Patricia Vargas

F20RO

intelligent robotics coursework 2

Stuart Marples & Robbie Dunn

**INTRODUCTION**

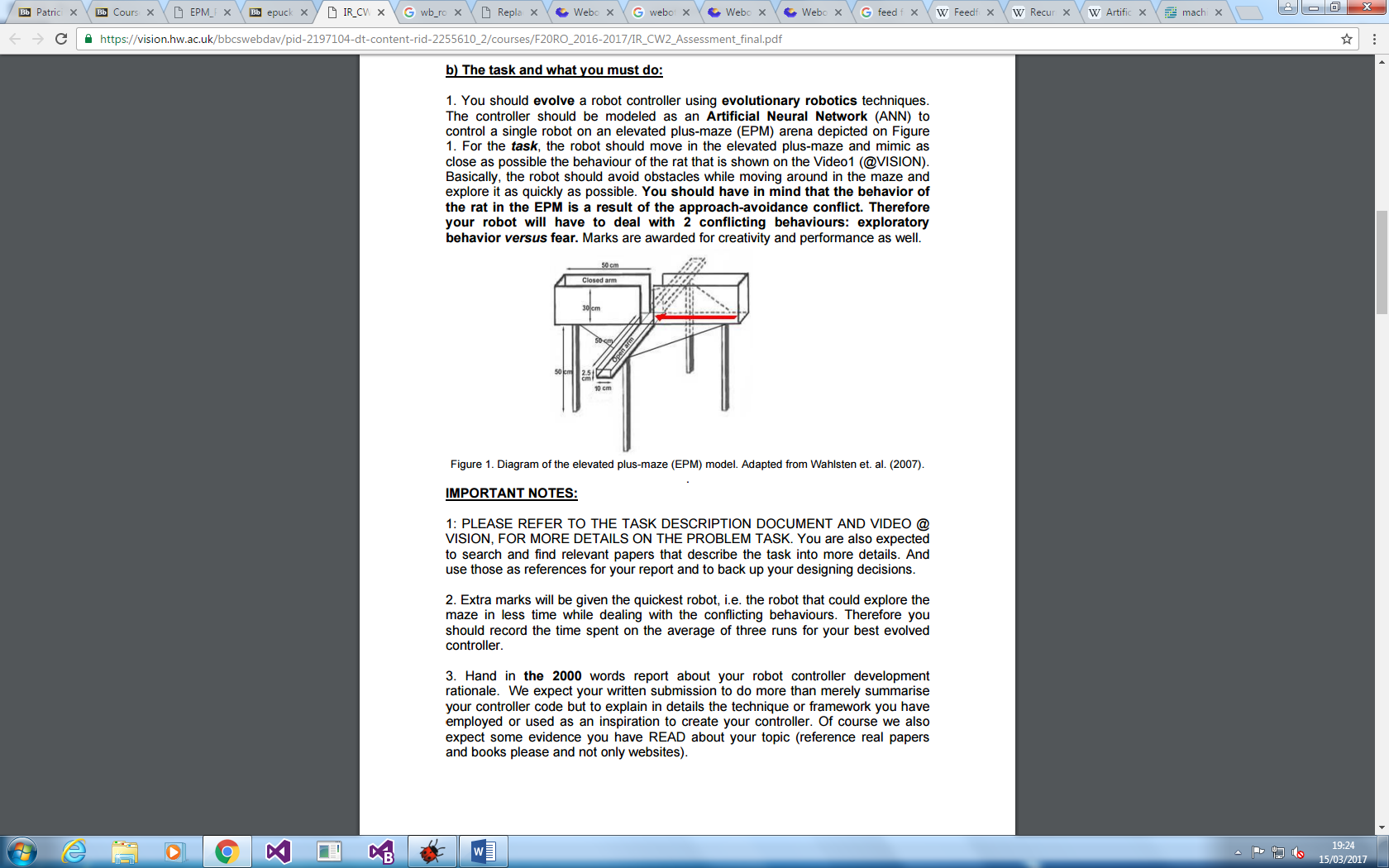
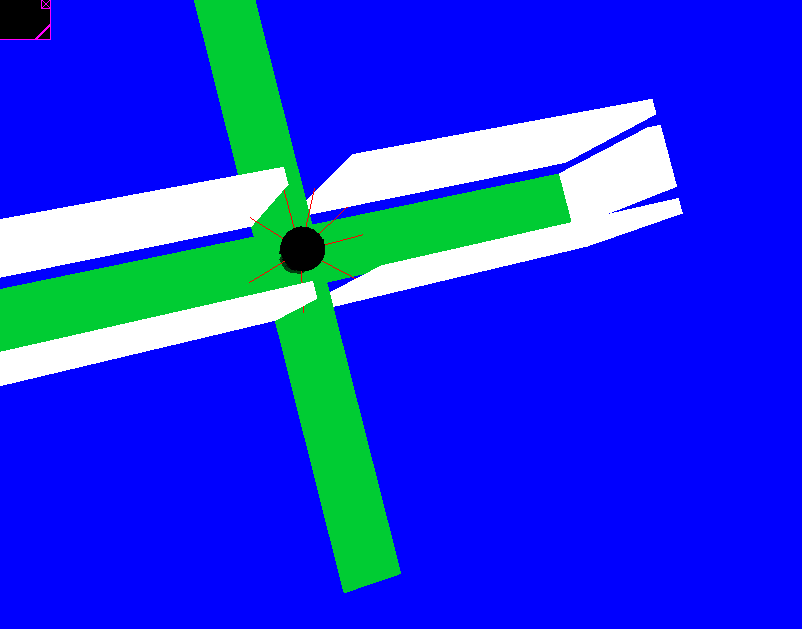
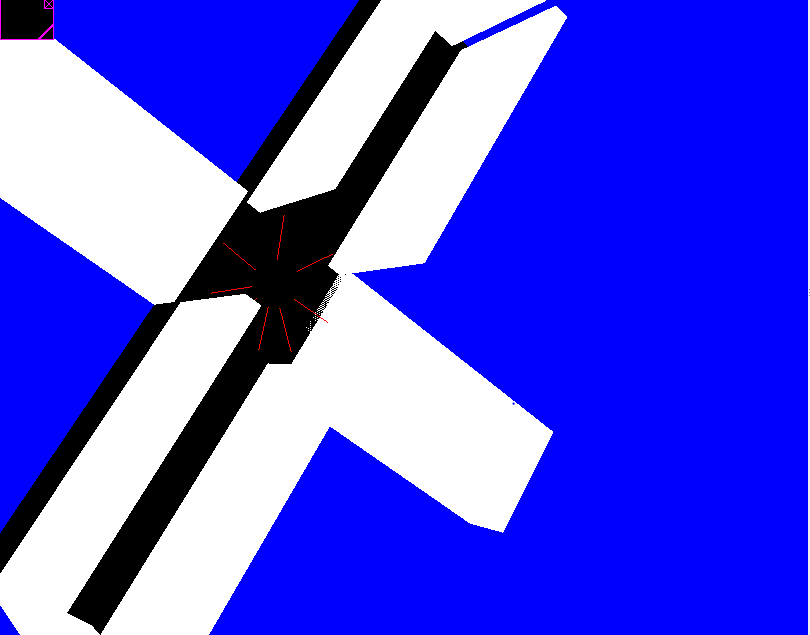
For this coursework, we attempted to recreate the Elevated Plus-Maze task using an E-puck in the Webots simulator. For creation of the Plus Maze, we used the sizes recorded in Figure 1 to create the most accurate and lifelike simulation. The E-puck was trained using a multi layered feed forward neural network which takes in the readings from the 8 distance sensors as input. Once this data has been fed through the network you are supplied with two outputs which are the speeds for each of the wheels. We then trained the E-puck using a Genetic Algorithm (GA) to try and complete the task in an optimal time. The goal of this coursework was to emulate the approach-avoidance conflict behaviour of a mouse.

Figure 1Diagram of Elevated Plus Maze - Adapted from Wahlsten et. al. (2007)

**DESIGN**

**Environment**

Initially, we designed the environment using the lengths as shown in the diagram in Figure 1. We did not include the legs shown as we decided this would be a waste of time and resources as they are unused. This allowed for a more accurate creation of the maze in the virtual environment leading to better comparable results with the real maze. After attempting to evolve the agent using this environment, we realised that having the floor protected by walls and the open floor overlapped was causing problems for the e-puck such as the wheels not functioning and the robot flying around the world.

After encountering this problem, we redesigned the environment, to eliminate the overlapping objects which were causing problems for the e-puck. We also decided to colour the open floor (unprotected) differently to the protected maze floor.

**Neural Network**

Our first design choice was to use a feed forward neural network or use a recurrent neural network. Dontas, g (2010) discussed when and why you should use each of these networks for the best results. A feed forward network is better for taking in input and getting one or more response outputs. The recurrent neural network was created to learn distributed representations of structure, such as logical terms and is harder to implement than feed forward. Both of these networks could have been used but we decided to use the feed forward network as it was the simplest Artificial Neural Network (ANN) to implement and would also be able one of the most effective types of networks for training the E-puck for this task.

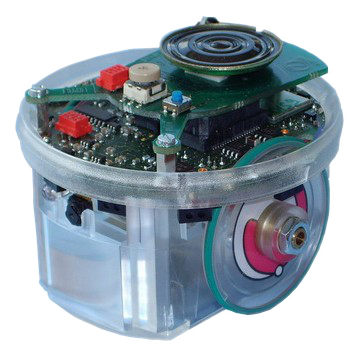
**Genetic Algorithm**

We then had to design the fitness function so that it could accurately represent the output that we want from the E-puck. We started by utilising the distance sensors and creating an area where the E-puck would prefer to be. This was done by detecting collisions and detecting when the E-puck was too far away from the wall and decreasing the fitness based on these conditions. This is because we wanted the E-puck to stay inside of the plus maze and would also not want to hide at a wall to attempt to mimic a mouse’s approach behaviour which favours enclosed spaces. We would also measure the distance that the E-puck was travelling during the simulation and increase the fitness based off how far the E-puck had travelled to encourage the E-puck to explore much like the mice would. To detect when the E-puck was going across the non-walled platform, we utilised the supervisor node which could detect the location of the E-puck and decrease the fitness based on how long the E-puck spends in that area. This is because a mouse would begin to feel anxious at a high height and in open areas so would turn back so the E-puck has a higher fitness the longer it ventures out there to mimic the level of anxiety. This was a main area of issue as we were unsure on how to create accurate and correct behaviour without limiting the simulation too much.

Mataric. M (1997) conducted an experiment that utilised reinforcement learning to try teach multiple agents with behaviours and conditions in various environments. The various behaviour rules utilised helped to define the behaviour for our E-puck as they used a behaviour called “homing” which is used to take the robot to a particular location. This behaviour led to the idea for the fitness to be increased for the distance travelled by the E-puck instead of just for reaching a certain position. They also utilised the “dispersion” behaviour which maintains a minimum distance between the robots. The E-puck was then programmed with a similar behaviour where it would try to be close to a wall without colliding but would also not stray too far away from the wall.

**IMPLEMENTATION**

**E-puck Controller**

The first class implemented, 'epuck\_ne.c' was used to control the e-puck within the Webots environment. When the class is run, the robot is initialised and the proximity (distance) sensors are enabled. The sensors are enabled using the wb\_distance\_sensor\_enable() method and are used as input for the neural network. There are 8 proximity sensors on the e-puck, 'ps0' to 'ps7', arranged in a circle around the robot.

Next, the neural network used for calculating the robot's movement is initialised (see Neural Network section). The robot then uses a 'receiver' device to receive a genotype of weights from the supervisor class and updates the neural network with the provided weights. Once the genotype is received, the controller runs and feeds the proximity sensor input to the neural network to determine the speed of the robot's wheels (output of the neural network). If the class receives another genotype from the supervisor class, the current genotype ends and the weights of the neural network are updated yet again.

**Neural** **Network**

The neural network was a class used in the project as a utility to provide methods to initialise the network, feed through the network and to update the connection weights. The init\_network() method initialises the neural network with 8 input layer nodes (representing each of the proximity sensors on the e-puck), 20 hidden nodes, and 2 output layer nodes (for each wheel of the robot). The memory for weights of the network is also allocated, with a vector of doubles to represent the connections of each layer of the network. Our network consisted of two layers, the input to hidden layer and the hidden to output layer.

The feed() method in the class allows inputs to be given as arguments and returns a vector of size 2 to represent the output to the robot's wheels based on the current input. Node activations are calculated using the weighted sums of inputs to said node, and the hyperbolic tangent function was used to normalise activations to values between -1 and +1. The final method, load\_weights(), allows a vector of doubles to be provided as an argument and updates the network's weights to these provided values.

**E-puck Supervisor**

The final class, 'epuck\_ne\_supervisor.c' was initially implemented as solely a genetic algorithm, however it was soon adapted to also act as a supervisor node for the e-puck robot. The supervisor provides methods to send genotypes, reset the robot's position, and calculate fitness of the e-puck, which are all essential for the agent's evolution. The class begins by reading and storing the robot's initial starting translation and rotation, in order to reset its position at the beginning of each genotype. The class stores these variables and then resets the robot appropriately using the methods wb\_supervisor\_field\_set\_sf\_vec3f() and wb\_supervisor\_field\_set\_sf\_rotation().

Before the supervisor can begin evolution, it generates the first population of genotypes. The POPULATION\_SIZE variable is used to create a genotype of weights for each individual of the population. The C rand() function is used along with RAND\_MAX to generate weights randomly between the range -1 and 1. Once the initial population has been generated, the supervisor begins sending genotypes to the controller class using an 'emitter' device. The device is set to the same channel as the receiver for the e-puck in order to create communication between the classes.

Once every genotype of a population has been run, the supervisor class evolves the current population. During the running of each genotype, the supervisor assigns the genotype a fitness based on its performance within the maze. The fitness for each genotype is calculated based on the distance it explores in the maze, without falling off the edges. Due to these requirements, the agent evolves to bravely explore within the closed sections of the maze though is more hesitant to visit the open arms without walls. This behavior is similar to that of the approach-avoidance behavior of a mouse in a real EPM. Using the fitnesses calculated, the class assigns a probability of each genotype for being a parent to use fitness proportionate selection. Each genotype is chosen based on a proportion of its fitness over the total fitness and weights between the chosen parents are picked randomly. Each individual gene of the genotype is then mutated with probability MUTATION\_RATE. The mutation involves replacing said gene with a new random weight within the determined range (-1, 1). The new generation of weights is then created, and the supervisor sends them to the controller for evaluation.

**RESULTS / EVALUATION**

**Results**

Whilst evaluating the agent, we utilised finely-tuned parameters for the neural network and genetic algorithm according to previous and runs and external sources. The neural network used 20 hidden nodes which we found was appropriate for the complexity of the problem. The genetic algorithm was run for up to 80 generations, or until the maze was explored to a determined extent. A population size of 15 was used with a mutation rate of 0.5% and a crossover rate of 30%.

When evolving the agent, in initial generations it is unable to explore the EPM effectively and tends to spin on the spot or navigate portions the maze backwards. At this stage of evolution, the robot also falls off the maze occasionally, causing a loss in fitness. After some generations of evolutions (usually around 30), the robot is able to explore the maze to an extent, without falling or spinning in circles.

After around 50 or 60 generations of evolution, the agent reaches its equilibrium and does not see much improvement from more epochs of evolution.

Once the agent was evolved, we gave it five minutes to completely explore the EPM and recorded the results, as shown in the table below. We determined the EPM to be explored when the e-puck had visited all closed areas of the maze.

The agent was able to complete the task in all runs except for the second. During the second run, the e-puck failed to explore a section of the closed maze, and hence was given a five minute penalty to its run time. The average time for our agent to fully explore the maze was six minutes and five seconds.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Time |  |  |  |
|  | First run | Second run | Third run | Total time (average) |
| Task | 4:13 | 10:00 | 3:48 | 6:05 |

**Conclusion / Evaluation**

Initially we coded the project without the use of supervisor node. When it became apparent that we will need a supervisor node to reset the e-pucks position and run genotypes, we edited the genetic algorithm file to also include these methods. Without the use of a supervisor we were unable to evolve our agent, though could see that the implementation of the neural network and genetic algorithm were working as we expected.

We also received numerous errors while using the Webots simulator as the E-puck would frequently flip out of control or teleport to unknown positions off screen. We did not know how to fix these errors as they seemed to be cause by the simulation itself and not by the controller. After the demo of the coursework, we realised that the problems were occurring due to 2 solid objects in our environment overlapping. This caused internal errors in the controller: redesigning the environment improved results.

In order to improve the environment, we removed the overlapping section of the solids, and recoloured the open (unprotected by walls) areas, to easier determine which zones are safe and which are not. These improvements prevented the e-puck from breaking, though we saw unusual readings from the proximity sensors. The proximity sensors were reading negative values, which should not occur, and were frequently providing unreliable results.

Recolouring the floors of the environment was initially for the use of ground sensors on the e-puck. However, we soon realised that the supervisor could read the location of the e-puck to determine where it was at all times, so ground sensors were unnecessary. If we were to rerun the experiment, it may be appropriate to use ground sensors as an input to the network also, which may improve results and show the avoidance behaviour of the robot more prominently.

After the coursework demo, Patricia advised us to approach the fitness function in a more suitable way. Initially, the robot was attempting to explore the maze, though the avoidance behaviour of the robot being on the open arms of the EPM was not being exhibited. We decided to update the function to better reflect the mouse’s behaviour, decreasing fitness for every time step the agent spends on an open (dangerous) area of the maze. We also needed to update the crossover method, in order to use random genes from each parent, instead of the original method which merged the first layer of weights of parent 1 with the second of parent two.

**References**

Dontas, G (2010). Feed Forward and Recurrent Neural Networks [online] Available at: http://stats.stackexchange.com/questions/2213/feed-forward-and-recurrent-neural-networks [Accessed 24/03/2017]

Mataric, M (1997). Reinforcement Learning in the Multi-Robot Domain. Brandeis University