Learning Machines – Group α

Nikolay A. Polyanov, Mick IJzer, Leon P. Berberich & Justus F. Hübotter
Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands

Abstract

In this report the concept of learning machines is used to accomplish three tasks in both simulation and in a real environment; obstacle avoidance, foraging, and hunter-prey behavior. For each, the task itself and the problems that arise are described. Furthermore, the gap between performance in simulation and the real environment is discussed. The robot that is used is a Robobo equipped with a smartphone camera. For the controller a deterministic policy is chosen and the learning component comes from an evolutionary algorithm. In order to deal with the increasing difficulty of the tasks, the fitness-function as well the state space are expanded accordingly. The report concludes with the chapter where the results and implications are discussed.

1 Introduction

It all started in June 1956 when a group of scientists sat down together in Dartmouth College. The topic of their discussion was something that didn't have a name yet. Grace Solomonoff, who was one of the members of this meeting, states: "People didn't agree on what it was, how to do it or even what to call it.", Solomonoff (1956). Later it turned out those people were discussing a machine that could think, Artificial Intelligence. Artificial Intelligence is the study of developing artificial entities, such as machines, that can act in an intelligent manner.

Within Artificial Intelligence, there are thinking machines and acting machines. Also known as digital and embodied intelligence. Learning machines are a combination of these types of intelligence. A learning machine gets input from the world around it, which is then processed by some controller. The controller decides how the physical robot should act. This changes the position of the machine in the environment and thus the input it gets from the world around it. The idea of a learning machine is that, given a task, it can use the information from the environment and the feedback it gets from certain actions to learn how it should behave. Continuous improvement of behavioral performance is the result.

The main goal of the paper is to complete several tasks by using learning machines. The hardware robot that will be used is the Robobo. Robobo is a relatively cheap educational robot. It has a wheeled based body. On top of the body Robobo has a placeholder for a smartphone. Architecture like this allows combining the base, which is able to move, grasp objects and get information of the surroundings, with the technology of a smartphone. The tasks that are carried out and described in this report include: object avoidance, foraging objects, and evading a predator/catching a prey.

During writing this report, our group has experimented with several approaches available in machine learning. Some of approaches tried, which are not included in the actual report are: Neural Network and Deep Q-learning. In order to successfully perform the desired tasks one approach was selected. And this approach is the combination of an Evolutionary Algorithm and policy iteration. During the experiments we show that the combination of these two methods provides us with desired results.

2 Related work

In this section relevant scientific documents and papers are discussed. These were used as an inspiration for this report.

First of all, it is necessary to mention Bellas and colleagues who initially published Robobo in 2017, Bellas et al. (2017). The Robobo is now used in high schools to teach AI, Bellas et al. (2019). Information provided in their work provided us an with information of the Robobo functionalities that are available. For example, the manner of working with sensor information, and the possibility of getting the image from the camera of the mobile phone.

Secondly, the work of Ilmas and his colleagues was crucial for our report. Because, in their work they talk about teaching units developed in last years AI classes. More importantly, they recommend to use evolutionary computing or reinforcement learning, In their opinion those are best ways to teach Robobo proper behavior. Llamas et al. (2019).

The chosen approach to implement the controller of Robobo is a form of policy iteration, which was inspired by the book "Reinforcement learning: An introduction" Sutton and Barto (2018). Furthermore, an evolutionary algorithm was used to enable the learning process in the population through inheritance and mutations, Such an algorithm can be used to optimize the behavior of the agent in the long run. For constructing the evolutionary algorithm we used the book "Introduction to evolutionary computing" as a guide. Eiben et al. (2003).

Another important work that needs to be mentioned is a masterthesis by JiunHan Chen, who described the occurring of reality gap related problems due to the construction and connections of the Robobo-Robot. The main issue Chen faced in his experiment was the shift of the transmission from 10 frames in the simulation to only 5-7 frames per second in the real world, caused by the connection over a local WiFi, ROS as well as router. This lower frames per second in the real environment, had an impact on object detection and thus delayed the execution of the algorithm which caused over rotation Chen (2019). This work provided useful information to trim the parameter in the simulation-environment accordingly in order to minimize the reality-gap.

3 Methodology

3.1 The proposed approach

The goal of this project was to develop agents that are able to perform certain objectives, such as obstacle avoidance, foraging, and hunting. To accomplish this goal a learning component is required. Therefore, as mentioned earlier, several approaches are suitable, such as reinforcement learning, a neural network, or an evolutionary approach. After some initial experimentation with all three options, the evolutionary approach was chosen. Central to this approach is the evolution of a population over multiple generations. Where a population consists of individuals or agents whose performance is to be evaluated. The population as a whole should be able to approach a global optimum in performance. This happens because of the exploration (cross-over and mutation) and exploitation (parent and survivor selection). In the pseudocode below the general steps of an evolutionary algorithm are shown.

Finding the optimal controller for each task will be carried out in simulation. After which the controller will be tested in a real environment. Some parameter tuning might be necessary to close the reality-gap. For the simulation the V-REP application by Coppelia Robotics is used. As for the real environment the Robobo and a regular smartphone are used to evaluate the performance. The Robobo was developed in The Robobo Project by MINT and has multiple infrared sensors in the front and back to get input of its environment. A comprehensive representation of the Robobo is shown in Figure S1. The software that is used to control the Robobo is a combination of the development app (available in the Google Play Store under the name "Robobo Developer"), the Robobo Operating System created by The Robobo Project and a custom python based wrapper function.

3.2 The controller representation

Each individual in the population has to have a way to perform the task. This is done by using a controller. The representation that is used in the experiments is based on a deterministic policy in the

Algorithm 1: General steps in an Evolutionary Algorithm

```
Result: A set of possible solutions
Initialize population(size);
for each generation do
    parents = parent_selection(population);
    for each pair(parents) do
        children = recombine(parent 1, parent 2);
        mutate(children);
        evaluate(children);
        population += children;
    end
    population = survival_selection(population);
end
```

form of a look-up table and is similar for all three tasks. The columns of the table correspond to the possible actions and the rows correspond to possible states. For all three tasks the policy consists of three actions, namely left turn, right turn, and forward, and a number of states. The possible states are equal for the second (foraging) and the third (hunter-prey) tasks. However, for the first (obstacle avoidance) a reduced state space is used. The controller works in a way that when an agent is in a certain state s the corresponding action will always be action a. This makes it a deterministic policy.

3.3 The learning algorithm

In an evolutionary algorithm the individuals within the population themselves don't learn, it is the population as a whole that moves towards an optimum over the generations. There are two important components to make this happen, namely variation and selection. Variation is essential to explore the solution space and find possible optima. The variation operators are recombination and mutation. Every generation several new individuals will be created by recombining the genotype of older individuals, thereby creating an individual with properties of both older individuals. This means that the deterministic policy of the new individual is a combination of the deterministic policies of its parents. However, when the population doesn't contain a certain part of the policy the children will also never be able to inherit that policy. Mutation solves that problem by randomly changing some parts of the genotype. With a set probability some parts of the deterministic policy are randomly changed. For example, the new individual inherited, by recombination, that it will always go forward when in state s. This part of the policy is selected for mutation, so now the new individual will always go left when in the same state s. These variation operators make sure that any possible genotype could theoretically be created in a new individual.

The selection operators are important to exploit good performing candidate solutions. Individuals within the population are selected to be parents based on their relative performance within the group of candidate solutions. The individual with the highest performance also has the highest chance to be selected as a parent. Similarly, after the new individuals are added to the population a selection will be carried out. This is important to make sure that bad candidate solutions are removed from the population. In this selection the same rule applies as in the parent selection; the best performing individual has the highest chance to remain in the population. In Table 1 the used evolutionary algorithm is comprehensively summarized.

4 Experiments

The overall challenge of this course is to let the Robobo complete three individual tasks, each bringing shared as well as its own problems to overcome. These tasks are (1) obstacle avoidance, (2) collecting food, and (3) hunting a prey / escaping a hunter. In task 3 we reuse the controller of the best individual from task 2 as the hunter robot and evolved only the prey controller.

In general, the controller of the Robobo is not to be hard-coded, so that an omnipresent problem is the ability to learn by trial and error (problem 1). As we use an evolutionary computing approach to tackle this problem, it is not a single virtual agent which learns from its experience, but the learning

EA	Setting
Phenotype	Behavior when in a state
Genotype	Deterministic Policy
Mapping	Readings and policy table are translated into movement command
Recombination	Uniform Crossover (behavior in a state is copied from P1 or P2)
Fitness	Depending on task (see text)
Mutation	Shuffle each row with probability $p = 0.1$
Parent selection	Probabilistic based on rank $(n = 2)$
Survivor selection	Probabilistic based on rank $(\mu + \lambda)$
Initialization	Random generation of deterministic policy for 10 individuals
Termination	10 generations (11 in Task 3)

Table 1: Summary of the Evolutionary Algorithm.

happens on a population level. Therefore, we need to design an appropriate controller representation (problem 2) as well as fitness function to enable goal driven evolution (problem 3). As a controller we chose a deterministic policy in form of a look-up table inspired by reinforcement learning algorithms. The fitness calculation is based on rewards collected by the agent and summarized in Table 2. In all tasks, the robots needs to perceive the environment (problem 4) and adjust their behavior accordingly (problem 5). Following Occam's Razor, we keep the solution space as simple as possible. Therefore, we limit the state representation S (explained per task below) as well as the action space A of all involved robots to $n_A = 3$ actions (turn left $[-v, v]$, turn right $[v, -v]$, and move forwards $[v, v]$ with v being a constant speed parameter given to the two weel DC motors [left, right]). Initially the policies are tested in simulated robots for 100 time steps of $\Delta t = 300ms$ in a specifically designed environment shown in Figure 1. The final evaluation takes place in the real world, so transferring the controller and adjusting several parameters is necessary (problem 6). Key points that lead to differences between simulation and the real world are the increased noise of sensor readings highly dependent on lighting conditions and delay introduced to perception and control from the Bluetooth and WiFi connections involved. In order to transfer the learned result from simulation to the real world, we adjusted the parameters detection threshold θ, wheel speed v, and action duration Δt.

	Event	Reward r	Simulation stop
Task 1	moving forwards	+1	
	collision	–50	yes
Task 2	moving forwards	+1	
	collision with green block	+50	
	other collision	–50	yes
Task 3	survived time step	+1	
	hunter prey collision	-0	yes
	other collision	–50	yes

Table 2: Summary of the given rewards r for each task during simulation, which are summed over 100 time steps to the respective individuals fitness value.

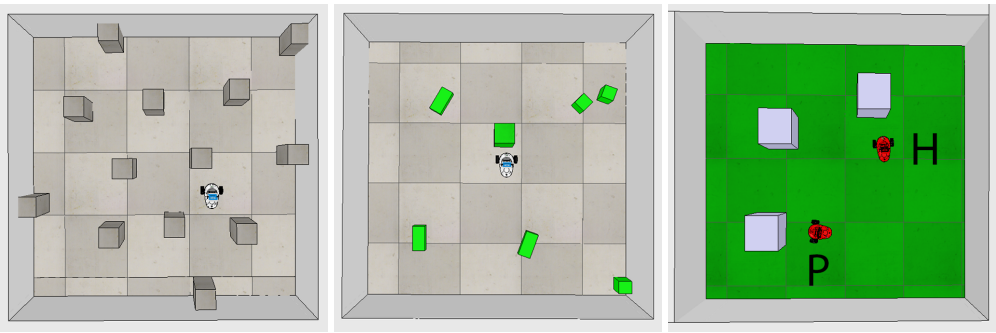

Figure 1: Scenes used for simulations during evolution for task 1 (left), task 2 (center), and task 3 (right). In task 2 the target objects are colored green, initialized in random positions each simulation and removed upon contact with the robot. In task 3 both hunter (H) and prey (P) are colored red.

4.1 Task 1: Obstacle Avoidance

The first task presented is obstacle avoidance in a closed arena with scattered blocks. In this case, the Robobo shall *move* through the environment, *detect* objects blocking its path, and *react* by changing its path to prevent collision. As all tasks took place in walled off environments, the solution to this task is also relevant for the following tasks. The Robobo is to cover as much ground over time as possible, such that standing still, moving only back and fourth, rotating in place, or driving in a small circle are undesired. The task specific key problem to solve during this task is environment perception via the robots IR sensors only. As we have limited the action space to moving forward as well as point turns, we decided to use only the five front sensors of the Robobo. We further decreased the the state representation by using only the maximum value of the right sensor readings as well as the the maximum value of the front and left sensor readings (green and red box in Figure S1 respectively). Each of the resulting values is compared to the threshold θ to decide if an object was detected or not. This results in a representation of perceivable state space S in two binary values and hence $n_S = 2^2 = 4$ possible states. Together with the $n_A = 3$ actions, there are $n_A^{n_S} = 3^4 = 81$ possible solutions for the target policy π to be searched. The results are of the evolution are shown in Figure 2 on the left. The graph clearly shows population learning over time with the first close-to-ideal solutions appearing after just three generations. After some manual tuning, the desired behavior was also exhibited by the hardware Robobo.

4.2 Task 2: Foraging

In the second task the Robobo should *detect objects* (food) of the desired color (green) in the arena and change it's path in order to *collect them*, while also *avoiding collisions* with other obstacles. In this task the robot also has access to the camera of the attached and downwards tilted smartphone, which helps to solve the new problem of *color sensitive object recognition*. We use openCV 2 (Bradski, 2000) to process the cropped camera image and mask the target color. We extend the state space representation from task 1 by three further bits: target detected, target seen in last τ iterations, and target last seen left or right. This results in a representation of perceivable state space S in five binary values and hence $n_S = 2^5 = 32$ possible states. Together with the $n_A = 3$ actions, there are $n_A^{n_S} = 3^{32} \approx 1.85e15$ possible solutions for the controller to be searched. The results of the evolution are shown in Figure 2 in the center. The graph clearly shows population learning over time with the population learning curve beginning to flatten out after about seven generations. After some manual tuning of the involved parameters ($\tau, \Delta t, v$ & θ), the desired behavior was also exhibited by the hardware Robobo.

Reflecting upon the exhibited behavior of the robots as well as the used training environment, perhaps it would have been more ideal to use the same placement of food blocks for all tests conducted to keep results comparable. With regard to our used fitness function, moving forward is imperative for the successful solution of this task anyways, so perhaps instead of setting an incentive for this action, we could have penalized each elapsed time step with $r = -1$ in order to find the solution that solves this task fastest.

4.3 Task 3: Predator & Prey

In the final task both Robobo should *detect* the other Robobo in the arena and change their path in order to *catch* the prey (hunter), or avoid the hunter (prey), while also *avoiding collisions* with other obstacles. As mentioned above, we reused the controller of the best individual from task 2 as the hunter, only evolving the policy for the prey. It used the same sensors and state representation as the hunter, except that the camera was pointing backwards. The results of the evolution are shown in Figure 2 on the right. The graph clearly shows population learning over time with the maximum possible fitness reached after only six generations. After some manual tuning of the involved parameters ($\tau, \Delta t, v$ & θ), the desired behavior was also exhibited by the hardware Robobo.

Reflecting upon the exhibited behavior of the robots as well as the used training environment, it may have been more ideal to block the direct path between hunter and prey. In the used instance, a prey that successfully avoids the first obstacle by turning right moves itself close to the hunter compared to turning left. This is rather arbitrary but induced a certain bias in the fittest individuals that could have been avoided by a different environment design choice. Reflecting on the choice of controller representation as well as state and action space, as expected from this approach we did not

observe complex behavior. A potential expansion of the possible actions (e.g. camera turn, backwards movement) and perceptive capabilities (e.g. more complex visual processing by a convolutional neural network) of the robots could have lead to some more interesting behavior, such as the prey actively hiding from the hunter.

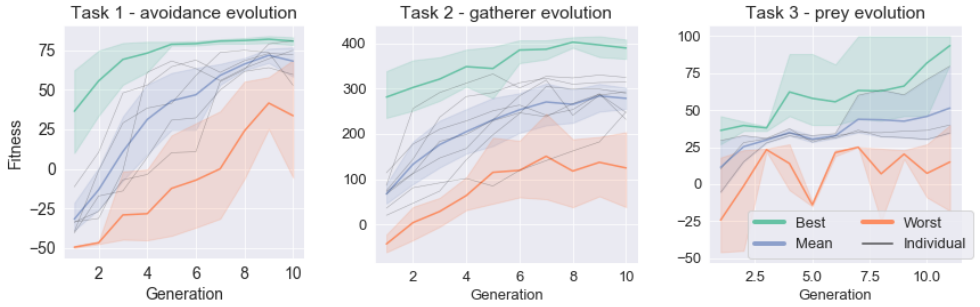

Figure 2: Population fitness development for 10 generations (11 for task 3) indicates successful population learning across all tasks. The experiments are repeated 5, 7, and 3 times respectively. The graphs show the mean fitness per run (black), as well as the overall average of each runs best (green), mean (blue), and worst (red) observed fitness (each $\pm 95\%$ confidence interval).

5 Conclusions

The goal of this report was to investigate the functionality of learning machines on three separate tasks. To accomplish these tasks a controller and learning algorithm were selected. For each task, obstacle avoidance, foraging, and hunter-prey interaction, the controller, the input from the environment, and the evaluation function were adapted to fit each task. The evolutionary approach that was chosen in combination with the deterministic policy as a controller resulted in a successful completion of the task, both in simulation and in a real environment.

It can be concluded that by using learning machines theory good task performance can be accomplished. Translating the input the agent gets from its environment, in this report from the infrared sensors as well as the smartphone camera, to something that can be used by a controller was an important part of the success. Furthermore, the controller representation was simple yet effective in its ability to get an accurate reading of the performance of the agent. Lastly, the finetuning of the robot movements was necessary to close the gap between simulation and reality. Our group managed to work around several difficulties faced when transferring from simulation to real hardware. Examples of those are: difference in the sensor calibration between simulation and real robot or difference between using non-identical mobile phones cameras for contour detection. Overall, all of the components of a learning machine were used to accomplish the three tasks.

References

Bellas, F., Mallo, A., Naya, M., Souto, D., Deibe, A., Prieto, A., and Duro, R. J. (2019). Steam approach to autonomous robotics curriculum for high school using the robobo robot. In *International Conference on Robotics and Education RiE 2017*, pages 77–89. Springer.

Bellas, F., Naya, M., Varela, G., Llamas, L., Bautista, M., Prieto, A., and Duro, R. J. (2017). Robobo: the next generation of educational robot. In *Iberian Robotics conference*, pages 359–369. Springer.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Chen, J. (2019). *The predator-prey evolutionary robots system: from simulation to real world*. PhD thesis, Vrije Universiteit Amsterdam.

Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing*, volume 53. Springer.

Llamas, L. F., Paz-Lopez, A., Prieto, A., Orjales, F., and Bellas, F. (2019). Artificial intelligence teaching through embedded systems: A smartphone-based robot approach. In *Iberian Robotics conference*, pages 515–527. Springer.

Solomonoff, G. (1956). Ray solomonoff and the dartmouth summer research project in artificial intelligence. In *Ray Solomonoff and the Dartmouth Summer Research Project in Artificial Intelligence*. Oxbridge Research.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Appendix

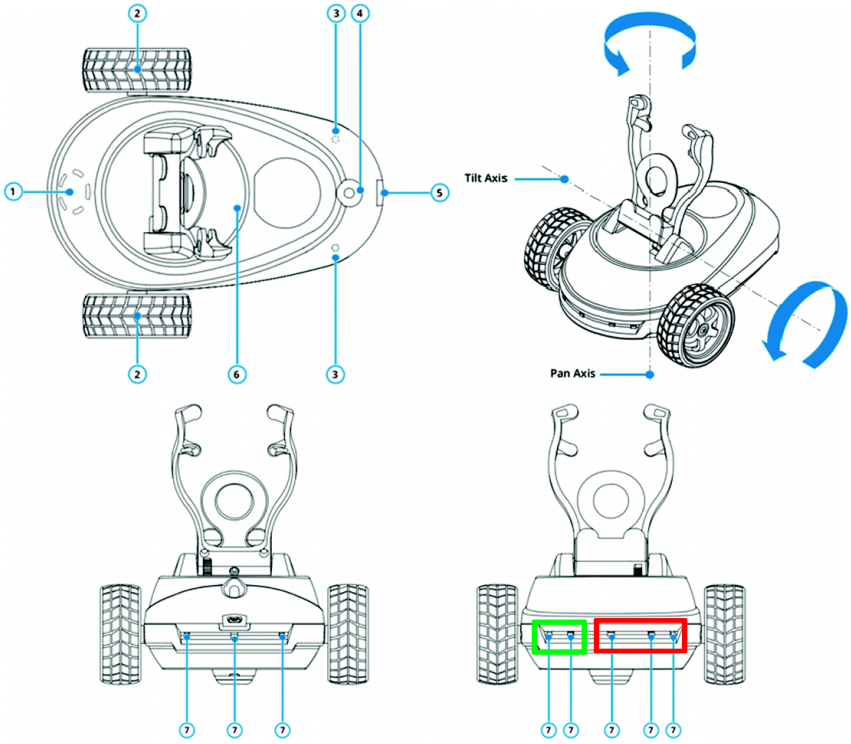

Figure S1: Robobo base main elements: (1) Front leds (2) DC wheel motors (3) Back leds (4) Power button (5) USB connector (6) Pan-Tilt DC motors (7) Infrared sensors. Figure and caption taken from Bellas et al. (2019) for educational purposes only without intend to publish.