ADAPTIVE AUTOMATA FOR INDEPENDENT AUTONOMOUS NAVIGATION IN UNKNOWN ENVIRONMENT

JORGE RADY DE ALMEIDA JÚNIOR, JOÃO JOSÉ NETO, ANDRÉ RIYUITI HIRAKAWA
Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
e-mail: {jrady, jjneto, hirakawa}@pcs.usp.br

Abstract

Adaptive automata have been developed as a formal device intended to handle complex input languages, featuring context dependencies. As other adaptive formalisms, adaptive automata collect information from its inputs and may change its own behavior in response to that information. Therefore, such formalisms are able to dynamically acquire and represent knowledge. Independent navigation require environment scanning by the autonomous vehicle and a corresponding update of the information included in the model of the environment, allowing proper path planning and adequate motion control.

This paper briefly introduces adaptive automata and discusses its use to solve some aspects of the problem of environment mapping and path planning for independent autonomous navigation of vehicles in unknown environment.

Keywords: path planning, adaptive automata, autonomous vehicle

1. Introduction

The present paper addresses two interesting applications of adaptive automata: navigation control of autonomous vehicles and mapping unknown environments. As a motivation to the exposition of this subject, a very simple example is presented after a quick review of the main conceptual background on adaptive automata, automatic mapping of unknown environments and autonomous vehicle navigation.

2. Formal Background

The purpose of adaptive automata [1] is *not* to replace classical and widely accepted equivalent formal models, but to offer an equivalent mechanism to do the same job in some practical way, by allowing the development of good usable (and eventually automatically generated) implementations from (preferably intuitive) rigorous and correctly derived formalisms.

In order to achieve these goals, the proposed model embed a few redundancies that, although being theoretically unnecessary, do ease the practical use of the model by offering shortcuts to several otherwise cumbersome operations.

3. Structured Pushdown Automata

Our first attempt to build a formal model allowing efficient implementations has been made with the development of a formalism named *structured pushdown automaton*.

As finite-state automata allow the most efficient possible formulation for regular language acceptors, structured pushdown automata have been designed as a collection of mutually recursive finite-state-like submachines.

In its most primitive form, restricting structured pushdown automata exclude the use of its pushdown storage, reducing its behavior to that of mere finite-state automata.

For this restricted type-3 version of the automaton, there is absolutely no need of changes in the notation.

For type-2 languages, a stack is needed, and the set of submachines may be easily chosen for the structured pushdown automaton so that the whole model performs as a finite-state machine all the time, strictly except at self-embedded syntactical constructs on the input string, when recursion is used.

Algorithms are available for designing structured pushdown automata that employ a minimal number of submachines, and execute a single recursive submachine call per self-embedded construct found in the input string.

Structured pushdown automata have enough power to decide context-free languages, and allow easy automatic building of O(n) acceptors for deterministic context-free languages.

An extension has been made to structured pushdown automata, giving rise to the structured pushdown syntactical transducer, a very flexible model for syntactical translation of context-free languages.

One useful particular class of syntactical transducers, derivable from the structured pushdown transducer, may

be designed to implement parsers for context-free languages.

One of these parsers' input may be any sentence of the language and its output is a valid parse tree for that sentence, based on the context-free grammar from which the structured pushdown transducer has been generated.

They have been also used as a theoretical framework for the design of several compilers and in the front end of many other practical implementation of textual-input system- and application software.

4. Adaptive Automata

As the context-free limitation of the structured pushdown automata seemed to be too restrictive, we started searching for other models filling the needs of applications demanding context-dependent power, and holding the requirement for not loosing the simplicity and the efficiency of the previous model.

Furthermore, we searched for a model allowing full description of syntactical aspects of context-dependent languages, including all so-called static semantics – symbol tables, block structure, nested environments, type-checking etc. – and even dynamic syntax – defining simple, recursive and parametric macros, and other syntactical extension mechanisms.

All these requirements must be accomplished by the formal model we were looking for, by means of syntactical methods only, with no aid of semantic routines at all.

Our natural choice has been to extend structured pushdown automata into the model we called *adaptive automaton* due to its added self-modifying feature, and the resulting model did fulfill both requirements.

The only difference between structured pushdown automata and adaptive automata derives immediately from the conceptual difference between context-free and context-sensitive languages: in contest-sensitive languages, the occurrence of some particular construct in some specified environment may imply modifying the way related constructs are to be interpreted throughout of the input, and that may be obtained by modifying the behavior of the acceptor for that language.

A very elegant and strictly syntactic way to modify the interpretation of the input is to modify the syntax the automaton is able to accept, so fitting the needs of the new situation.

A straightforward method for modifying the syntax accepted by an automaton is to changes its topology according to the desired syntax modifications.

Any changes to an automaton may be accomplished through a set of elementary editing operations acting upon its set of transitions: adding a new transition to the set, selecting from the set of transitions a target transition that fit some given predicate, and eliminating a selected transition from the set.

Editing operations may be grouped, and the resulting groups are named *adaptive actions*.

Adaptive automata are built from a subjacent structured pushdown automaton simply by attaching adaptive actions to its transitions, so allowing the automaton to edit itself through the application of the corresponding adaptive actions whenever performing the particular transitions they are attached to.

Adaptive automata have been used thereafter as the starting point to a number of other correlate developments, both in their theoretical and practical issues, giving rise to variants as Adaptive Statecharts and Synchronized Adaptive Statecharts [2]

Adaptive automata have the power of a Turing Machine but, in contrast to this classical formalism, it is sufficiently attractive and effective for being used in an easy way by implementers as a specification language, because it behaves as a formal description metalanguage that allow the natural construction of practical and efficient implementations.

By observing the operation of adaptive automata, a single unusual feature may be indicated to be responsible by most of their practical power, namely its input-driven self-modifying capacity.

This adaptive paradigm is not exclusive to adaptive automata, but it may be applied as well to several other originally static formalisms, giving rise to a growing family of powerful dynamically modifiable formal models.

5. Notation and Interpretation

Adaptive automata are structured pushdown automata whose state-transition rules may be attached adaptive actions. The expression below,

$$(\gamma\, g, e, s\, \alpha),\ \boldsymbol{A} :\rightarrow (\gamma\, g', e', s'\, \alpha),\ \boldsymbol{B}$$

represents the general form of a rule in an adaptive automaton. The left-hand side of the expression refers to the current configuration of the adaptive automaton before the execution of the state transition, whereas its right-hand side encodes its configuration after the state transition.

The components of the 3-tuples encode the situation of the pushdown store, the state and the input data, respectively. After the 3-tuples, adaptive actions are optionally specified: the left-hand one represents modifications to be applied before the state transition, while the right-hand one specifies the changes to be imposed to the automaton after the transition.

Figure 1 represents the expression above in graphical notation.

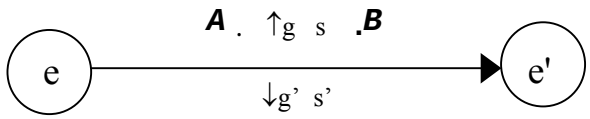

Figure 1 - Graphical version of the general form of a transition in an adaptive automaton

Adaptive actions are calls to parametric *adaptive functions*, which may be roughly regarded as collections of *elementary adaptive actions* to be applied to the transition set of the automaton.

Parameters are special variables, used in adaptive functions, to which actual values (*arguments*) are assigned whenever the adaptive function is called.

Elementary adaptive actions are the only editing operations supplied by the formalism. Three different operations are allowed: inspection, deletion and insertion of transitions. The expression,

$$\otimes\ [\ (\gamma\ g\ ,\ e\ ,\ s\ \alpha\ ),\ A: \rightarrow (\gamma\ g'\ ,\ e'\ ,\ s'\ \alpha\ )\ ,\ B\ ]$$

denotes any elementary adaptive actions by replacing the operator ⊗ by **?** for the inspection, + for the insertion and **-** for the deletion of a transition having the shape enclosed in brackets.

Although it is possible to create some graphical notation for adaptive actions, it would usually seem more difficult to understand than the algebraic notation. Therefore, no graphics will be used in this text for this purpose.

Note that both inspection and deletion elementary adaptive actions search the current set of transitions for any transition matching the given pattern. When such a transition is found, the *variables* used in place of any of the components of the elementary adaptive action are assigned the actual corresponding values in the matching transition. When used in an inspection or deletion elementary adaptive action, variables become either *undefined* (in the case no match was found) or *defined* (otherwise). Anyway, no further assignments to used variables are allowed.

An additional feature is also present in the model: the concept of *generator*. Generators are used to assign names to newly created states. They are also special variables, which are automatically assigned unique values at the start of the execution of an adaptive function, together with the assignment of argument values to the parameters. Again, once assigned, both generators and parameters are not allowed to change anymore.

6. Path Planning

In the robotics research field, one of the most difficult problems is related to the autonomous navigation of vehicles. Several approaches have been tried to solve this problem, but most of them are based on the proposal of some new algorithm to determine the optimal path to be described by a specific vehicle, moving in a well-known environment in order to accomplish some desired task.[3] [4].

In [5] it is presented an approach for solution of path planning using automaton concepts.

Usually, these works are addressed to indoor vehicle navigation, such as for industrial loading and transportation vehicles, hospital and office job helping machines.

However, there are few approaches that really perform independent autonomous navigation in unknown environment, reacting to generic external static and dynamic events. This is because, it is very difficult to model the real world and to adequately map the environment, due to the large amount of information needed, to a wrong or poor representation scheme for the environment facts and to the difficulties of representing and linking dynamic events to the map. The usual way to achieve environment mapping is by defining some actuation boundary, corresponding to physical limits of the region which can be identified and scanned by the available sensors, and whose collected information is stored into some adequately organized information area. After this first step, the desired path to be described is calculated that performs some specified job achieves some established goal. As a complement to this classical approach, some researchers have introduced an additional step for updating the database, by using information collected by the sensors during navigation.[6]

7. Usual Map Representation Strategy

The usual method to model and represent the real world for autonomous navigation make use of geometrical features of the environment. One of the most popular method consists of representing space by means of a bidimensional evenly-spaced grid, called occupancy grid. [4]

Each grid cell represents the occupancy probability of the corresponding area in the real world. The model consists of a graph whose nodes represent distinct regions of the real world and whose arcs indicate spatial relations. Since occupancy grids explicitly reproduce the geometrical structure of the environment, they are easy to learn and maintain. Additionally, the vehicle position and orientation may also be easily represented and recovered. However, this approach is very expensive, due to the large amount of memory and time required to build the occupancy grids. Indeed, in order to accurately model the real world, the resolution of the occupancy grid must be high, therefore the learning program module will have to manage a huge amount of data.

Recent research suggests a more qualitative representation of the world, based on the more compact storage, comprehending a few relevant features of the environment only, such as the topological approach. The major advantage of this approach is the compactness of

this model, for it optimizes the use of memory resources. In addition, since this representation is based on a graph structure, it allows the use of fast path planning activities.

8. Proposed Method

In this work, the adaptive automata is used as the conceptual representation device for the facts known to the autonomous vehicle.

In our application example, we make the following assumptions:

- An autonomous vehicle is placed in an environment having fixed obstacles (walls, shelves, files, etc.) and movable objects (chairs, tables). Other vehicles are allowed as well.
- All the objects recognized by the autonomous vehicle, that are fixed objects, movable objects or other autonomous vehicles, are treated as being obstacles. Otherwise, in the absence of an obstacle, it is configured a free space.
- The autonomous vehicle hasn't, a priori, knowledge of any geographical configuration of the environment it is going to travel through.
- Several goals may be established for the autonomous vehicle, such as:
  - *Topological scanning of the environment*: given its initial position in the environment, the autonomous vehicle must travel through the whole environment in order to collect all geometrical information needed to build a map representing the environment in a suitable level of detail. In this case, the autonomous vehicle must collect and register the location of all detectable elements in its environment.
  - *Searching for some specified target*: given its initial position, the autonomous vehicle must move through its environment, searching for some path leading to some desired destination (for example, a given final location in the environment, avoiding some specified fixed obstacle or movable object, or another specified autonomous vehicle). Then, the autonomous vehicle is expected to reach some established goal. There is no need to map the whole environment. Instead, the mapping activity may be interrupted just by the time the autonomous vehicle reaches the required target.

In both cases, the environment representation must initially include information on the location of both fixed and movable elements in the environment.

9. Application Example

Figure 2 depicts an environment including some fixed objects (walls) and one movable object. The environment is divided into square cells of fixed length. In the example, the environment is a 6×6 matrix where an autonomous vehicle can move vertically or horizontally only, one cell at a time.

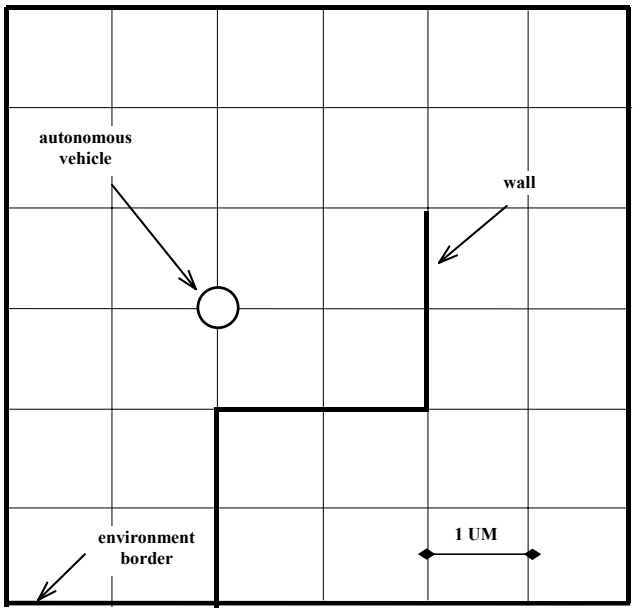

Figure 2 – Example of an Environment for Simulation

The logical representation of the above environment is as follows:

States = each cell intersection in the matrix

Movements = {North, South, East, West}

Returned values: {North, South, East, West} → {obstacle, free}

The following algorithm represents the actions performed by the autonomous vehicle in its task.

Supposing that the autonomous vehicle is in a certain position, it executes the following actions:

1) sensors the physical environment, detects X
2) consults the already built model of the environment, gets X'
3) If X = X' do nothing; otherwise X' ← X
4) Repeat 1 – 3 for the remaining directions
5) Executes some navigation strategy. For example, follow (x,y) such as DIFF (actual position, final position) → 0, where x, y is 2D position coordinate system and DIFF is a function that calculates the distance between the two positions
6) Executes the suggested movement
7) If the target was obtained, go to 1, otherwise ends processing

Our system consists of the following modules:

- The module Assembly of the Environment is constituted by a graphical editor which the positions

of fixed and movable obstacles are defined through. This environment has permits the initial positioning of autonomous vehicles.

- The module Generation of the Map/Route is responsible for the estimation of the map and generation of the route. To obtain the positioning of fixed and movable obstacles, the autonomous vehicle has a system of monitoring, having the capacity to detect objects in a distance of 1 UM. Through this resource, the autonomous vehicle can make the mapping of the environment, while moving over it.

The representation of the map is made through adaptive automata. Figure 3 shows the first step of the mapping execution – the autonomous vehicle does the monitoring for the right, finding a fixed obstacle (wall), generating the state P1 and a corresponding transition P, starting from initial state VA0. The same occurs when the monitoring is made below the initial state autonomous vehicleVA0, generating state P2.

In the case of an empty transition, it is suppressed, in the figure, the symbol ε. Transitions with arrows in both sides represent the possibility that the transition can occur in both directions.

When monitoring is made for the left, the autonomous vehicle doesn't find any kind of obstacle, generating a state VA1 and two transitions in blank (indicating allowed movement for the cell in question), one for the state VA0 to the state VA1 and another for the state VA1 to the state VA0. The same occurs if the monitoring is made for the top, generating the state VA2.

In the next step, the autonomous vehicle assumes, following some decision criteria, some of the possible positions for its movement, following the generation of the environment map, as it's showed in figure 4.

Preserving this mode to obtain the environment configuration, the autonomous vehicle can cover all environment, making its complete mapping, resulting in a representation in the form of an automaton, as it's represented in figure 5.

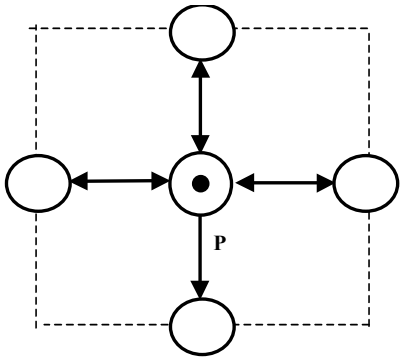

Figure 3 – Initial Mapping of the Environment (the vehicle is placed as in figure 2)

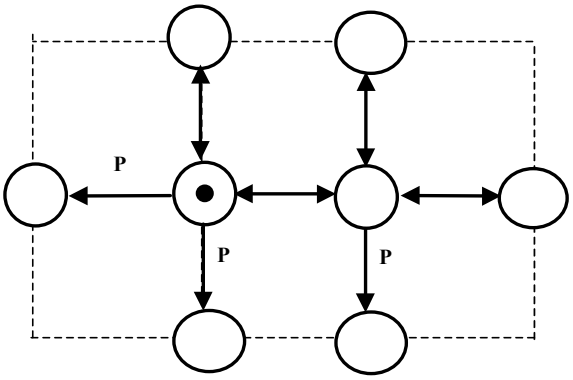

Figure 4 – Intermediate Mapping of the Environment

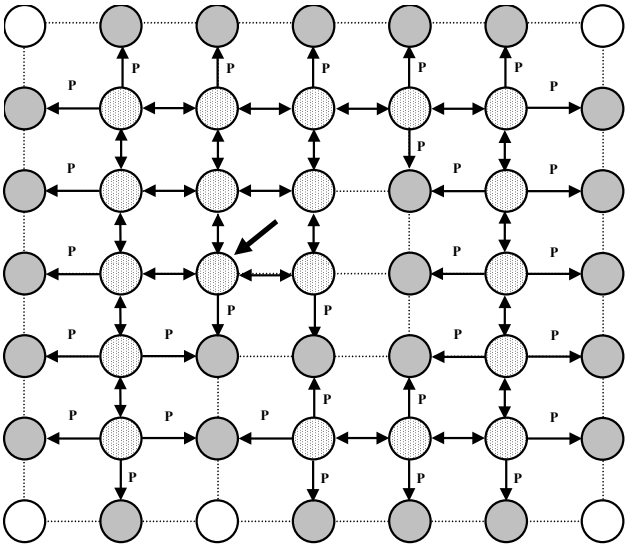

Figure 5 – Complete Mapping of the Environment

The module Route Presentation is responsible for the exhibition, in the screen, of movements executed by the autonomous vehicle, in its task of making the environment configuration.

In order to implement the suggested adaptive method, a software tool has been developed including such features into a previously implemented version [7].

In figure 6 is presented an environment already assembled, using this software tool. It can be noted, in this figure, some buttons for the assembly and execution of the environment, besides the menu bar and the configuration and movement simulation area for the autonomous vehicle.

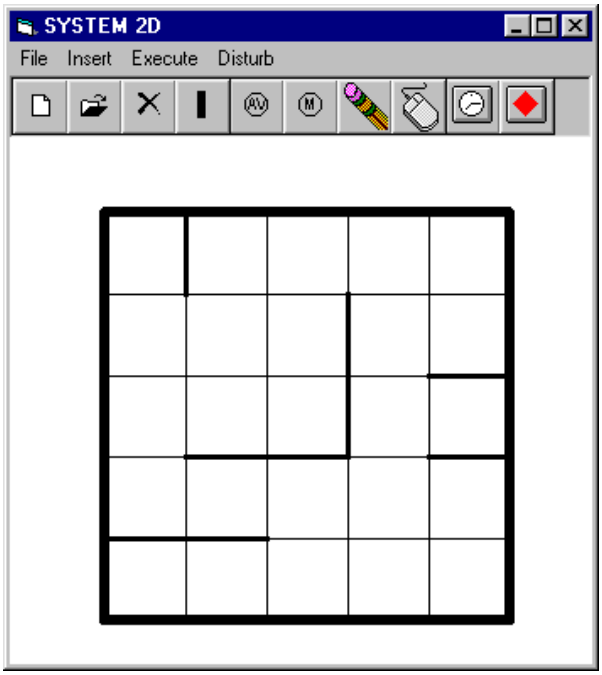

Figure 6 – Example of part of an Environment Assembled Using the Software Tool

10. Conclusions

In this paper, a method for recognizing an unknown environment has been proposed. This method explores some concepts from adaptive automata theory, and uses them for dynamically building a representation for the environment topology. This representation may then be used by any standard navigation strategy in order to estimate the best choice for the next move of the autonomous vehicle.

This method showed to be a convenient option for the representation of geographical aspects of the environment and is adequate for dynamic choice of navigation paths.

The task of acquiring information from the physical environment is seldom performed by conventional systems at navigation time since it is very complex and space consuming. Our approach suggests a simpler scheme due to the learning feature of the underlying model, which allows dynamically changing partial representation of the part of the physical environment already know by the scanning system.

Consequently, the autonomous vehicle will be free to start its navigation task immediately, without needing any external information on the topology, so navigation may proceed at any time despite the lack of information on the whole environment, since the vehicle has the ability to get all further information needed simply by scanning the vehicle's neighborhood and storing the collected information onto the adaptive automaton representing the environment.

Furthermore, the dynamics of the adaptive automata allow discarding old or unused historical information whenever the system goes out of memory. This feature allows the system to make better use of available memory resources.

The next in this experiment will be to increase the sensing capabilities of the vehicle in order to allow it to distinguish among several kinds of obstacles, and to detect other vehicles, when some kind of knowledge interchange may occur, especially on the shared environment's topology.

References

[1] José Neto, J. Adaptive automata for context-sensitive languages. ACM Sigplan Notices, v.29, n.9, p.115-24, Sept. 1994]

[2] José Neto, J.; Almeida Jr., J. R. and Santos, J. M., Synchronised Statecharts for Reactive Systems, Applied Modelling and Simulation, 1998, Honolulu, Hawaii, USA, 246-251

[3] Kavraki L. E., Kolountzakis M. N., and Latombe J.; Analysis of Probabilistic Roadmaps for Path Planning; IEEE Transaction on Robotics and Automation, V. 14, No. 1, pp. 166-171, February 1998.

[4] Arleo A., Millan J. R., Floreano D.; Efficient Learning of Variable-Resolution Cognitive Maps for Autonomous Indoor Navigation; IEEE Transaction on Robotics and Automation, V. 15, No. 6, pp. 990-1000, December 1999.

[5] Ben-Shahar O. And Rivlin E.; Pratical Pushing Planning for Rearrangement Tasks; IEEE Transaction on Robotics and Automation, V. 14, No. 4, pp. 549-565, August 1998.

[6] Pagac D., Nebot E. M. And Durrant-Whyte H.; An Evidential Approach to Map-Building for Autonomous Vehicles; IEEE Transaction on Robotics and Automation, V. 14, No. 4, pp. 623-629, August 1998.

[7] Almeida Jr., J. R. and José Neto, J. Using Adaptive Models for Systems Description, Applied Modelling and Simulation, September 1-3, 1999, Cairns, Australia