Evolutionary parameter optimization in Robocode

Using evolution to optimize paramaters in Robocode tanks

Leendert van Achteren

University of Groningen

leendert@vanachteren.net

Abstract

This paper describes an attempt to find out whether it is possible to use evolution to determine how strong separately developed single behaviors should be present in the final behavior of a robot and to determine the values of certain parameters used in robot functions, in order to be successful. Where successful is defined being able to win over its competitors. The study was done using separate behaviors put in a subsumption architecture and using potential fields. Evolution was taking place to find an optimal set of weights for the different layers in the subsumption architecture and parameters for different functions used by the robots. Unfortunately the experiments were stopped before having reached an optimum. The fitness was improved, but not enough to on average beat the opponents. In the end, some suggestions for improvements on the experiments and for further research are done.

1 Introduction

Robocode was first developed and introduced by IBM in 1991 as a programming game which made it fun to learn the Java programming language[i] [ii]. A user programs one robot or a team of robots as java objects, which battle against other robots on screen[iii]. The behaviors of the robots depend completely on how they are programmed. Different behaviors can be programmed in one robot. Different behaviors should even be programmed in order to be able to be a successful robot in terms of winning battles. A robot can for instance be very good at staying alive, but will be even better when he also kills other robots.

Robots in Robocode are easy to program. The programmer can therefore focus on developing the behavior he wants. And as the programming language is java, there is no problem to make the behavior very advanced. The advanced behavior can be tested visually with the on screen battle as well as with the statistical information using the Robocode output.

The goal of this research is to find out whether it is possible to use evolution to determine how strong separately developed single behaviors should occur in the final behavior of a robot and to determine the values of certain parameters used in these behaviors, in order to be successful. We define successful as a team of robots being able to win over its competitors. Evolution on Robocode was done before, but never on the weights in a subsumption architecture[iv] [v].

2 Method

The different separate behaviors are combined using a subsumption architecture as discussed below. The movement behaviors are combined. This is done using potential fields. These are also discussed below.

2.1 Subsumption architecture

A subsumption architecture is a robot control structure invented by Brooks[vi]. It is organized in different layers, where each layer represents a behavior of the robot[vii]. Higher layers subsume the role of lower layers when they wish to take control. Behaviors can be developed separately from the lower level up, with each level increasing the level of competence of the complete robot. Whereas a traditional decomposition of a mobile robot control system into functional modules is horizontally oriented [Figure 1], the subsumption architecture is vertically oriented [Figure 2].

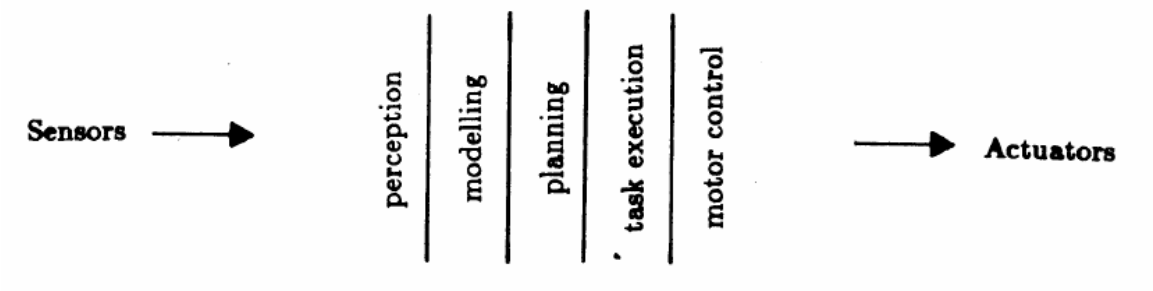

Figure 1 **Traditional decomposition of a mobile robot control system, Brooks '86**

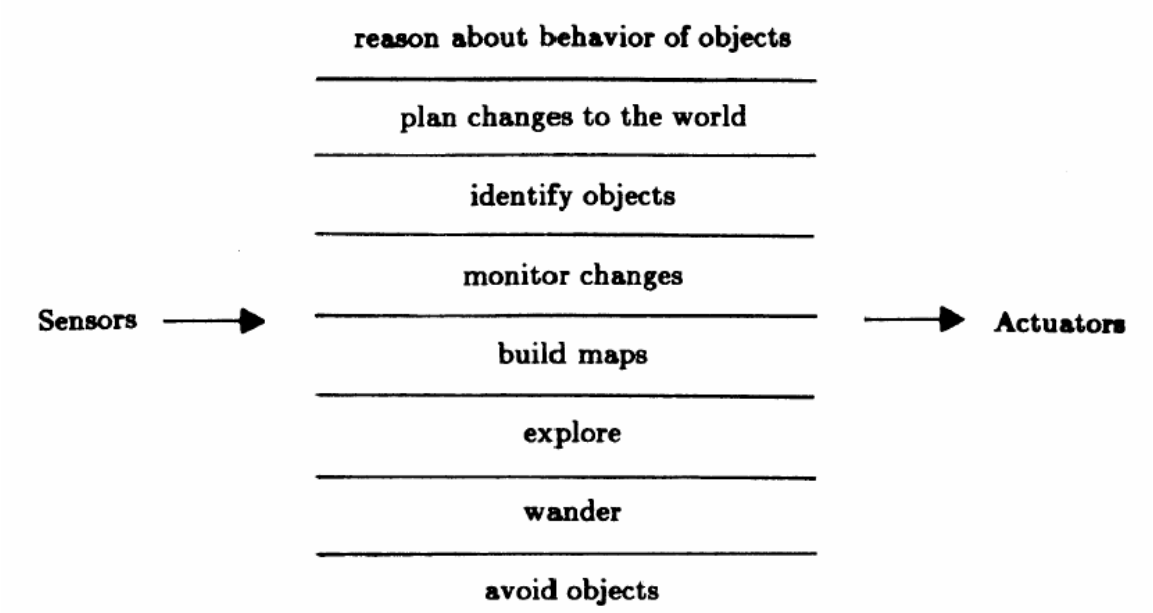

Figure 2 **Subsumption architecure, Brooks '86**

Using the example in figure 2; if a robot wanders, it is still also avoiding objects. The layer "avoid objects" is subsumed by the layer "wander" in order to still be avoiding them.

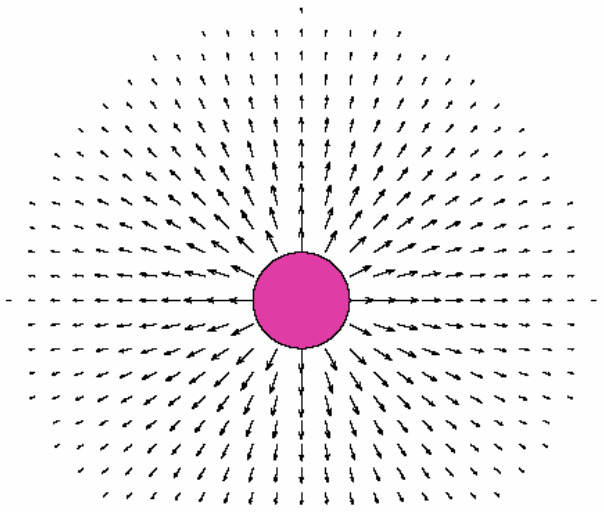

Figure 3 **A reject potential field, M.A. Goodrich, 2000**

2.2 Potential fields

In Figure 2, "explore", "wander" and "avoid objects" are all layers representing movement behaviors. When the "wander" layer subsumes "avoid objects" and is subsumed by "explore", the outcome of each layer can be regard as the vector belonging to the current position of the robot in a potential field[viii]. Each behavior outputs a desired output vector. The avoid objects behavior for example, outputs a vector that makes the robot move away from one or more objects [Figure 3].

Not only do the vectors point away from the, in this case one, object. The vectors tend to enlarge when the location in the potential field is closer to the object.
The movement that is eventually sent to the actuators of the robot is the weighted average value of the vectors corresponding to the different movement behaviors.

2.3 The behaviors

The layers in the subsumption architecture each have a certain weight. These weights are to be evolved. The influence of the different behaviors on the final behavior of the robot depends on the weight of the specific layer compared to the sum of weights. The following behaviors are built in the robots.

Movement:

All the six movement behaviors get an evolved weight. After that the vectors are combined as described above.

random	A random movement is performed in order to move in a way that is hard to predict by opponents.
wallavoidance	The walls of the arena are avoided to prevent collision damage
friendavoidance	Friendly robots are avoided to prevent collision damage.
teamattraction	The robot moves toward the center of the group in order to be protected by its teammates.
enemyavoidance	Enemy robots are avoided.
targetattraction	The robot moves towards its target.

Turret:

The turret behavior targetallignment is dominant. Only if there is no target defined, the other two turret movement behaviors will influence the turret movement. All three behaviors get an evolved weight in order to determine how far the turret will move regarding the current position.

targetallignment	The turret aligns with the target.
pointfromteam	The turret away from its team.
velocityalignment	The turret aligns with the direction the robot itself is moving.

Radar:

Turnradarright	The radar spins around continuously looking for other robots.

Communication:

broadcast_friends	Broadcasts the location of detected friendly robots.
broadcast_enemy	Broadcasts the location of detected enemy robots.

2.4 The parameters

The following parameters are subject to evolution.

bDIV	Random movement takes place under a maximum angle.
bBCK	Determines how far the robot needs to drive back in case of a collision.
bMOV	Determines how far the robot moves each turn.
tPOW	The power of a gunshot.
T2I	The time to impact determines how long a bullet may be on his way to the target at maximum in order to shoot.
tMRG	Determines radians in which no friendly tank may be present in order to fire.

2.5 Evolution method

The objective of evolution in this research is to search for and find the optimal setting of weights for the subsumption architecture and the optimal settings. Evolution as described by Darwin is based on the following aspects[ix]:

- Better fitted individuals more often survive and they have stronger influence on forming new generations;
- Individual in new generation is formed by recombination of parent's genetic material;
- From time to time mutation (random change of genetic material) takes place.

Evolution is a sort of genetic algorithm and in this case has the following structure:

- **search space**	The nine subsumption weights as well as the six parameters can get any value. Besides that, also the mutation speed of each weight and parameter evolves (how much a value changes when it is mutated). Third of all, each weight has a chance of being mutated;
- p**opulation**	The population exists of all possible combinations of the nine weights and their search space;

- **string space** The string space consists of an array of three times nine double digit weights;
- **functions for conversion** The nine weights each have there own position in the array;
- **set of genetic operators** Mutation is used for getting new genes.
- **fitness function** The fitness of each item is determined by the sum of scores against the three selected opponents. The opponents are selected for being able to win a match and for not giving errors too often while battling.

The following steps were taken to let the evolution take place:

1. **Initialization** A random population of 32 robots is created. A population of 32 is fairly small, but unfortunately, due to time restrictions it was not possible to use a larger population. A larger population increases the chances of finding a very good solution. There is also improvement possible in the "random" population generator. The genes differ from team to team, but all the start values are close to the value of 0.5.
2. **Evaluation** The battles take place, each team battling against the same three selected opponents. Three opponents may not be enough. As we also only battle each team once, the coincidence factor for getting a good result from the battles is high. This is especially not good with a small population of 32.
3. **Selection** The numbers battle results for teams number 1 and 2, 3 and 4, … , 31 and 32 are compared. The winner of each pair continues to the next round.
4. **Recombination** The losers of the pairs described above are being replaced by a mutation of the winner. After being replaced, the losers get a random position in the array of 32, in order not to battle the same opponent again.
5. **Repeating steps 1.-4.** The steps are repeated until the population on average beats the selected opponents. Or until it appears that no further progress by can be reached by this evolution.

The evolution method described above is partly based on the Tournament Selection method[x].

3 Results

Unfortunately time was too short to determine how far the genes could be optimized. After 23 rounds of battles, each taking three quarters of an hour, with an average of one human intervention per round, necessary to help the evolution continue after an error, there was no optimum reached. The fitness was still increasing as can be seen in Table 1 and Figure 4.

Rnd	Fitness	Trend
1	-13006	-13610
2	-14127	-13520
3	-14255	-13430
4	-12797	-13340
5	-13712	-13250
6	-13405	-13160
7	-13199	-13070
8	-12959	-12980
9	-11962	-12890
10	-12947	-12800
11	-11931	-12710
12	-13200	-12620
13	-13257	-12530
14	-12205	-12440
15	-11499	-12350
16	-11865	-12260
17	-12239	-12170
18	-12141	-12080
19	-11411	-11990
20	-11817	-11900
21	-11817	-11810
22	-12408	-11720
23	-11136	-11630

Table 1 Fitness scores per round

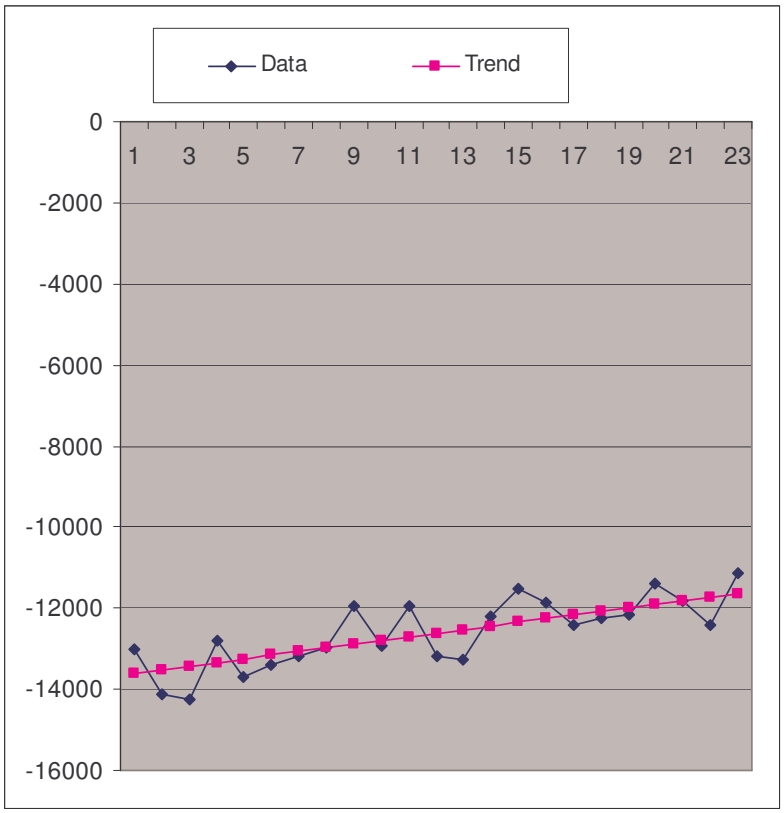

Figure 4 Results of evolution during 23 rounds

Where the fitness at first decreases from the initial average of −13006 to −14255 in round 3, it then starts to increase gradually with a best result of −11136 in round 23.

4 Discussion

This research unfortunately did not show whether it is possible to use evolution to determine the influence of single behaviors in the final behavior of a robot and to determine the values of certain parameters used in these behaviors, in order to be successful. We did however show that the process of evolution improves the fitness of the entire population. And this triggers us to do further research on the topic.

4.1 Significance

This paper proved that evolution can help in improving fitness scores by changing the weights in a subsumption architecture and parameters in certain functions. At least this is the case for robots with a very low fitness score. It was already known that evolution can have a role in Robocode development, but never was

this shown for optimizing the weights in a subsumption architecture.

4.2 Future research

In future research it will be important to take a larger population of perhaps 128 pieces, with genes that are more random at start. In the current situation the genes are too much alike each other and the fitness advantages have to occur out of mutation instead of that there are some "successful" genes already. This process takes too much rounds and therefore too much time.
Another problem was that human intervention was still needed in order to keep the evolution running. It would be better to make a workaround for this that in case of occurring errors, the system still continues. This way, more battles will be fought when the computer runs at night.
The last problem was the large coincidence factor in winning a battle. Three battles to determine the fitness value might be too small. It may be safer to take 6 teams or so, or if there are not 6 teams available or the team has to perform against these three specific teams, the number of batches against each team can be increased.

---

[i] http://www.alphaworks.ibm.com/tech/robocode

[ii] http://robowiki.net/cgi-bin/robowiki?History

[iii] Robocode as a tool for simulation of autonomous systems, S.Rzydzik, 2003,http://kpkm.mt.polsl.gliwice.pl:8083/papers/52.pdf

[iv] Attaining Human-Competitive Game Playing with Genetic Programming, M.Sipper,Y.Azaria,A.Hauptman,Y.Shichel, 2005, http://www.cs.bgu.ac.il/~sipper/papabs/gpgames.pdf

[v] GP-Robocode: Using Genetic Programming to Evolve Robocode Players, Y.Shichel,E.Ziserman,M.Sipper, 2003, http://www.cs.bgu.ac.il/~sipper/papabs/eurogprobo-final.pdf

[vi] A robust layered control system for a mobile robot, R.A.Brooks, 1985, http://people.csail.mit.edu/brooks/papers/AIM-864.pdf

[vii] Evolution of the layers in a subsumption architecture robot controller, J.Togelius, 2003, http://cogprints.org/3306/01/Dissertation.pdf

[viii] Potential Fields Tutorial, M.A.Goodrich, 2000, http://students.cs.byu.edu/~cs470ta/readings/Pfields.pdf

[ix] Fine Grained Tournament Selection for the Simple Plant Location Problem, *V.Filipović,J.Kratica,D.Tošić,I.Ljubić, 2000,* http://www.matf.bg.ac.yu/~vladaf/Papers/Filipo00.pdf

[x] A Comparison of Two Competitive Fitness Functions, L.Panait,S.Luke, 2001, http://cs.gmu.edu/~sean/papers/twocompetitive.pdf