Tuning the strategy for a game using Evolutionary Learning

Borbely Dominik, Fellner Christoph, Siegel Bernhard, Groiß Simon,
Sztavinovscki Jeremy, Steigenberger Julian
Department for Computer Science
Secondary Technical College
Wiener Neustadt, Austria
Email: anubis@robo4you.at

***Abstract*—In our experience robots in most robotics competitions do not differ much from each other, the biggest advantage over the opponents lie in tuning parameters like correction factors and developing a good strategy. However this can be a challenging and time-consuming task. Therefore, the team has experimented with evolutionary learning for decision making and adjusting parameters. Evolutionary learning is a very reliable and rarely used method based on Charles Darwin's theory of natural selection. The purpose of this paper is to determine the influence of this procedure on the strategy and behaviour of a bot and the applicability of this algorithm to robotics competitions by simulating the double elimination task of compAIR 2023 [3]. Through simulations, the efficiency of evolutionary learning and the use of this method in robotics competitions could be determined.**

I. Introduction

The last year Anubis experienced that in robotics competitions like Botball [1] robots do not differ much from each other. Because of this it is necessary to focus on tuning parameters and developing a good strategy in order to compete against other teams. However, the preparation time for a competition is limited, and tuning the parameters through trial and error takes a lot of time. Because of that instead of trial and error the idea was to use evolutionary learning to improve accuracy and reduce time. Evolutionary learning is an algorithm based on Charles Darwin's theory of natural selection that precises parameters over generations. The applicability of this algorithm for robotics competitions was determined by simulating the Double Elimination Task of the compAIR[3] robotics competition.

A. Evolutionary Learning

A genetic algorithm [6] like an evolutionary algorithm is inspired by Charles Darwin's theory of natural evolution. This procedure applies the process of natural selection, where the individuals, that display the best characteristics for survival, are selected for mating. In this case, at each step, the genetic algorithm [7] selects the fittest individuals from the current population to be parents and uses them to produce children for the next generation which inherit the parameter properties of their parents. What constitutes the fittest individual depends on the problem the algorithm has to solve - in this case the generation with the most points scored. After the first generations, the parameters were significantly adjusted because the range of values is not yet limited by previous generations. Over time, due to the continuous generation of children and the selection of the most valuable individuals, this algorithm offers an even higher accuracy, as the range of values that the next generation can take on becomes smaller after each generation and is based on the parameters of its parents.

[1]Botball is a robotics competition developed by kipr for students to use science, engineering, technology, math, and writing skills to design, build, program, and document robots in a hands-on project that reinforces their learning.[2]

II. State of the Art

Evolutionary Algorithms (EA) and Artificial Intelligence (AI) were first acknowledged in the 1960's from pioneers like Nils A. Barricelli[2] Even though almost 70 years passed, modern researchers are still highly interested in such technologies. There are various use cases for EA today, such as "Evolutionary Tuning of multiple support vector maschines parameters" (SVM) [4] which is used for tuning hyperparameter to improve the outcome for a kernel or "Grey Wolf Optimizer"[8] which is capable of imitating the habitat and the hunting behaviour of a grey wolf by using a new written EA specified for this experiment. In comparison to the already mentioned papers, the most distinctive feature that distinguishes ours from the others is our approach by using a selfmade simulation engine which is distributable.

III. Problem Statement

To test the applicability of evolutionary learning to robotics competitions, the Double Elimination (DE) task of compAIR 2022 [3] is simulated. The compAIR is a robotic contest which first appeared in February 2020. It's only one of few Mathematics, Informatics, Natural Science, and Technology (MINT) based competitions in Austria hosted by robo4you[3]. The core idea of the contest is to bring young people into the MINT division. It consists of two challenges called Seeding

[2]Nils A. Barricelli was a Norwegian-Italian mathematician who simulated the evolution of populations of artificial organisms.[9].

[3]Robo4you is a association founded by Michael Stifter, Harald R. Haberstroh and Christian Reischl supported by the Secondary Technical College in Wiener Neustadt, Austria

(SE) and Double Elimination. For the experiment the concept of DE is used to demonstrate the effect of evolutionary learning. In this challenge two teams compete on the same table for one and a half minutes as displayed in figure 1. The table has an empty flat bottom and is surrounded by a border. The starting position of the bots is the opposite corner of each other. However, in the simulation it is made use of only one bot to get a clear result of evolutionary learning independent of the behaviour of the second bot. In addition, the game ran for only 15 seconds to have a shorter total simulation time. There are also 100 points awarded for reaching a goal to get a clearer result for the average points after each generation.

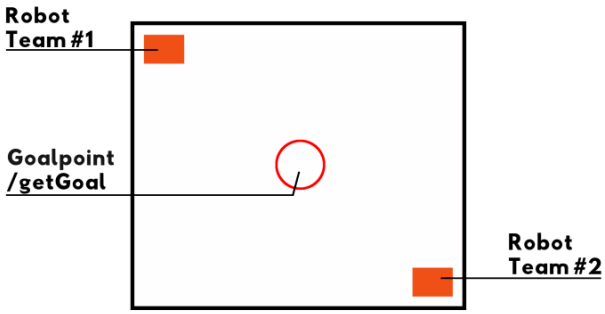

Figure 1. Double-Elimination Board of the compAIR Competition. The goal is represented by the red circle. The bots are displayed as red rectangles.

The aim is to score more points than the opponent before time runs out. Teams can obtain points by driving into circular areas that are randomly generated on the game table. Besides these goals we added two items that can be collected:

Definition 1 (Teleporter)**.** *When the Teleporter is picked up, the bot gets teleported to a random location. This item can not be found in the official double elimination tournament of compAIR. But we implemented it to give the bot more possibilities to evolve its strategy.*

Definition 2 (Oil-spill)**.** *The Oil-spill performs as an obstacle. When the Bot drives through it, the robot spins randomly for 30ms.*

IV. The genes

The bot starts with a basic strategy that can be adapted with the help of 5 genes.

1) The first gene is the backwards gene. It determines whether the bot can drive backwards. This gene can be a number between 0 and 100. Everything bigger than 50 enables the bot to drive backwards. Everything less than 50 disables it. We chose this range of numbers to give the algorithm a chance to slowly evolve in one direction.
2) The goal-gene gives the goal a weighting. This genes values also ranges between 1 and 100 for the algorithm to slowly evolve.
3) The Teleporter-gene gives the Teleporter a weighting. This genes values also ranges between 1 and 100 for the algorithm to slowly evolve .
4) The Oil-spill-gene gives the Oil-spill a weighting. This genes values also ranges between 1 and 100 for the algorithm to slowly evolve.
5) The last gene is the speed gene it determines the maximal speed percentage the bot is able to drive. Since these are percentage values, the values range between 0 and 100.

These genes will be generated and inherited to the children by the parent which achieved the most points.

V. Decision making with genes

During manoeuvring the bot is directly affected by the weight the genes are giving different items and the goal. The equation for evaluating the preference of certain items can be seen in equation 1.

$$val(w) = 2 - \frac{w}{100} \tag{1}$$

Figure 2. Equation for estimating the value of items

The variable w stands for the weight the gene generated for this item. The value *val* stands for the importance the collectable gets. Another parameter which is important in deciding to which item or goal to drive is the distance to the respective target. So the decision to which object the bot should drive can be seen in the equation 2.

$$I = cm \ * \ val(w) \tag{2}$$

Figure 3. Equation for estimating the preference of items

I stands for the preference value the item or goal got assigned. The bot then moves to the item or goal with the smallest preference value. An example code for this procedure can be seen in the following listing 1.

VI. Turning with genes

If the first gene allows the bot to drive backwards this can result in a much faster alignment since the bot only has to rotate for less then 90 degrees. To evaluate the optimal rotation, the bot calculates a line to the destination and decides by the gradient how it rotates. If the gradient is bigger than 90 degrees the bot turns until the gradient is 180 degree and then drives backwards.

VII. Experiment

For testing the erlang vm [1] is used to run the game simulation which can be accessed via beam[5]. This simulation enabled us to set the position and angle of the bot. After every angle or position update the simulation determined how many items or goals were currently in the game and if they don't exceeded the maximum amount of entities, randomly

```
#function for running the game
defp run_game(weights,time, name) do
    #[...]

    # simulates
    #for a specific time
    if (time is not over) do
        #gets the bots position
        bot = get_Bot()
        items = sim.get_items(name)

    # calculates the score
    # of the Items
    for item in items do
        type = get_type(item)
        weight = get_weight(type)

        #add distance
        score = weight * distance
        Map.put(item, score)
    end

    #drives to the Item/goal
    #with the highest score
    goal = Highest_score_Item()
    drive_to(goal)
    run_game(max_speed, weights, rev, time, name)
    end
end
```

Listing 1: Example code of function for running the game and evaluating the weight for items. Source Code: GitHub repository

spawned a new one. Furthermore, each time a position was set, a check was made to ascertain whether the change was valid and whether the bot was within the range of an item or goal and therefore activated it or scored a point. In addition, each time the bot collected a goal, an async thread handle is returned, which the bot had to await. This async thread handle is used to execute effects of items without prolonging the server request, so that it remained inaccurate even in the case of 2 simultaneous bots.

VIII. Results

After the first two generations, there is a clear increase in the score, as displayed in graph 4. After that the points increase steadily. Furthermore, a steady increase in points is to be noted as well.

As expected items which hinder the bot from scoring points such as the Teleporter and the Oil-spill have decreasing

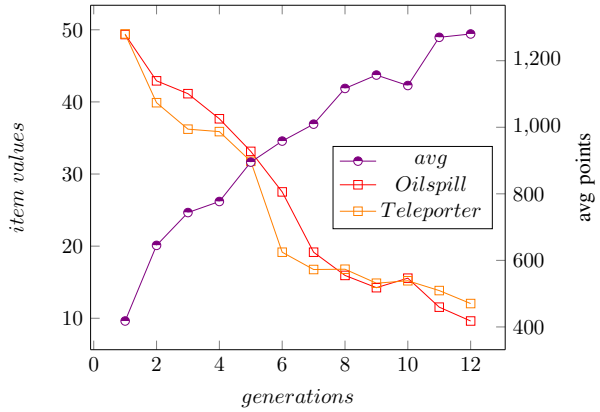

Figure 4. Graph depicting the value of the Oilspill and Teleporter parameters in relation to the increase of average points

weights over generations. Whereas the speed and goal weight experience an incline as can be seen in graph 5. The parameter for driving backwards only fluctuates around the starting value. This is achieved because the bot drives backwards once the value is over 50 any other variation in value does not influence the behavior of the bot.

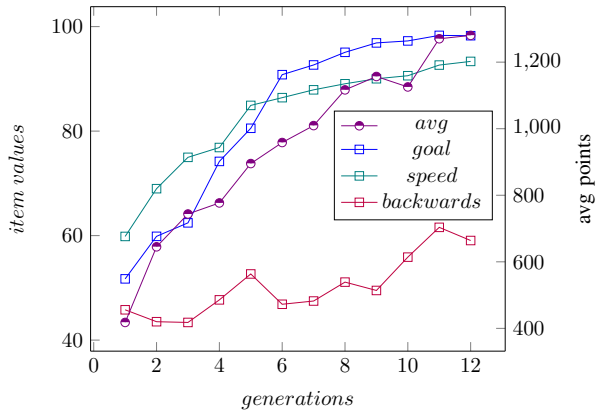

Figure 5. Graph depicting the value of the goal and speed parameters in relation to the increase of average points

IX. Conclusion

After the experiment, it can be concluded that using generational learning to tune parameters is accurate and can be used in various robotics competitions, such as tuning motor values for more accurate movement. Although setting up the simulation can be time consuming, the result allow for more efficient parameter tuning. In order to use these parameters effectively outside of the simulation, environmental influences such as uneven ground must also be taken into account. However, simulation cannot be used to develop a strategy from scratch. There must always be a basic idea of what the bot can do, because it is on this basis that the parameters for the algorithm to be tuned are selected.

ACKNOWLEDGEMENT

The authors would like to thank Dr. Michael Stifter, robo4you and the Secondary Technical College Wiener Neustadt for the great support in realizing this project.

REFERENCES

[1] Ericson AB. *Erlang/OTP Documentation*. URL: https://www.erlang.org/doc/.

[2] *Botball*. URL: https://www.kipr.org/botball.

[3] *CompAIR*. URL: https://comp-air.at/.

[4] Frauke Friedrichs and Christian Igel. "Evolutionary tuning of multiple SVM parameters". In: *Neurocomputing* 64 (2005). Trends in Neurocomputing: 12th European Symposium on Artificial Neural Networks 2004, pp. 107–117. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2004.11.022. URL: https://www.sciencedirect.com/science/article/pii/S0925231204005223.

[5] John Högberg. *A brief introduction to BEAM*. URL: https://www.erlang.org/blog/a-brief-beam-primer/.

[6] Vijini Mallawaarachchi. *Introduction to Genetic Algorithms — Including Example Code*. URL: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

[7] Naveen Upreti & Deepti Gupta Manoj Kumar1 Mohammad Husian. *GENETIC ALGORITHM: REVIEW AND APPLICATION*. URL: https://dx.doi.org/10.2139/ssrn.3529843.

[8] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. "Grey Wolf Optimizer". In: *Advances in Engineering Software* 69 (2014), pp. 46–61. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2013.12.007. URL: https://www.sciencedirect.com/science/article/pii/S0965997813001853.

[9] institute for advanced study. *Nils A. Barricelli*. URL: https://www.ias.edu/scholars/nils-barricelli.