Graph Memory Development in a Robot Control Architecture

Patrick McDowell, Cris Koutsougeras
Department of Computer Science and Industrial Technology
Southeastern Louisiana University

Abstract

The objective of this work is to develop a robot control architecture that detects and adapts to changing conditions without relying on a-priori information or human-in-the-loop calibration and setup. The focus of this work is the integration of memory based learning algorithms with coordinated observer and exploration modules through the use of a common graph based situational memory.

1. Introduction

This work is about the development of a control architecture that provides a basic level of autonomy for robotic systems that have the ability to interact with the environment through the use of sensors and actuators. The control architecture will provide an operating environment in which the various modules cooperatively work together so that the robot can achieve its goal without having to rely on preprogrammed databases, rules of thumb, or simulations. In doing so, the architecture will promote learning through exploration and analysis of the environment so that the robot can avoid undue physical repetition of movements. An important part of this architecture is a memory which is manifested as a directed graph of situations which typically appear in the environment of the application at hand. The development of this graph must be unsupervised and goal directed. In the following we summarize the architecture but the focus of this paper is the treatment of the memory part. The bird's eye view of our approach to the development of the memory is to formulate the observations of the environment as sequences which encode robot actions and sensor readings and then try to discover chunks (subsequences) within these sequences which seem to recur or otherwise seem to have a particular significance. These define "situations" which are used as nodes in the graph which represents memory in the system.

2. Background

Learning on board a robot must address challenges in dealing with the situations that a robot may encounter when operating in an unstructured or changing environment. Unlike artificial environments like that of a laboratory or a factory floor, unstructured/changing environments present problems in the form of rough changing terrain, unanticipated obstacles, fouling of sensors, and poor communication, among others. While simulations can help, the difficulties arising from simulating a complex environment can be prohibitive. It has been put forward [1] that "In order to really test ideas of intelligence it is important to build complete agents which operate in dynamic environments using real sensors." This leads back to the question of anticipating the various problems that the robot will encounter during its operations. If human operators are in close proximity to the robot, and the task is appropriately simple for the environment, learning and adaptation may not be necessary, but the more remote the area of operation, and the less structured and more dynamic the environment, the more appealing learning and adaptation becomes.

Learning and adaptation aboard a robot present certain difficulties that are not present in simulations. For instance, it may take a simulated robot 1000 attempts to learn how to correctly associate its sensor readings with its actuator controllers in order to avoid a dangerous obstacle. If each attempt takes 10 minutes, but can be simulated in .01 seconds, then the simulated robot can learn its avoidance behavior in about 10 seconds. On the other hand, if the same algorithms were used onboard a real robot, it would take almost a solid week to learn that task, excluding the time humans would take to measure results and to execute other test procedures, battery charging, maintenance, etc. To take the example one step further, if about $1/10^{th}$ of the trials resulted in damage to the robot during the simulation, then when learning with the real robot, about 100 repairs to the robot would be needed during the training period.

Clearly, learning in simulation is different than in a physical setting, but capturing physical realism in the simulation can be prohibitively difficult. Finally, suppose that a simulation of high enough fidelity sufficient for learning is produced, when the setting changes, or the type of robot is different, a new simulation must be developed. In a simulation, the process of learning, testing and evaluation is handled by the simulator, but for onboard robot learning these processes are facilitated by the robots control architecture.

Several researchers have studied this problem. According to Gat, the predominant view of the artificial intelligence (AI) community [2] before the introduction of Brooks' Subsumption Architecture [3] was that the control system for an autonomous mobile robot should be decomposed into three parts: a sensing system, a planning system, and an action system. This is commonly referred to as sense-plan-act (SPA). While SPA definitely has had its successes, the difficulty involved in world modeling and planning limited its success. Even with perfect world modeling, noise from sensors could easily cause the information that the sensors return to become out of sync with the world model. This in turn would cause errant planning and incorrect actions. Realizing these limitations, researchers began to look in different directions. Brooks was one of the first to come up with an alternative approach, coined the Subsumption Architecture.

The Subsumption Architecture is a method for controlling mobile robots that differs markedly from centrally controlled, top down approach, classical AI methods (SPA). It can be described as a hierarchical, asynchronous, distributed system whose components produce behaviors. The interactions of the behavioral components in the subsumption architecture produce the system functionality. In general, sensor inputs are available throughout the system and thus sensor input and actuator actions are closely coupled. The behavioral components are set up in a hierarchy of layers, in which higher layers can inhibit or subsume lower layers, thus the name Subsumption Architecture.

Both of these systems rely heavily on the developer to anticipate the problems that the robot will encounter during its operations in the environment. Neither one is focused on learning and adaptation, but instead relying on the functionality of the existing modules to arrive at the goal. Anytime Learning [4] is a technique which pairs an execution system with a simulator. The idea is that as the robot operates in its environment, it uses it sensors to enhance the fidelity of the simulation. When it encounters a situation that its onboard software does not know how to handle, it signals the simulator to find a solution, thus saving time and mechanical wear and tear. This system has had its successes [5] but because it uses a simulator, it requires considerable a priori knowledge. For unstructured environments, a method of learning that does not rely modeling or a priori knowledge is desired.

Q Learning [6] is a technique in which an agent randomly explores its environment and learns to reach a goal state through the use of delayed reward. The end result of the learning algorithm is a Q table that associates sensor readings with actions. This system does not require a-priori knowledge, but does require a considerable exploration/learning phase. Although it took in the neighborhood of one million test games, Tesauro [7] showed that Q Learning could be used to teach a computer to play backgammon at world championship levels. This system was effective for this application of game playing in which the environment was discrete and non-changing.

The goal of our approach is to combine the strengths of Anytime Learning and Q Learning into a robot control architecture that does not require a-priori knowledge and does not force the robot to physically test large numbers of candidate solutions during the learning process. To achieve this goal we propose to use a memory of "situations" entity which develops automatically during a goal directed exploration for the purpose of observing/assessing progress towards goals, essentially facilitating an adaptive critic within the system. Additionally, this memory can facilitate the generation of neural network controllers for various situations encountered. The situational memory thus provides a common data structure from which the learning, observation, exploration, and control modules will operate.

3. Approach

Our approach assumes an underlying architecture whose components share a common situational memory comprised of sensor/action pairs.

Sensor/action pairs collected from the immediate environment are used with the objective that a-priori databases and model based reasoning are minimized. The architecture is laid out in a similar manner to that of a three layer architecture [2] in that its components will run concurrently and work cooperatively to control the robot. Moment-to-moment control of the robot will be handled by the control and observation modules while longer term strategies for environmental knowledge acquisition and neural network controller generation will be handled by exploration and learning. The observer method will act as the system coordinator in that it will monitor progress toward goals, and trigger learning and exploration. The common thread between the architecture's modules will be the situational memory that the observer maintains. Figure 1 below shows a data flow diagram of the architecture.

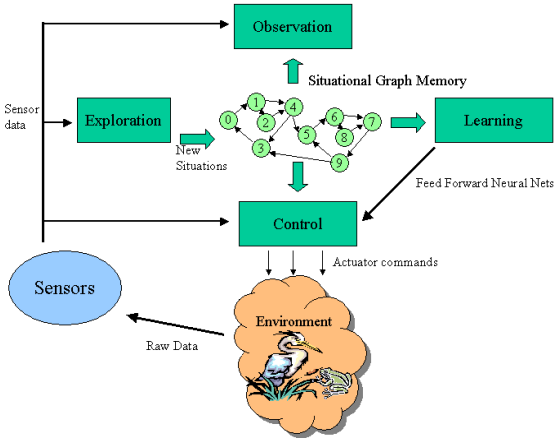

Figure 1. The interrelationships of the various components of the Memory-based Learning Architecture. The Situational Memory is a key component to the system.

As can be seen from the figure, the graph memory plays a central role in the architecture. It serves as the observer's memory; the exploration routine plays a major role in its creation, but also uses it to monitor and guide its progress; it is the primary input to the learning routines and it is used in control cases in which an alternate sequence of steps to the goal state is needed. Figure 2 below illustrates the interaction of the modules with one another.

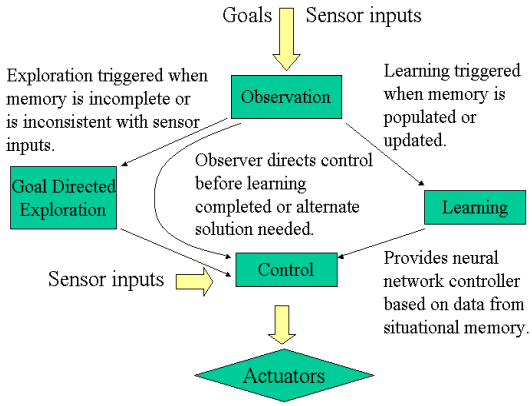

Figure 2. This figure shows how the various modules of the architecture interact with one another. As can be seen from the figure, the observer directs the other modules.

Once the exploration routine creates the memory, it will be monitored and updated by the observer process. The observer process will oversee exploration, learning and control. When the observer process determines that the memory is up to date with the current environment, it will trigger a learning function that will create a neural network controller that associates sensor readings with actions in order to realize the programmed goal. The controller will use the neural network to make decisions based on the current sensor inputs. It will be the observer's job to discern when actions taken by the robot are giving the anticipated results. In this event, the observer may opt to start the exploration/learning process again.

The exploration will not be completely random, it will be goal directed. It will be aware of the goal, but will not attempt to associate sensor readings with actions, at most it will continue to do an action as long as the robot thinks it is achieving its goals; when it is not realizing its goals, it will choose a random action, or one that it has not already taken when in its current situation. The exploration process will be monitored by its progress in populating a memory that can be best described as a group of temporarily related situations. When an observer function determines that the memory has enough data in it to create a draft controller, it will trigger the learning function to create a neural network that can associate sensor readings with actions in order to allow the robot to make correct decisions to reach its goal/goal state.

Initially the exploration process will attempt to populate the situational memory in a coarse but evenly distributed manner.

As stated earlier, once this is completed, the observation function will trigger learning, under the assumption that there exist enough data to create a draft control function that operates in a "broad brush" manner. That is, at this point, the controller generated will not be able to discern detailed nuances of the environment, but will work well enough to achieve the goal in a basic way. In this way, the initial exploration period will be kept to a minimum.

Since the memory is comprised of sensor action pairs arranged temporally it is possible to identify efficient paths to "goal" states. The paths to the goal states are the basis of a learning algorithm, described as "graph learning" [8], which creates a training set for use by a genetic algorithm (GA) that in turn creates a neural network controller. The end result is a feed forward neural network trained with the data from the most efficient routes to the goal state.

To illustrate the idea consider the following example. Imagine that a person wanted to walk to a bakery that was ahead of them and to the left. If they were not sure where the bakery was and did not know how to follow the smell of the baked goods wafting out of the bakery, they would have to learn to follow their nose. Suppose that on the first attempt to get to the bakery, they walked along straight until they came to an intersection, turned right, went to the next intersection, made another right, then another, until finally they made one more right and arrived at the bakery. Then sometime later another trip to the bakery was made, but this time when they arrived at the first intersection they turned left, and went directly to the bakery, avoiding the round-about route taken earlier. If that evening the person wanted to tell their friend how to get to the bakery, they would remember the two possible routes. In the interest of timeliness, they would likely tell their friend the direct route. Figure 3 below shows an illustration of this example.

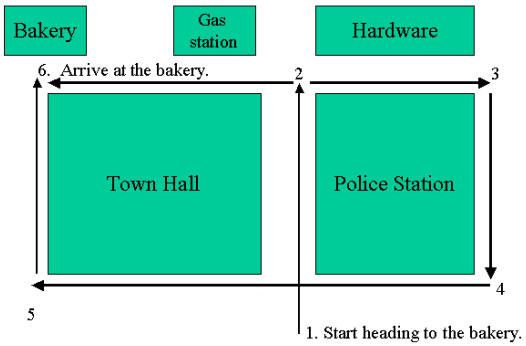

Figure 3. In this example, a person starting at position 1 heads to the bakery. On the first trip they take the route demarked by the path {1, 2, 3, 4, 5, 6} and on the second trip they take the path {1, 2, 6}. Clearly the 2nd path is much shorter, suggesting that an efficient strategy would be to learn the environmental queues at the points 1 and 2 so effective decisions could be made later on when presented with these or similar situations.

The two trips in this example would be analogous to the exploration period. Along the way, if notes on the perceived direction that the smell of the bakery originated were made, along with its intensity, a directed graph could be drawn. Many situations could be represented in the graph, such as "the smell is weak and about the same either direction", or "the smell is strong and to the left", and so on. Figure 4 below illustrates a graph memory that could have been formed from the above example. In the states in the diagram, L is the relative intensity sensed to the left, R is the relative intensity sensed to the right, and I is the intensity. All values are normalized to 1. State 6 is the goal state.

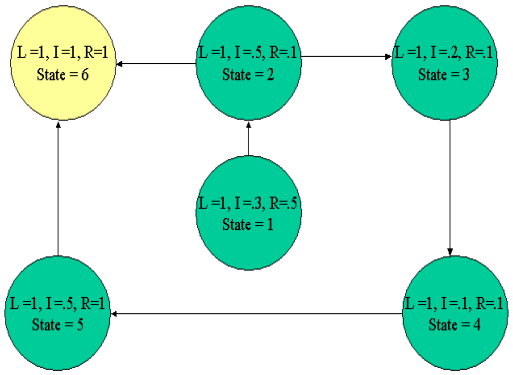

Figure 4. This figure illustrates the graph memory from the bakery example. The correct decisions given the sensor inputs are determined by which nodes lie on the shortest path to the goal state. So at state 2, the correct action is left

because this action results in the path {1, 2, 6} as opposed to {1, 2, 3, 4, 5, 6}.

Using these ideas, given associated sensor readings an action is deemed correct because it is part of a shortest path to the goal state, instead of using a supervisory program that checks the goal parameter at each step or makes judgments of correctness of an action based on prior knowledge or rules of thumb.

One of the main differences in this work and others such as Q learning is the method used to explore the environment and the way the collected information is stored. In this work the goal drives the exploration method and the temporal sequence of events that the robot encounters is converted to a directed graph. This directed graph serves as a dynamic environmental model, which is the central feature of the architecture. The graph contributes to the exploration by directing the robot to explore unseen situations instead of using a random search. In Q learning, temporal aspects of the exploration are typically not preserved, that is, there is no effort to learn from the order of sensor/action combinations experienced during the environmental search.

The above discussion is focused on the learning technique. While it is important, it is but one component of the architecture. The exploration and observation functions are of equal importance. For the learning system to be able to create a viable controller, it needs data that contains enough good examples to form a usable controller. Using the bakery example above to illustrate, if the second trip had never been made, the person would have never been able to learn to associate stronger smells to their left with turning left.

Previous testing [8] has shown that data collected from a semi-random goal driven exploration can be used to generate a graph memory from which a simulated robot can learn to follow a computer guided or operator directed robot effectively. In these tests, there was no observer function present, and so the exploration was more random than that proposed here. The tests were conducted without the benefit of any automated controls or architectural support, but nevertheless did show that goal directed exploration and learning using a graph based memory, can save exploration time, diminish the need for a-priori knowledge and promote learning without the need to physically execute all trial solutions. In essence, the process generates a controller by learning how to associate sensor values with what worked correctly during the exploration.

Graph development

As the robot explores its environment, it collects memories consisting of sensor action pairs. These memories are subsequently arranged into a directed graph whose nodes represent the situations that the robot experienced during its exploration. The nodes and connections of the graph preserve the temporal sequencing of the information collected during exploration, thus providing a structure that can be traversed in real time to facilitate control, or be used for learning, observation, and other functions.

The graph memory and the algorithms that it supports is based on the concept that the memory that the robot collects during its exploration of the environment can be broken down in a sequence of situations. The word “situation” in this context refers to some finite time period in which the robot is doing an activity while immersed in a certain set of environmental circumstances. The robot’s activity is defined as one or more of its actuator functions, while the environmental circumstances are defined as the robot’s sensor inputs and/or its internal state variables. In this work the recent memory is recorded during an exploratory period in which the robot tests the results of different actions in its environment. Many methods for exploration exist but are not within the scope of this publication.

The formal definitions of memory and situations are as follows. At each time slice, the robot records a sensor/action pair. A recent memory consists of a sequence of sensor action pairs, and a situation is considered to be a contiguous stream of sensor/action pairs. Thus:

SA: sensor/action pair. A representation of sensor readings and/or internal states, paired with the actuator values.

M: recent memory consisting of sensor/action pair sequences SA[0] …. SA[n-1]

M = SA[0] ….SA[n-1];

S: situation, consisting of a contiguous sequence of sensor/action pairs from memory M such that

S[i]=SA[k], SA[k+1],…, SA[k+(situation length - 1)]; k >= 0 and k < ((n –1) – situation length)

C: Set of all situations {S[0], S[1], …S[m]} found in M

The graph which encodes/implements memory in the above described architecture would represent situations as nodes. A path in that graph would represent a sequence of situations and thus memories (as sequences of situations). The task at hand is how to define the situations so that the graph is optimal in the sense that its size is kept reduced without loss of required granularity. To state the same problem in a different way, the task at hand is to extract subsequences (SA[k], SA[k+1],…, SA[n)]) which seem to recur and which seem to play an important role in the given robot environment and the goal. These subsequences then define the situations which correspond to nodes in the graph memory. Obviously, the graph arcs then define relative time sequencing between situations that have appeared consecutively in some memory.

In the past, we have tried a radial basis clustering algorithm used to cluster the memory. The RBF clusters are analogous to the situations. By regulating the inter-cluster distance and cluster size, the accuracy of the approximation M_s can be tuned. A large cluster distance results in less accuracy, while a cluster size that is two small results in situation lengths that are two short to be meaningful. An accuracy check is made by compressing the original recent memory M, with the new set of clusters C, referred to as the codebook. The technique described above is essentially a lossy vector compression. While it works well in settings in which the robot encounters situations that are all about the same time duration, it breaks down when the robot encounters situations of varying time duration. Normalization techniques can help here, but at the expense of loss of information. What is needed is a method in which the situations can be accurately represented regardless of their length in time.

In order to solve these problems, recently we have been exploring a different approach, one that is based on the idea of using compression methods similar to the ones used to compress text.

A sequence (SA[k], SA[k+1],…, SA[n)]) is essentially encoded in a sequence of symbols. This sequence of symbols can be treated just as text and then methods which are ordinarily used to compress text can be used to compress it. An example would be the gzip of the Unix operating environments. Although details about the function of gzip are abundantly available, the summary of it is that gzip finds duplicated strings in the input data. After the first occurrence of a string, further occurrences are replaced by a pointer to the previous string, in the form of a pair (distance, length) where distances and lengths are limited. Duplicated strings are found using a hash table. The gzip file format is specified in [9]; the Lempel-Ziv algorithm used in zip and PKZIP is specified in [10] ; the *"DEFLATE Compressed Data Format Specification"* [11] available at ftp://ds.internic.net/rfc/rfc1951.txt.

The methods used in text compression thus can be used to identify situations as subsequences (SA[k], SA[k+1],…, SA[n)]) which correspond to strings identified by the compression method as recurring with the rate of recurrence indicating its importance.

Once the memory has been broken down into a sequence of situations, the sequence is used to update the memory graph. In the example below, the memory M has 100 SA (Sensor/Action) pairs. The compression method found that the memory contained 5 unique clusters (the set C contains clusters S[0] … S[4]), meaning that the memory can viewed as a sequence of those clusters. For example, C (which essentially is a codebook) contained 5 clusters, each with a cluster size of 10, and the recent memory, M, had 100 SA pairs the resulting list would have 10 cluster indices in it. Below is an example list of cluster indices:

M is the memory viewed as sensor/action pairs.
M = {SA[0], SA[1], S[2]…SA[98], SA[99]}

C is the set of clusters (situations) that the compression algorithm finds.
C = {S[0], S[1], S[2], S[3], S[4]}

For this example, M_S is the memory viewed as the sequence of situations:

M_S = {S[0] , S[1], S[2], S[4], S[0], S[1], S[4], S[3], S[0]}

Relying on the single dimensional aspects of time, questions concerning the temporal arrangement of situations can be made. For example, the question can be asked, "If I am in situation A, and I take action 1, what is likely to happen?" These questions can be answered by tracing the sequence of clusters that are used to compress the recent memory. From the example above, it can be seen that the situation represented by cluster 1 follows the situation represented by cluster 0 twice in the memory.

Using the sequence of clusters produced by the compression, an adjacency matrix can be formed which indicates how nodes (situations) in a graph memory are connected temporally to each other. Table 1 shows the adjacency matrix for the above example. Figure 1 illustrates the corresponding directed graph.

Cluster number	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	1
2	0	0	0	0	1
3	1	0	0	0	0
4	0	0	0	1	0

Table 1. An adjacency matrix for the example above. The node numbers in the leftmost column show that cluster 1 follows cluster 0 in the example, as indicated by the 1 in column 1, row 0 of the table.

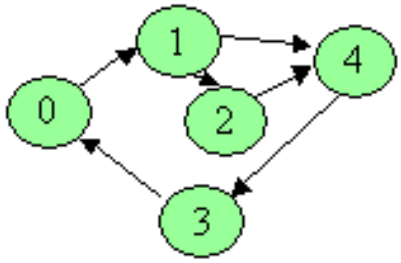

Figure 1. The graph from the example above and the adjacency matrix of Table 1

Once the adjacency matrix is formed the graph memory is ready to be used. In previous work, paths to goal states were used to train neural network controllers for specific behaviors.

4. Conclusion

In conclusion, the advantages of the proposed architecture include non-reliance on a-priori knowledge and avoidance of repetitive testing of candidate solutions by the robot/agent. The central data structure is the graph memory, which holds temporal and situational information for use by the exploration, observation, control, and learning functions. The use of compression methods was discussed here for facilitating the development of the graph memory.

Bibliography

1. Brooks, R.A., "The Role of Learning in Autonomous Robots", Proceedings of the Fourth Annual Workshop on Computational Learning Theory (COLT '91), Santa Cruz, CA, Morgan Kaufmann Publishers, August 1991, pp. 5–10.
2. Erann Gat, "On Three Layer Architectures," *Artificial Intelligence and Mobile Robots*, David Kortenkanp, R. Peter Bonnasso, Robin Murphy, eds., AAAI Press
3. Brooks, Rodney A. "A Robust Layered Control System for a Mobile Robot", *MIT AI Lab Memo 864*, September 1985
4. Grefenstette, J. J. and Schultz, A. C. , "An evolutionary approach to learning in robots," Machine Learning Workshop on Robot Learning, New Brunswick, NJ, 1994
5. Gary B. Parker, "Punctuated Anytime Learning for Hexapod Gait Generation", Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, EPFL, Lausanne, Switzerland, October 2002
6. Tom Mitchell, Machine Learning. McGraw-Hill, 1997, pp 367-386.
7. Gerald Tesauro, "Temporal Difference Learning and TD-Gammon", Communications of the ACM, Vol. 38, No. 3, March 1995
8. McDowell, P, "Biologically Inspired Learning System", Ph.D. dissertation., Louisiana State University, 2005. http://etd.lsu.edu/docs/available/etd-10192005-144729/

9. P. Deutsch, GZIP file format specification version 4.3, <ftp://ftp.isi.edu/in-notes/rfc1952.txt>, Internet RFC 1952 (May 1996).
10. Ziv J., Lempel A., ``A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.
11. ``DEFLATE Compressed Data Format Specification" available in ftp://ds.internic.net/rfc/rfc1951.txt