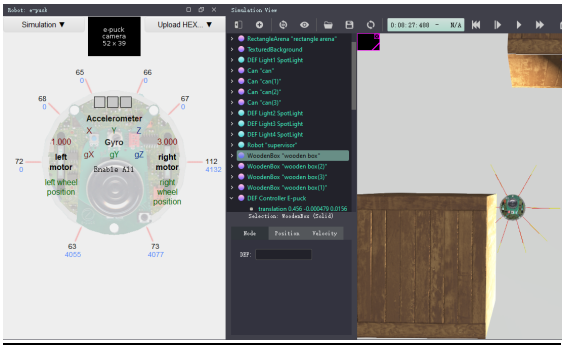


Figure 14: another possible case

Considering the robot keeps following the box (figure 13). We found that the box still located at the right-hand side of the robot, but the light sensor 5 output went from 4080 to zero eg. if the obstacle on the right-hand side, but the left side light sensor's output is zero, the robot can start following the lights. Considering that the light sensors are very sensitive, we can say if the output of light sensors is 0, there is a light source.

```
elif (self.current_state
== 2 or self.current_state == 3) \
      and ((self.side
== "right" and self.inputs[8] == 0)
           or (self.side
== "left" and self.inputs[5] == 0)):
   self.current_state = self.FSA[4]
```

For the actuator, we group light sensors by side.

```
if count_right == count_left:
   if self.inputs[6] == 0 and self.inputs[7]
== 0:
      self.velocity_left = -4
      self.velocity_right = 4
   else:
      self.velocity_left = 4
      self.velocity_right = 4
elif count_right > count_left:
   self.velocity_left = 3
   self.velocity_right = 1
elif count_right < count_left:
   self.velocity_left = 1
   self.velocity_right = 3
State 5: Here we use self.side to decide
whether spinning clockwise or
anticlockwise.
if np.max(self.inputs[0:3]) < 0.55:
   self.current_state = self.FSA[5]
……
elif self.current_state == 5:
   if self.side == "right":
      self.velocity_left = -4
      self.velocity_right = 4
   elif self.side == "left":
      self.velocity_left = 4
      self.velocity_right = -4
```

Results:
The size of arena is the same as task 1. There are four spotlights in each corner, and every 45 seconds, one light will turn on by random whilst rest of lights turn off, the robot position will reset to the centre of the arena, coordinate is (0,0,0).

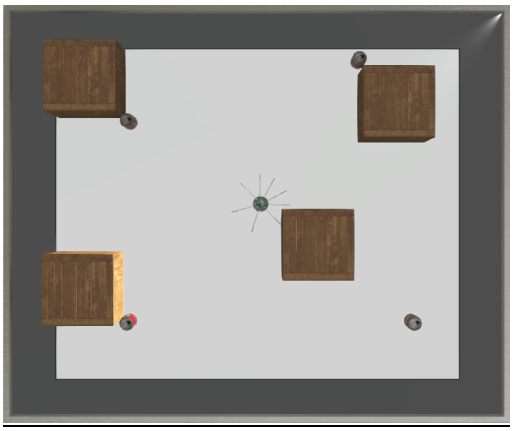


Figure 15: test area (top view)

The total number of iterations is 36640 and time step = 32ms, duration = 1172.48 seconds. The robot performed correct behaviours in different situations. From figure 16 and 17, we can see that the output value is mainly 1 or 0. Compare ls5 and its neighbour sensor ls6, we found that although these two sensors are close to each other, the number of zero outputs of ls5 more less than the ls6 did.

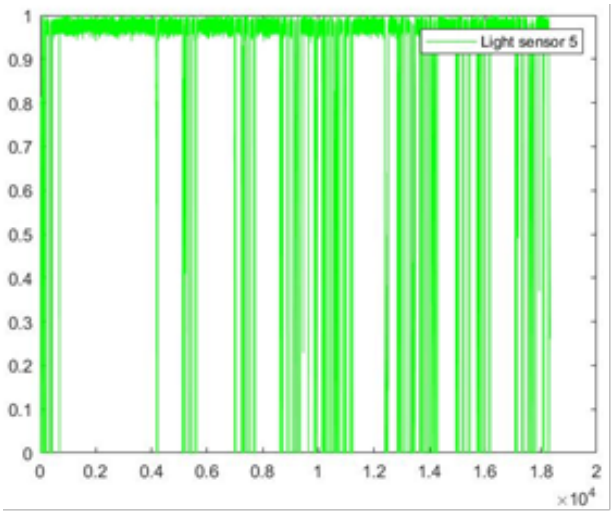


Figure 16: the output value of ls5

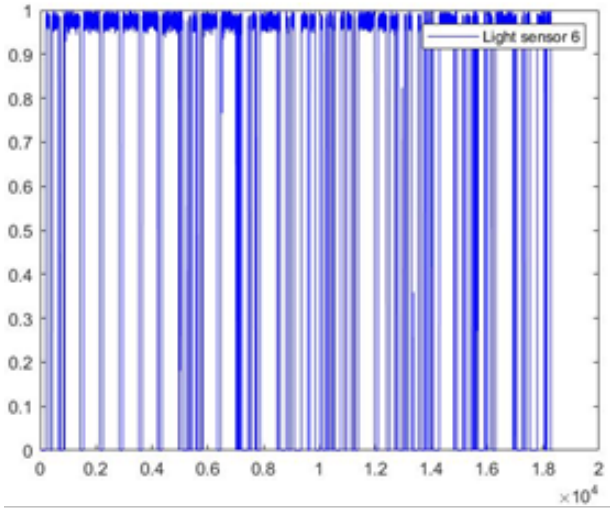


Figure 17 : the output value of ls6

Figure 18 shows that although there is noise and error in the output value, the output value greater than 0.65 when the robot in white area, so threshold of 0.55 in state 5 is correct. The front proximity sensor 0 has a sensible range of output value, which the same as task 1.

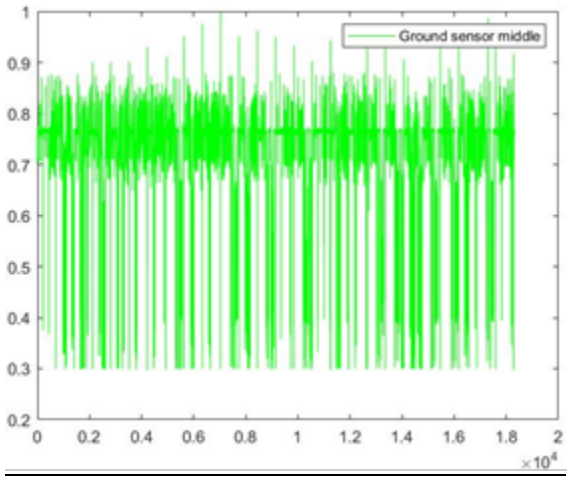


Figure 18 : the output value of middle

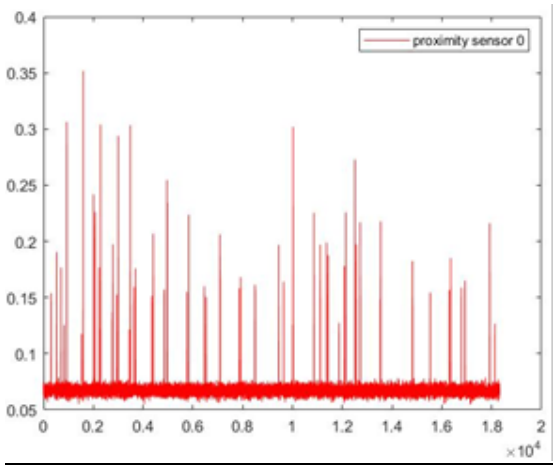


Figure 19: the output value of ps0

Task 3 : ER – obstacle avoidance and line following

In this task, we have to decide the hyperparameters of neural networks and genetic algorithms first, and the fitness functions of evaluating the performance of a neural network.

For the architecture of a neural network, from task 1 we know that only front and side proximity sensors and ground sensors are enough, therefore the input layer has 9 neurons. Because we have two actuators, the output layer has 2 neurons. After serval testing, we finally decide to use two hidden layers, the first hidden layer has 12 neurons, and the second hidden layer has 10 neurons.

In the genetic algorithm, the higher number of populations increases the probability of finding an optimal result, but it also increases the duration in each generation significantly. The number of generations decides how long the search going, but we need a correct fitness function first, otherwise, the search will go in the wrong direction and the result is not the ones expected. When generating a new population, in this case, the cross over point is always at the middle, the first half part of a new gene from the first parent, second half part of a new gene from the second parent. The number of elites means how many top rank individuals can go to the next generation directly without any crossover. Crossover rate means the probability of whether an individual that is not elite can go to the next generation without any crossover.

We found that most of individuals have low fitness, so we set the crossover rate below 30. When reproducing, the new gene will have mutations. The mutation rate represents the probability of adding a random number for each point in the new gene. This gives the individual the capacity of jumping out of local minima.

The fitness function consists of four part : forwardFitness, followLineFitness, avoidCollisionFitness, and spinningFitness . When designing the fitness function for neural network, we should follow some rules; first, the function must be continuous otherwise we will get a runtime exception; second, calculate the fitness should be very fast; third, the higher fitness value must be the actions expected.

forwardFitness is to evaluate the speed of the robot. We expect the robot to move faster with a straight line. Our function has two parts, first part being a base value. Assuming the velocity value is between 0 to 3, the maximum base value is 42.

The maximum ballpark radius of curvature is near 0.18m, and then using angular velocity formula to calculate the delta angular velocity between to left actuator and right actuator, given the axle length is 52mm and wheel radius is 20.5mm.

Assuming the robot will follow the line stable and correctly, the angular velocity is the same of two side of the robot, not the actuator's angular velocity. So, the delta velocity becomes $(2*Pi*r_1/T) – (2*Pi*r_2/T)$, and the result is 0.326726/T where T is how long we want the robot run a 0.18m radius circular.

Hence, we divide 0.326726/T by the wheel radius, and we will get the difference of angular velocity between two actuators, that is 16.015963/T. In our case, the robot usually finish a 0.18m radius circular around 13 second, so, the abs(self.velocity_right - self.velocity_left) is around 1.3.

We can use the same approach to calculate this delta angular velocity of two actuators when the robot avoids an obstacle. Although the obstacles are quite small in our case, this delta value is below 2. This is the reason why threshold is equal to 2.

If the delta angular velocity greater than 2, we can believe the robot has a sharp turn. And the penalty is dividing the base value by the delta angular velocity. Figure 20 shows the graph of this function, note that we will not use the discontinuous part.

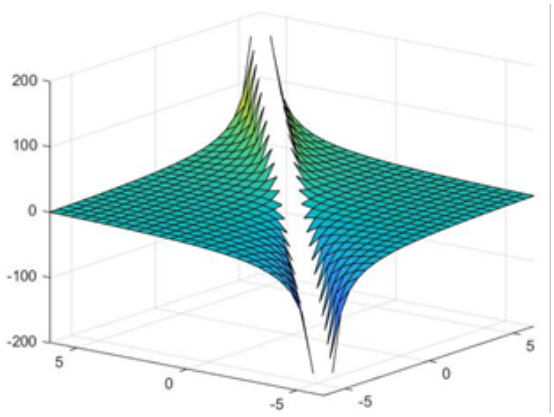


Figure 20: forwardFitness

followLineFitness evaluate whether the robot is following the line or not. According to previous tasks, the output value of ground sensors is around 300 if the floor is black and around 750 if the floor is white. So, the robot will get a negative fitness which is a very strong restrain. However, we did not restrain which side of the line the robot can follow, that is all three ground sensors detect black floor or only one sensor detect black floor. It is to keep the robot neither too close to the obstacle nor too far. However, the robot is avoiding an obstacle constantly, therefore we decided to make this fitness as a factor of final fitness function.

The normal case is "no obstacle", and the fitness is 1.

Due to the fact that all the obstacles are small in the arena, we decided to decompose "avoiding" action into two parts, one is after first detect an obstacle, the robot has to turn left or right immediately; another part is following the

obstacle until it reaches the line again. After few experiments, we found that the obstacles are shorter than the robot and there is an angle of proximity sensors, shown in figure 21. In this case, although the robot almost collided to the obstacle, the output value of proximity sensor still around 65. Hence, we cannot directly reuse the boundaries from task 1. After few tests, we found that the output value is around 250 of the first time detected.

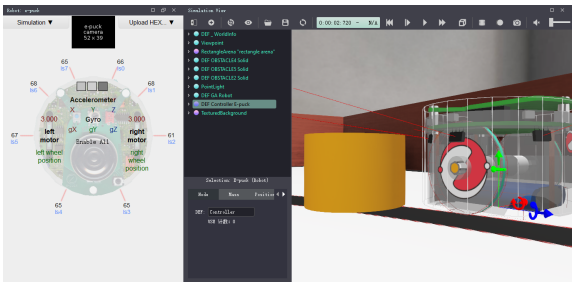


Figure 21: the robot already very close to the obstacle, but the proximity sensor still cannot detect it

When following the obstacle, we hope the robot will not have collisions or too far from the obstacle.
spinningFitness is to filter those populations that output one negative and one positive value. We simply reward those moving forward population and not reward those spinning population.

Results:

After 200 generations, with 40 populations, 5 top elites, 20 crossover rate, and 40 mutation rates, we got a result has a high fitness. Table 1 shows the each running duration.

	Time in minutes			
	First run	Second run	Third run	Average time
Task	1:37	1:43	1:36	1:39

Table 1: statistics

Discussion

BBR and ER comparison

We compared the BBR and ER approach performances in this arena. Overall, the ER approach is much faster than BBR approach, 2954 and 3649 iterations respectively.
When using ER approach, we found that the robot only followed the outer boundary of the line rather than the middle of the line. From figure 22, 23, and 24 we can see that in ER approach, green curve, mainly left ground sensor worked which kept output lower value in figure 22, and seldom middle ground sensor has only few lower value in figure 23, whilst the right ground sensor kept outputting high value with no lower value in figure 24. With BBR approach, red curve, all the three ground sensors have to work in order to keep the
robot in the middle of the line. According to performances, two robots indeed followed the line accurately, and the fitness function does not require the robot to be in the middle position. And two ground sensors are enough for the robot to follow the outer boundary, for example, if both two ground sensors detect black, the robot should go right; if both sensors detect white, it goes left; if left sensor detect black, and right sensor detect white, it goes straight forward (Figure 22, 23, 24).

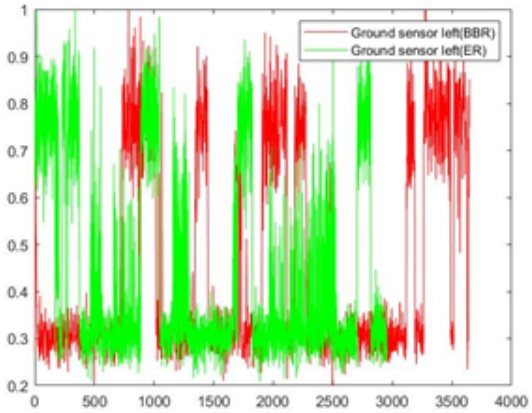


Figure 22: left ground sensor

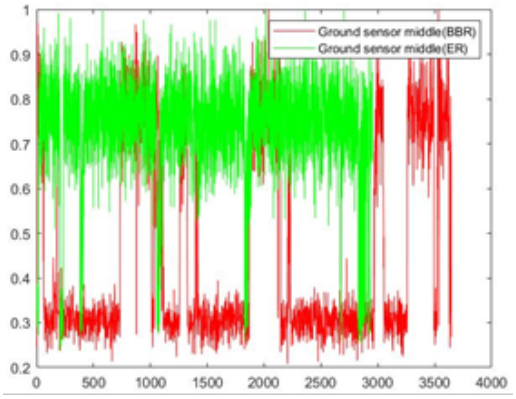


Figure 23: middle ground sensor

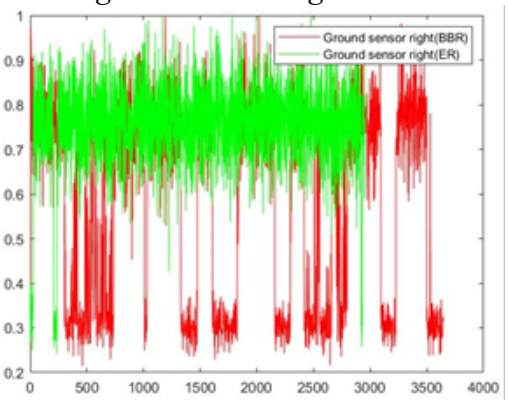


Figure 24: right ground sensor

Due to ER approach following outer boundary, the robot did not use left side sensors at all. But the robot collided the first yellow obstacle repeated, since the proximity sensors has often high output value.

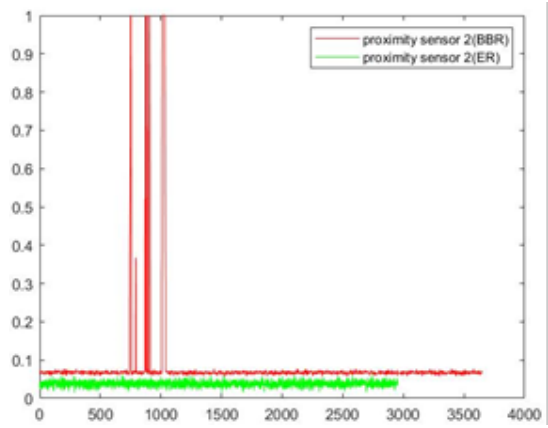


Figure25:ps1

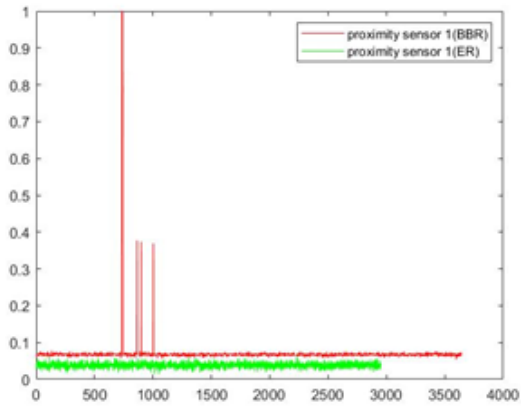


Figure 26: ps2

For the right-side sensors (Figure 27, 28), the robot collided to the first yellow obstacle of ER approach, as with the BBR approach. It is arduous for the robot to avoid first obstacle without any collisions. Moreover, both approaches have the same well performance for second and third obstacle. The difference is that BBR approach has many peak values when avoiding an obstacle, but ER approach has only one peak value. One reason is that the robot used a shorter path to avoid the obstacle from the outer boundary of the line, when the robot only needed a slight turn for avoiding an obstacle.

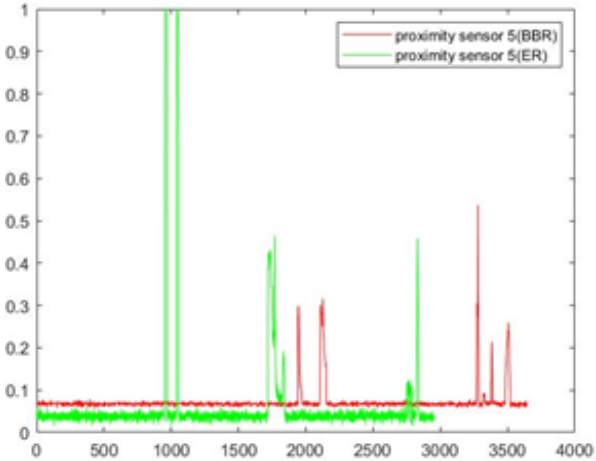


Figure 27: ps5

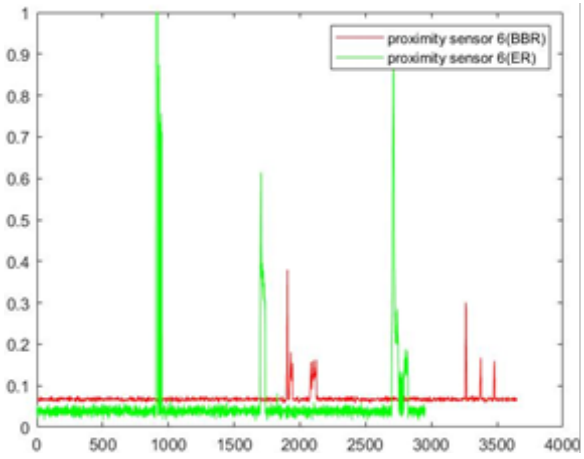


Figure 28: ps6

Figure 29 shows the differences of velocity between both approaches. We can see the average of "no obstacle" velocity of ER approach is higher than BBR approach, and this is the main reason why ER is faster than BBR approach. Second reason is ER approach spends less time avoiding the obstacles. None the least, the range of velocity of is ER smaller than BBR approach, so the robot moves smoothly.

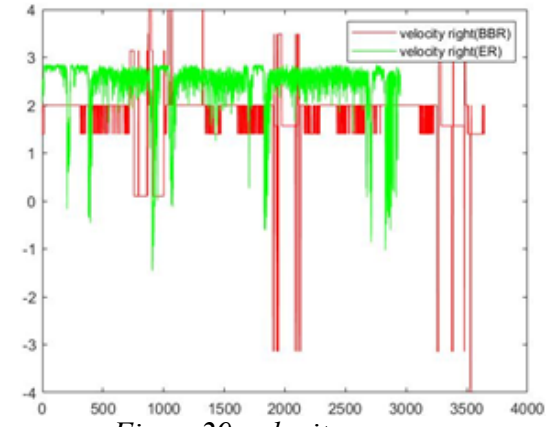


Figure29:velocity

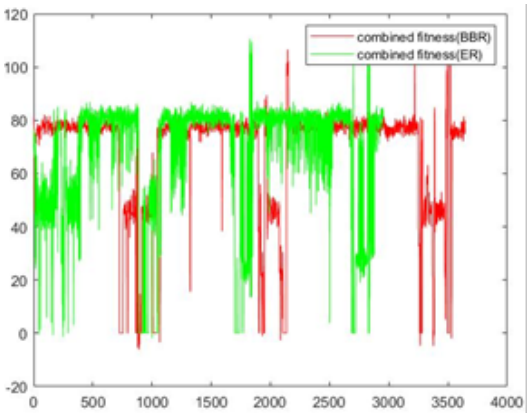


Figure 30: fitness

From figure 11 we can see that ER approach has higher combined fitness than BBR approach due to higher velocity.

There are also other methodologies used to create intelligent robot controllers. [5] Eg. Hierarchical Task Network, Dynamic Programming, Divide and Conquer, Rapidly exploring Random Tree, Simulated Annealing. These are more complicated to implement but efficient and smooth controller that we can apply to larger domain comparing to BBR and ER approaches.

Conclusion

To conclude, we can say that we successfully executed the tasks and analyzed the results in a thorough way. From our research and own experiment, based mainly on navigation and obstacle avoidance, we can establish that BBR and ER approach are both a good way of controlling robot but where we may find a better solution using ER approach, the cost is it is very time-consuming to train a model and hard to fine-tuning the hyperparameters, and we have to define a good fitness function first.
This helped us have a better understanding of how there are many approaches for on command like navigation on robots and which characteristic are more important or better to consider from a practical and theoretical view.

Reference

[1] Donald J. Norris, 2017, '**Beginning artifical intelligence with the raspberry Pi'** pp 313-345, Barrington USA, Springer Link.

[2] Nicolas Bredeche, Jean-Baptiste Mouret, Agoston Eiben, 2015, '**Evolutionary robotics: what, why, and where to'**, UMR 7222, ISIR, Sorbonne Universités, ; UPMC Univ Paris 06, Paris, France ; UMR 7222, CNRS, ISIR, Paris, France ; Department of Computer Science, VU University, Amsterdam, Netherlands, Front. Robot. AI.

[3] Brooks, R. (1986) '**A robust layered control system for a mobile robot'**, IEEE journal of robotics and automation, 2(1), pp. 17–18.

[4] Dr. Patricia Vargas, **'Intelligent Robotics Lectures'**, 2021, Heriot-Watt University, Edinburgh.

[5] Hossein Ahmadzadeh, Ellips Masehian **Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization** , 2015, Tarbiat Modares University, Tehran, ScienceDirect.