Adaptivity on the Robot Brain Architecture Level Using Reinforcement Learning

Tijn van der Zant

University of Groningen
Artificial Intelligence dept.

Abstract. The design and implementation of a robot brain often requires making decisions between different modules with similar functionality. Many implementations and components are easy to create or can be downloaded, but it is difficult to assess which combination of modules work well and which does not. This paper discusses a reinforcement learning mechanism where the robot is choosing between the different components using empirical feedback and optimization criteria. With the interval estimation algorithm the robot deselects poorly functioning modules and retains only the best ones. A discount factor ensures that the robot keeps adapting to new circumstances in the real world. This allows the robot to adapt itself continuously on the architecture level and also allows working with large development teams creating several different implementations with similar functionalities to give the robot biggest chance to solve a task. The architecture is tested in the RoboCup@Home setting and can handle failure situations.

Keywords: adaptivity, behavior selection, RoboCup@Home, robot brain development, interval estimation algorithm, reinforcement learning.

1 Introduction

Just a decade ago the main problem with the development of robotic brain was that every group had to implement most algorithms themselves. Many development groups were already happy if the most important modules could be constructed before a deadline such as the RoboCup[6] competitions. In the mean time many groups realized this and groups with common goals started to share code using the Internet. RoboCup assisted in this effort by steering developments through the competition and stimulating teams to share their code. Slowly the problem of developing robotic systems has steered away from *only* building components towards the orchestration of components created by other groups into an overall architecture adding only a few functionalities. This allows the development teams to focus on their own expertises, hopefully also leading to the publication of their code. In the RoboCup@Home competition, for example, it is very common to see groups 'gluing' together all sorts of hardware and software components, focusing only on a few aspects. This is essential since the development of intelligent robots that can operate in the real world requires a huge effort. It is unlikely that any university-based development group can accomplish a good performance in the RoboCup@Home competitions if they were to build all the modules and components from scratch.

The problem nowadays is how one can choose between all the different implementations and components. The following list represents only a small portion of the possibilities but gives an impression of the overwhelming amount: Carmen[1] [7], Player/Stage[2] [4], MRPT[3], OpenCV with many functions[4], ROS with over 2000 modules[5], OpenSlam with dozens of SLAM algorithms[6], RoboRealm, a commercial robotic vision package[7].

This multitude of modules create the problem of deciding which module to use. How can one determine which SLAM or robot vision algorithm will work for the robot? Which behavior implementation is the best and in what situation is it the best? Which combination of modules are most likely to work correctly together? How can behaviors or other modules automatically be tweaked to improve performance? These are important questions that this article addresses. The main idea is that developers provide implementations for the robot to use, but that the robot decides which implementation to use in a specific situation. This decision process is steered through interaction with the environment and the users interacting with the robot.

A robot can be viewed as a mobile empirical data gathering device. It should therefor be possible that the robot is checking how well it is performing. This is the idea behind many learning algorithms but very few algorithms operate on the architecture level. Researchers are working on approaches to automatically develop components [2] stimulated by the concepts behind Cognitive Developmental Robotics [1]. Also there is focus on the construction of hierarchical behavior systems using reinforcement learning [9,3]. The point of view in this article is different in the sense that it assumes that there is a multitude of modules with similar functionalities and the robot has to choose between them in an automated manner. The components can be manually crafted, learned or a combination of the two. This is useful for the comparison of machine learning algorithms and for large development groups where many people, for example students, create similar behaviors for the robot.

For example, if the robot gets the command to get a certain item in a certain location, how can the robot decide which `GoToLocation(location_1)`, `SearchForObject(object)`, `GraspObject(object)`and for going back `GoToLocation(location_2)` to choose from? This is a common problem and can be solved using some strictness on the behavior implementation level. This will be explained in the next section.

[1] Carnegie Mellon Robot Navigation Toolkit (`http://carmen.sourceforge.net/`)
[2] The Player/Stage Project (`http://playerstage.sourceforge.net/`)
[3] The Mobile Robot Programming Toolkit (`http://babel.isa.uma.es/mrpt/index.php/Main_Page`)
[4] The Open Computer Vision Library (`http://sourceforge.net/projects/opencv/`)
[5] `http://www.ros.org/`
[6] `http://openslam.org/`
[7] `http://www.roborealm.com/`

2 On-Line Adaptation

It happens that the developers arrive with the robot at a certain location and the algorithm that worked well before is not working so well anymore? This is a common tragedy that many development groups have experienced. It happens at places with dynamic environments, for example during competitions such as RoboCup@Home [11] where robots have to operate in an apartment and in a shopping mall. The specifications of the environment are poorly defined and the situations the robot finds itself in are various. Also, in the last years the General Purpose Service Robot test has been introduced where the robot is being put in situations it cannot solve. The mechanisms described in this article solve this problem by keeping track of how well the modules performed in previous, but similar, situations. Using the Interval Estimation [5,12] algorithm the robot can calculate the 95% confidence interval and decide when there is less than 2.5% probability to succeed and revert back to a default action.

This section will explain the adaptive architecture bottom up, starting with the adaptivity on the behavior level. First learning on the level of simple behaviors is explained, gradually building up towards more complex behaviors. Through generalization over the machine learning it is shown how non-behavioral components such as SLAM and robotic vision components can be taken into account.

It is important to keep in mind that we assume that the Markov property holds in the selection of the components. This means that we assume that it does not matter how the robotic system arrived in the state it is in at the moment and that by optimizing the actions in the current state the entire architecture is optimized. It is a research question whether this holds in the real world, but for the time being we explicitly assume it holds.

2.1 Interval Estimation Algorithm

Interval Estimation (IE)-learning [5] is a method for dealing with the exploration/exploitation dilemma in reinforcement learning [8]. The IE algorithm is a method that allows the robot to adapt itself on-line and real-time to the situation at hand. First the algorithm is explained. Then it is shown how the algorithm can be adapted to keep exploring in case of a changing environment, such as changes in user preferences, changes in the physical environment or changes in light conditions.

The problem is that a learning agent wants to select its current best behavior as much as possible (to exploit) and explore to find the optimal behavior at the same time. Since the agent cannot explore and exploit at the same time, there is a dilemma. IE has been successfully applied in [10] for model-based exploration in simulation. The IE-algorithm select the optimal behavior and can be extended to take state information into account.

For any state s the best action a^* has to be chosen. By trying the action, the environment gives feedback about the reward $r_a(t)$ for the action a selected at time t. The optimal action corresponds to:

$$a^* = \arg\max_a E(r_a|a)$$

Where E denotes the expectancy operator. However, we do not know the true expected reward, but only obtain samples around this average. From n samples we can construct sample averages Q_a:

$$Q_a = \frac{\sum_{i=1}^{n} r_a(i)}{n}$$

Purely selecting the action with highest Q_a value does not work well, exploration is necessary [8]. The IE algorithm stores an estimate of the expected reinforcement of an action and some information about how good the estimate is [5]. The IE algorithm estimates the confidence interval of the average of the data obtained when executing actions.

For small amounts of data, typically for a robot, the student T distribution is used to estimate the confidence interval. The upper bound of the confidence interval can be calculated using the following standard statistics, with n as the number of trials a behavior has been selected and $\sum_{i=1}^{n} r_a(i)$ the total reinforcement a behavior a has received. The upper bound of a $100(1-\alpha)\%$ confidence interval for the mean of the distributions is calculated by

$$nub(n, \sum_{i=1}^{n} r_a(i), \sum_{i=1}^{n} r_a(i)^2) = Q(a) + t_{\alpha/2}^{(n-1)} \frac{s}{\sqrt{n}}$$

with $Q(a) = \frac{\sum_{i=1}^{n} r_a(i)}{n}$ as the sample mean, and

$$s = \sqrt{\frac{n \sum_{i=1}^{n} r_a(i)^2 - (\sum_{i=1}^{n} r_a(i))^2}{n(n-1)}}$$

being the standard deviation. $t_{\alpha/2}^{(n-1)}$ is the Student's T function with n-1 degrees of freedom at the $\alpha/2$ confidence level. The IE algorithm selects the actions with the highest or lowest upper bound, depending on maximization or minimization, and is therefore optimistic about the results. If the spread of the data points is high, then the interval is large. As there are more data points collected through time, the interval shrinks because there is more information available.

Initially, the first action will be chosen at random and all actions will be tried out at least once (with no information the upper/lower bounds are: $0 \pm \infty$), but as the bounds tighten, the better the choices become. The IE algorithm balances exploration (a big interval in case of a high uncertainty) with exploitation. The upper bound can be high because it either has little information about the action or because the entire confidence interval is high and the action is good. The DeMoivre-Laplace theorem states that it will converge to its true underlying values in the limit. In practice IE sometimes gets stuck with a suboptimal behavior, because we do not have an unlimited amount of runs on real robots and robots may suffer from "unlucky" experiences.

On a real robot the distributions of the reinforcement values might not be distributed in a way that is favorable for the IE algorithm. While doing experiments on the robot the algorithm got stuck sometimes. To counteract this several solutions exist: one can choose to randomly drop a data point from a randomly chosen action, causing an increment in the size of the interval; or by introducing a discount factor to let actions further in the past have less influence on the calculation of the confidence interval.

With the behavior-based IE algorithm the programmer can give the robot several solutions to solve a particular problem. The developer actively puts the domain knowledge in the system as a hypothesis to be explored. Together with a evaluation criterion (such as time, amount of grasped objects or any other criterion observable for the robot) the robot is able to explore the different behavioral hypotheses and chooses the one that, at that moment in time, it has the highest confidence in that it is the best one. The robot explores and as it gains experiences it will explore less and choose the best action.

It might be the case that the environment changes and that the robot has to adapt itself. For this reason it is useful to have several different behavioral hypotheses ready to use. It is easy to adapt the algorithm to remain adaptive: diminish the influence of a data point related to the amount of time that has past. For example, the amount of data points between the present and when the data point was collected can be used with a discount factor. Both the data point and n can be multiplied with discount factor d using a for the amount of data points that have past, as in d^a. If d is small (< 0.8) then the algorithm will try new hypothesis rapidly, if d is large (> 0.97) then the algorithm is more careful with choosing a new hypothesis.

2.2 Applying the Interval Estimation Algorithm on the Behavioral Level

For every behavior we have several implementation, a postcondition and a criterion for the IE algorithm. This section demonstrates the behavior selection for our grabbing behavior. It is not important how the grabbing behavior is created. In this case two bachelor students created six different implementations using a visual programming environment, since this was the simplest and fastest method to bootstrap the robot. In another case, where the robot has to follow a person, we have two handcrafted implementations and several instantiations using reinforcement learning with different parameter settings.

We used a set of object to train the grabbing behavior on, being a box, a dessert cup (named 'dessertCup' during the tests), a ball, a bottle (named 'bottleP'), a regular cup, a sponge, some tape, a can and a pringles box. The behavior selection was bootstrapped by forcing the robot to use every behavior on every item six times. The results are shown in table 1 and in figure 1.

2.3 Applying the Interval Estimation Algorithm on the Architecture Level

The application of the IE algorithm can be fully automated if one takes into account that the reward function should be inspectable by the robot. This implies that every behavior that is going to be used by the behavior selection mechanism

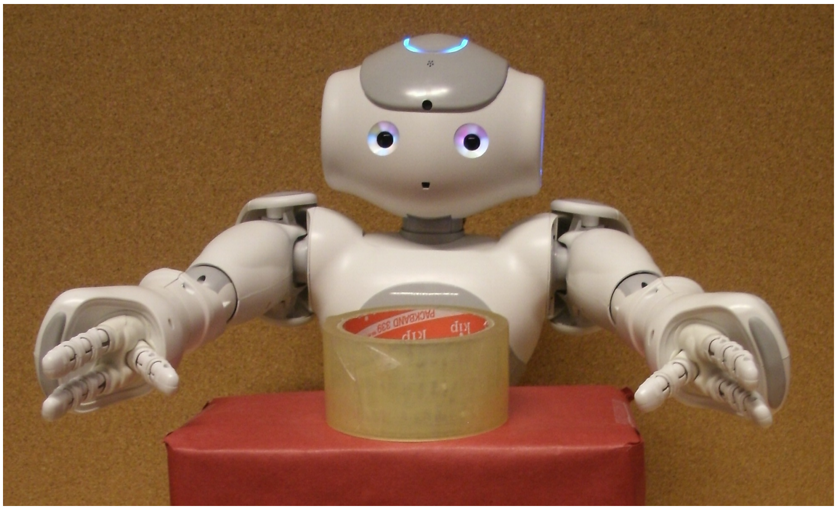
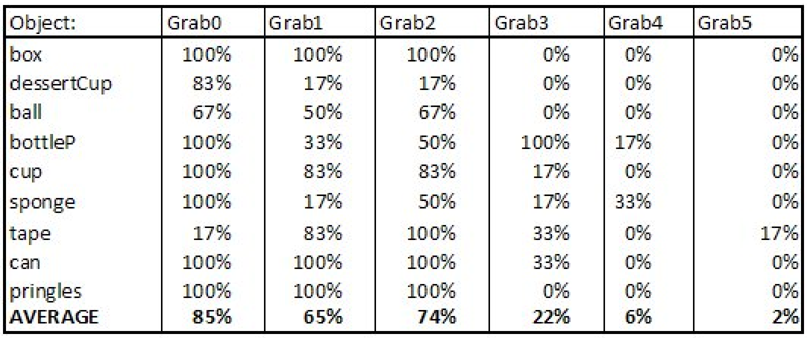


Fig. 1. Figure 1: Setup of the Nao grabbing an object



Object:	Grab0	Grab1	Grab2	Grab3	Grab4	Grab5
box	100%	100%	100%	0%	0%	0%
dessertCup	83%	17%	17%	0%	0%	0%
ball	67%	50%	67%	0%	0%	0%
bottleP	100%	33%	50%	100%	17%	0%
cup	100%	83%	83%	17%	0%	0%
sponge	100%	17%	50%	17%	33%	0%
tape	17%	83%	100%	33%	0%	17%
can	100%	100%	100%	33%	0%	0%
pringles	100%	100%	100%	0%	0%	0%
AVERAGE	**85%**	**65%**	**74%**	**22%**	**6%**	**2%**

Fig. 2. Table 1: Average success of grabbing behaviors

of the IE algorithm has a postcondition which can be checked upon by the robot. The list below gives some examples that we apply in our RoboCup@Home robot and the postconditions that we use. If the postcondition is satisfied the data is stored, to be used the next time the robot has to execute the behavior. PC means postcondition, IE the criterion to optimize using the IE algorithm using the postcondition.

GoTO(location) Go to a location.
PC: upon reaching the target location
IE: minimize time
Search(object) Search strategy for an object near to the robot.
PC: visual detection of the object
IE: minimize time

Grab(object) Grab the object, object should be in gripper
PC: object remains in fixed location in the camera image while the robot moves for at least ten seconds
IE: maximize success rate (boolean)

Get(object, location) Satisfy the following: `GoTo(location)`, `Search(object)` and `Grab(object)`
PC: `Grab(object) == True`
IE: none, it is contained in the sub-behaviors

This describes our basic methodology to optimize on the simplest level, only optimizing behavior implementations. While the robot is running, executing behaviors, perhaps executing behaviors with some semi-random variables such as `GoTo( location l, l elemOf listOfLocations)` it collects information about the success of the behaviors. Once the robot has executed a behavior a fair amount of times it can take into account more information. Our research indicates that if, on average, every implementation has to be tested at least six times, preferably a bit more. For example, the previous list of behaviors could then become the following:

GoTO(location, navigationAlgorithm) Go to a location using a specific navigation algorithm.
PC: upon reaching the target location
IE: minimize time

Search(object, imageProcessingAlgorithm) Search for an object near to the robot using a specific algorithm for vision.
PC: visual detection of the object
IE: minimize time

Grab(object, surface) Grab the object from a specific surface, such as table or floor
PC: object remains in fixed location in the camera image while the robot moves for at least ten seconds
IE: maximize success rate (boolean)

Get(object, location) Satisfy the following: `GoTo(location)`, `Search(object)` and `Grab(object)`
PC: `Grab(object) == True`
IE: none, it is contained in the sub-behaviors

In this second example it is clear that the implementation issues are hidden from the higher level behaviors, in this case `Get(object, location)`. This implies that higher level behavioral script can be generated using, for example, a reasoning mechanism while retaining the dynamics which is needed on the behavior implementation level. It is possible to specify (using natural language, for example) what, according to the user, is important for the robot to pay attention to. When the robot gets more information about the world, it can take more state information into account and react accordingly. We do not have a module at the moment to generate the natural language required for teaching the robot or for interpreting it, but expect to have this ready soon. This should allow us to create scripted behavior which is adaptive just by talking with the robot.

2.4 Automated Analysis of Module Combinations

The last step that can be performed is to test the combinations of states without being connected to a behavior. For example, for the selection of a vision algorithm and navigation method for a specific behavior it is possible to execute `behavior_1( someParameter, visionAlgorithm v, v elemOf listOfVisionAlgorithms , navigationAlgorithm n, n elemOf listOfNavigationAlgorithms)` many times. But if there are other behaviors, as in `behavior_N( anotherParameter, visionAlgorithm v, v elemOf listOfVisionAlgorithms , navigationAlgorithm n, n elemOf listOfNavigationAlgorithms)` then it is possible to calculate the upper and lower bounds of the 95% confidence interval of the combinations of `(v,n), v elemOf listOfVisionAlgorithms, n elemOf listOfNavigationAlgorithms`. If there is a new behavior XYZ that uses `(v,n)` as a part of its parameters, it can immediately use the learned confidence interval of the combinations of `(v,n)`. This can be generalized to any amount of modules (parameters) that are being used by the behaviors. This effectively speeds up the the learning and results in the transfer of knowledge from previous behaviors to new behaviors.

2.5 Completely New Situations for the Robot

The robot will, after training in an environment, be utilized in a new location. This does not mean that the learned knowledge is worthless, but it does mean that the uncertainty about the learned knowledge should be increased. Using the IE algorithm this is very easy to do. If one replaces separate data points with the average of those data points, the algorithm will either make the same decisions if the learned knowledge is correct, or it will start exploring using as a start the modules with the highest chance of actually being the best. This manner of a 'soft reset' implies that it is possible to have training schools for robots, and that the knowledge is transferable to new situations and perhaps even to new body types.

For the competition of RoboCup@Home we will reset our learned knowledge and use a discount factor of 0.9. The discount factor is not used with absolute time, with with the amount of times that the modules were executed between that data point and the present. This assures a high rate of adaptability. Because the algorithm is not intended to find the best modules, but to deselect against poorly functioning modules and poorly functioning combinations of modules, we are certain that the robot will behave properly. It is not important to have the best performance, because due to changing circumstances that is a moving target and therefor unattainable. For this reason it is difficult to give results beyond individual behaviors and modules. It is clear how one can select the best image processing and object recognition algorithm for a defined data set. In the real world one cannot measure this because of the rapid changes that occur. A robot that can deal with these changes and one that does not uses poorly functioning modules is much more preferable than a robot that operates perfectly in a constrained but does not know how to adapt itself if the situation changes.

3 Conclusion

This article discusses the use of the Interval Estimation algorithm for the selection of modules on the architecture level of a robot brain. It demonstrates how the algorithm works and how it can be utilized to perform behavior selection, the selection of interpreting modules such as navigation or robot vision, how the learned knowledge can be transfered to new behaviors and how a trained robotic system can get a 'soft reset' to adjust itself fast to a new environment.

The problem with the proposed method is that it is difficult to measure the exact success of it. We think that the success should not be measured in selecting the best combination of modules, but in not selecting poor combinations. Also, in the real world, due to its dynamic properties, there are no optimal solutions. There are good solutions, not so good solutions and bad solutions. The Interval Algorithm calculates this sliding scale and does not need parameterization to decide whether it should explore possible solutions or exploit the gained knowledge.

Acknowledgments. The author would like to thank Christof Oost and Eric Jansen for their work on the testing the grabbing behaviors.

References

1. Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., Yoshida, C.: Cognitive developmental robotics: A survey. IEEE Transactions on Autonomous Mental Development 1(1), 12–34 (2009)
2. Bellas, F., Duro, R., Faina, A., Souto, D.: Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots. IEEE Transactions on Autonomous Mental Development 2(4), 340–354 (2010)
3. van Dijk, S.G., Polani, D., Nehaniv, C.L.: Hierarchical Behaviours: Getting the Most Bang for Your Bit. In: Kampis, G., Karsai, I., Szathmáry, E. (eds.) ECAL 2009, Part II. LNCS, vol. 5778, pp. 342–349. Springer, Heidelberg (2011)
4. Gerkey, B.P., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics, pp. 317–323 (2003)
5. Kaelbling, L.P.: Learning in Embedded Systems. MIT Press (1993)
6. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: RoboCup: A Challenge Problem for AI. AI Magazine 18(1), 73–85 (1997)
7. Montemerlo, M., Roy, N., Thrun, S.: Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 2436–2441 (2003)
8. Sutton, R., Barto, A.: Reinforcement Learning: an Introduction. MIT Press (1998)

9. Vigorito, C., Barto, A.: Intrinsically motivated hierarchical skill learning in structured environments. IEEE Transactions on Autonomous Mental Development 2(2), 132–143 (2010)
10. Wiering, M., Schmidhuber, J.: Efficient model-based exploration. In: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 6, pp. 223–228. MIT Press/Bradford Books (1998)
11. Wisspeintner, T., van der Zant, T., Iocchi, L., Schiffer, S.: RoboCupHome: Scientific Competition and Benchmarking for Domestic Service Robots. Interaction Studies 10(3), 392–426 (2009), http://dx.doi.org/10.1075/is.10.3.06wis
12. der Zant, T.V., Wiering, M., Eijck, J.V.: On-line robot learning using the interval estimation algorithm. In: Proceedings of the 7th European Workshop on Reinforcement Learning, pp. 11–12 (2005)