Learning Behaviour Based Control in Autonomous Mobile Robots

Humberto Martínez Barberá Antonio Gómez Skarmeta
Departamento de Informática, Inteligencia Artificial y Electrónica
Universidad de Murcia
30071 Murcia
e-mail: humberto@fcu.um.es, skarmeta@dif.um.es

Palabras clave: Robots Móviles, Control Difuso, Fusión de Comportamientos.
Keywords: Mobile Robots, Fuzzy Control, Behaviour Fusion.

RESUMEN. Este trabajo presenta cómo se han utilizado técnicas de aprendizaje, basadas en algoritmos genéticos, para resolver el problema del conflicto entre distintos comportamientos en el marco del control de un robot autónomo móvil. Para ello se describe el entorno de trabajo, basado en un lenguaje de programación propio (llamado BG) y en una arquitectura de agentes, que exhiben una serie de comportamientos, desarrollados utilizando lógica difusa. Se presentan resultados de un ejemplo sencillo de navegación en un entorno totalmente desconocido.

ABSTRACT. This paper presents how we use learning techniques, based on genetic algorithms, to solve the problem of the conflicts between different behaviours in the context of an autonomous mobile robot. We also describe the working environment, based on a custom programming language (named BG) and an agent architecture, where these agents exhibit a series of behaviours developed using fuzzy logic. Last, some results related to a simple navigational task in an unknown environment are presented.

1. - INTRODUCTION.
The operation of a mobile autonomous robot in an unstructured environment, as it occurs in the real world, needs to take into account many details. Mainly, the controller has to be able to operate under conditions of imprecision and uncertainty. For instance, the a priori knowledge of the environment, in general, is incomplete, uncertain, and approximate. Typically, the perceptual information acquired is also incomplete and affected by the noise. Moreover, the execution of a control command is not completely reliable due to the complexity and unpredictability of the real world dynamics. Mobile robots are increasingly required to navigate and perform purposeful autonomous tasks in more complex domains, in real world like environments. These requirements demand reactive capacity in their navigation systems. Over the last decade much of the work on reactive navigation has been inspired by the layered control system of the subsumption architecture (Brooks 1986), which tightly couples sensing and action. The emergent behavior of the robot is the result of the cooperation of independent reactive modules, each one specialised in a particular basic behaviour.

Behaviour-based control shows potentialities for robot navigation environment since it does not need building an exact world model and complex reasoning process. However much effort should be made to solve aspects like formulation of behaviours and the efficient coordination of conflicts and competition among multiple behaviors. In order to overcome these deficiencies,

* This work has been founded by CICYT TIC97-1343-C002-02 and CICYT-FEDER 1FD97-0255-C003-01 projects.
some fuzzy-logic based behaviour control schemes have been proposed (Saffiotti 1997). Although, most of the behaviour-based approaches focus on static behaviour configurations, introducing the fuzzy rules by means of an expert.

In recent years, fuzzy modelling, as a complement to the conventional modelling techniques, has become an active research topic and found successful applications in many areas. However, most present fuzzy models have been built based only on operator's experience and knowledge, but when a process is complex there may not be an expert (Wang and Langari 1996). In this kind of situation the use of learning techniques is of fundamental importance. The problem can be stated as follows. Given a set of data for which we presume some functional dependency, the question arises whether there is a suitable methodology to derive (fuzzy) rules from these data that characterise the unknown function as precise as possible. Recently, several approaches have been proposed for automatically generating fuzzy if-then rules from numerical data without domain experts.

To combine reactivity, fuzzy modelling, behaviour based control and learning techniques we have defined and developed a simulation environment based on a programming language, which uses a multi-agent paradigm over a blackboard model (Section 2). Using this language we have defined a sample series of agents to control a robot in a navigational task (Section 3). As a key point in behaviour coordination we use learning techniques (Section 4). Finally we have tested this procedure simulating the robot in some environments (Section 5) and obtained some conclusions (Section 6).

2. - THE BG PROGRAMMING LANGUAGE.

Several robot control languages has been proposed in the literature. Each one of them has its own pros and cons. As our main goal was to tackle with the behaviour blending problem and the learning of the rules that drive the blending process, we chose COLBERT (Konolige 1997) as the working base. This makes extensive use of fuzzy rule bases both for the behaviours and the blender. As we wanted to try some different functionalities, we decided to define and implemented a new high level programming language, named BG (Gómez et alt. 1999b), which is based on the C language syntax and semantics and most of the COLBERT language features. It is based on the multi-agent paradigm, where each agent possesses a series of behaviours. Its intended use is robotics and control applications, mainly autonomous mobile robots. The examples in this paper yield in the later field, and it is supposed that a robot will execute them. In fact that robot is our custom one described in the previous section. The main goal of the language is to aid in the development cycle of robotics applications and in the behaviour modelling (as it is a robotics oriented high level language with built in fuzzy logic operations), as well as to expedite it (by using automated tuning systems). BG, in its current version, incorporates many traditional elements and control structures of classical imperative languages:

- Local and global variables.
- Expressions and assignments.
- Arithmetical, logical and relational operators.
- Conditionals.
- C-like functions.

Basically, BG supports two data types: real numbers and fuzzy sets with standard shapes (triangle, trapezoid, bell, and sigmoid). In addition to that, BG presents advanced non standard control structures like:

- Deterministic finite state machines.
- Fuzzy rule bases.
- Grid map based calculations.
- Agents definition.
• Behaviours definition and blending. These elements are described in the following subsections.

2.1. Finite state machines
The use of finite state machines allows the programmer to define tasks that need to be run sequentially, and the conditions to change from one task to other. The BG language, as of its current implementation does not have instructions for loops (while, for, or repeat-until), but these can be modelled using a finite state machine. The main reason for this lack of loops is to have under control the duration of each control loop (so that an error can not lock it). In a future, loops will be incorporated as in COLBERT (Konolige 1997), which, in fact, are finite state machines.

2.2. Fuzzy rule bases.
One of the strengths of the BG language is the possibility of using fuzzy rule bases. These are intended for both control and behaviour blending (the later will be discussed below). Two operators have been defined: a t-norm (the min) and a t-conorm (the max). The defuzzification is carried out by means of the center of gravity method. Typically the rules will consist of an antecedent with some input variables and a consequent with some output variables (fuzzy MIMO type I rules). There is a special kind of fuzzy rule, called background rule, that is useful to cover undefined input space. The way they work is by applying a fixed alpha-cut to the consequent part of the rule.

2.3. Grid map based calculations.
The BG language provides a basic structure (grid) and methods to manage bi-dimensional spatial information. When this grid is initialised, the user provides two parameters: the number grid cells (a N x M matrix) and the length in metres of the cell sides. The left-bottom cell is used as the origin of co-ordinates. Each cell has two values, ranging from 0 to 1, corresponding to the degree of certainty that the cell is empty and occupied (Oriolo et alt. 1998). As the robot moves, the following data is processed:
• The current absolute location \((x, y)\) of the robot.
• The current absolute location \((x, y)\) of the goal place.
• A relative sonar reading \((\theta, \text{len})\) from the given absolute location \((x, y)\).
While the robot moves, the previous information is aggregated to the cell emptiness and occupancy values, and an A* algorithm searches for the shortest path to the goal from the current robot position. Then an absolute heading angle, which can be used for planning, is returned.

2.4. Agents and behaviours.
The organisational units in the BG language are based in the agent notion. We arrange these agents using a blackboard based architecture (Hayes-Roth 1985) as the main paradigm for communication and control. Under this model, a series of agents (also known as knowledge bases) share the available information about the system and the environment. Each agent can read from the blackboard the information that is relevant for it, perform its own processing, and then write each possible result onto the blackboard.
A key idea is that each agent is decomposed in very simple behaviours. These access the blackboard by way of some input and output variables. As two different behaviours may modify the same variable, a fusion method (called behaviour blending) is provided (Saffiotti 1997). The advantage of this method is related to scalability and learning: while in a subsumption architecture (Brooks 1986) the management of agents depends on the behaviours they
implement (Gómez et alt. 1999a), with the proposed architecture the management resides on the fusion method. A fuzzy rule base carries on this fusion, enabling/disabling the output of the different behaviours, both totally and partially. This way, the system can be trained with only some simple behaviours, and then new behaviours can be added without modifying the previous ones: only the fusion rule base needs to be re-trained.

When an agent is specified, the user may supply three kinds of blocks that are placed inside the agent's execution loop:

- A common block that is executed beforehand any behaviour.
- A series of behaviours that are executed concurrently.
- A blender block that specifies how the outputs of the behaviours will be fused.

All these blocks must satisfy a constraint: the execution time of each one must be bounded, so there are not infinite loops. The language itself assures that this constraint holds for all the blocks, and it is a necessary requirement for getting basic real time support, although the language doesn't support fixed time scheduling. It is the user responsibility to make the appropriate tests to obtain certain control cycle duration.

For the blending process, each behaviour defines what global output variables are to be fused. Thus they only modify local copies of those variables during the behaviour execution. The blender applies inference over fuzzy rules, defined using some input global variables as the antecedent and some behaviour names as the consequent. Lasts, the following formula (eq. I) is applied to each blended output variable:

\[
\alpha_j = \frac{p_{rj} \cdot b_{ej}}{\sum_j \alpha_j}
\]

\[
o_{vi} = \frac{\sum_j \alpha_j \cdot l_{ovij}}{\sum_j \alpha_j}
\]

where:
- \(o_{vi}\) is the i-th global output variable.
- \(l_{ovij}\) is the local copy of the i-th output variable in the j-th behaviour.
- \(p_{rj}\) is the priority of the j-th behaviour.
- \(b_{ej}\) is the result of the fuzzy inference for the j-th behaviour.

This way the fusion depends both on a fixed priority and a variable activation degree calculated using fuzzy inference. Each particular application will dictate which combination of priority and inference is the best (to use only priorities, to use only inference, or to combine both). Moreover, this novel approach for behaviour blending generalises the concept of hierarchical fuzzy rules, because, in fact, this hierarchy may combine fuzzy rule base with other techniques (from both soft computing and classical imperative programming). We will show, later on, how learning techniques can be used in conjunction with this blending mechanism.

3. – A SAMPLE AGENT ARCHITECTURE.

Once defined a programming language and its inherent programming model, we chose a generic robotics platform and a simple navigational task. The platform is a differentially steered and kinematically holonomic robot with some ultrasonic sonar sensors. The task is to navigate from a given to a goal location in a completely unknown environment, but similar to a real indoor floor (with different rooms and corridors). In addition, the robot should neither collide nor get frozen (or inside an infinite loop). To accomplish this goal the robot must be able to perform certain tasks: navigation with obstacle avoidance, goal seeking, map building and localisation, some form of high level planning, etc. Using the BG language and its implicit architecture we have made a decomposition of a real system, extracting the most important elements (Armstrong and Crane 1998). We have identified the following agents (Figure 1) that will be
described in the next subsections:

- **Reactive Control.** This is the agent that is in charge of the reactive navigation.
- **Planning.** This is the agent that is in charge of high level planning and global goal completion supervision.
- **Localisation.** This is the agent that is in charge of building a more or less precise model of the environment, and then locating the robot on it.
- **Vehicle Control.** This is the agent that is in charge of low level elements control and signal filtering: sensors and actuators.

![Figure 1. Agents architecture.]

### 3.1. Reactive Control.

This is the most important agent. It implements the basic reactive behaviours (Gómez et al. 1999) that the system will exhibit, using fuzzy rule bases. These rules, that have been obtained both from previous experience and the literature, affect both the desired robot velocity \( v \) and steering angle \( \theta \) based on the current sonar readings \( \text{sonar} \).

- **Obstacle-avoider:** drives the robot to the opposite direction to the nearest obstacle.
- **Align-right:** holds the robot at a given distance from the wall at its right.
- **Align-left:** holds the robot at a given distance from the wall at its left.
- **Move-to-goal:** drives the robot in the direction of the current goal.
- **Noise:** drives the robot in a random direction to escape from a hole.

The output of these agents is fused by the blender block, which is defined using a fuzzy rule base as well. The key point of how these rules are defined is addressed in the next section.

### 3.2. Vehicle control.

This agent is composed by two kinds of behaviours: sensor related and actuator related ones. The first type is in charge of processing and filtering (Delgado et al. 1998) the information provided by the sensors, and the second type is in charge of sending the right signals to the actuators to perform the given control commands. Both types of behaviours are composed of simple fuzzy controllers. In our system we have the following behaviours:

- Range sensors acquisition and filtering.
- Collision detection.
- Robot control vector \((v, \theta)\) execution.

### 3.3. Planning.

Due to the simplicity of the tasks the robot is going to perform, the planning agent is quite simple. At present, it is a finite state machine, whose states are: looking for goal location, and halted. These states have associated goal co-ordinates that will be received by the reactive
control agent, which will activate the corresponding behaviours to accomplish that.

3.4. Localisation.
The localisation agent is simply in charge of making a map of the environment and keeping track of the home location. To implement the map the agent uses the grid structure provided by the BG language. On the map, the robot is located using odometry (at this time we assume it is ideal, without errors).

4. – LEARNING BEHAVIOUR FUSION.
As stated previously, a key issue of behaviour-based control is how to efficiently coordinate conflicts and competition among different types of behaviour to achieve a good performance. Instead of just inhibiting and suppressing strategies according to assigned priorities, a fusion method (called behaviour blending) is used (Dorigo and Colombetti 1998)(Saffiotti 1997). A fuzzy rule base carries on this fusion, enabling/disabling the output of the different behaviours, both totally and partially (Gómez et alt. 1999a) (Arrúe et alt. 1997). The use of learning techniques (Bonarini and Basso 1994)(Dorigo and Colombetti 1998) in the fusion rule base, can result in robot navigation performance improvement. This way, the system can be trained with only some simple behaviours, and then new behaviours can be added without modifying the previous ones: only the fusion rule base needs to be re-trained.

Genetic Algorithms (Golberg 1989) are adaptive procedures of search and optimisation that find solutions to problems, inspired by the mechanisms of natural evolution. They imitate, on an abstract level, biological principles such as a population based approach, the inheritance of information, the variation of information via crossover/mutation, and the selection of individuals based on fitness.

GAs start with an initial set (population) of alternative solutions (individuals) for the given problem, which are evaluated in terms of solution quality (fitness). Then, the operators of selection, replication and variation are applied to obtain new individuals (offspring) that constitute a new population. The interplay of selection, replication and variation of the fitness leads to solutions of increasing quality over the course of many iterations (generations). When finally a termination criterion is met, such as a maximum number of generations, the search process is terminated and the final solution is shown as output.

As our main concern is the learning of the fusion rule base, we apply GAs for the task (Gómez, et alt. 1999c). The user selects an agent that will serve as the base of the learning procedure. The key idea is to start with a predefined set of fuzzy rules for behaviour blending, and then apply the learning method to them. This set can even be the empty set. In either case, the individuals of the GA are fuzzy rule bases (Pittsburgh coding), and the result of the GA is the proposed fusion fuzzy rule base.

4.1. Initial population and coding.
As stated previously, the learning procedure can be started in two ways: using a predefined set of rules (rule tuning) or using the empty set (rule discovery). If the algorithm is run in the rule discovery mode the initial population is set randomly. Otherwise, the initial population will consist of the user-defined rules and random modifications of them. The later corresponds to a typical set up, because it narrows the search space, and thus requiring shorter learning times.

The representation we use for the individuals imposes some constraints to the rule bases: the fuzzy sets must be trapezoidal and the maximum number of rules must be set up beforehand. These are not really hard constraints because the conversion from triangular to trapezoidal is straightforward, and we can specify a big enough maximum number of rules. We extract the fuzzy rule base from the blender module of a given fixed agent. The user also specifies the different input variables that will be used in the rules. The output variables are the different
behaviours of that agent. Each rule is coded by concatenating two parts. A rule part, which consists of the concatenation of the real numbers that define the fuzzy sets of the input and output variables. And a control part, which consist of the concatenation of two boolean numbers that represent if the rule is active or not, and if the rule is a background rule or not. Then, the representation of the individual is the concatenation of the data for all the different rules (both active or not).

4.2. Genetic operators.

The user must set up a series of parameters (Golberg 1989) that control how the algorithm proceeds. Basically, which kind of genetic operator is going to be used, the probabilities of both mutation ($P_m$) and crossover ($P_c$), the maximum number of generations ($Gens$) and the population size ($Popul$).

We have implemented several (Gómez and Jiménez 1997) genetic operators for the rule part:
- Classical mutation
- Non uniform mutation
- Classical crossover
- Arithmetical crossover
- MaxMin crossover

and for the control part of each rule:
- Classical mutation
- Classical crossover

Each time the algorithm needs to apply crossover or mutation, it randomly selects one of the previous methods. Additionally the user may specify different probabilities for the different operators. In our examples we have kept all the operators with the same probability.

4.3. Objective function.

While the other topics are quite standard, the definition of the objective function is not a simple matter. This function measures the goodness of each individual. As the individuals, in fact, affect the performance of the robot, the objective function must take into account how the robot executes a predefined task. For this reason the user must define a simulation framework with, at least, the following properties:
- Goal: where the robot starts, where the robot ends, and what the robot has to do.
- Map: a floor plant with walls, doors, corridors and obstacles.
- Iters: number of simulation runs per individual.

To avoid infinite loops in the simulation some additional parameters are needed:
- Timeout: maximum allowable time for each run, measured in epochs.
- Maxcolls: maximum allowable number of collisions.

When evaluating the fitness function for each individual, it is used as the fusion fuzzy rule base for a given agent, and a simulation run is executed. At the end, each simulation run returns four values:
- Gotcha: -1.0 if the goal is accomplished, 1.0 otherwise.
- Epochs: number of simulations steps performed.
- Colls: number of robot-wall collisions.
- Way: percentage of distance left from the starting to the goal location.

Using the different parameters and values defined above, the fitness function is calculated using the following formulae (eq. II, III, IV, V):

\[
Total = \sqrt{(\text{start}_x - \text{goal}_x)^2 + (\text{start}_y - \text{goal}_y)^2}
\]
5. – SIMULATION RESULTS.

We have developed, using the Java language, an integrated development environment for the BG language. This way we have a multi-platform application named BGen (available at http://ants.dif.um.es/~humberto/asy). This implements an interpreter for the BG language, a simulator for our custom robot, some data visualisation and recording tools, and the GA based learning tool. The BGen output includes a representation of the grid map (Figure 3) that the robot builds, and the degrees of activation of the different behaviours as a result of the fusion mechanism (Figure 2). Both are updated in each simulation step.

Using this software we have performed some test to verify the validity of our theses. We will focus on the results of one of such tests. Basically, the goal is to navigate on a simple floor plant (Figure 5), starting from a given location (the left of the arena) and finishing in the opposite side (small circle on the right of the arena). The robot has no idea about the floor plant but the coordinates of the finishing point. The agents, behaviours, and control rules for the robot controller are the ones described and showed in the previous sections. We have fed the learning tool with the Reactive Control agent and the following GA parameters (Table 1):

![Figure 2. Degree of activation of the different behaviours.](image)

![Figure 3. Aggregation of empty and occupied cells.](image)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Pc</th>
<th>Pm</th>
<th>Geners</th>
<th>Popul</th>
<th>Iters</th>
<th>Timeout</th>
<th>Maxcolls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.9</td>
<td>0.25</td>
<td>1033</td>
<td>8</td>
<td>2</td>
<td>500</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1. Learning parameters.

Then we have obtained the following results. Using the user defined fusion rules, the initial robot controller neither achieves the goal nor collides. This is a good starting point for learning because the controller is pretty fair and intuitive (in fact designed on the fly just a while before the tests), and the learning time should not be very high. As shown below (Figure 4), the fitness

\[
\text{ToGoal} = \sqrt{(\text{robot}_t - \text{goal}_t)^2 + (\text{robot}_t - \text{goal}_t)^2}
\]

\[
\text{Way} = \frac{\text{Total} \times 100}{\text{ToGoal}}
\]

\[
\text{Fitness} = \sum_{i} \left[(\text{Timeout}_i - \text{Epochs}_i) - \text{Colls} \times 25 - \text{Gotcha} \times 100 - \text{Way} \times 5 \right]
\]
function starts with negative values (goal not achieved) and ends up with high positive values (goal achieved in a small number of epochs). The best individual does drive into the goal despite the fact of the surrounding walls, and minimising the total distance travelled. To test the generalisation capability of the procedure, after the learning method we use the final fuzzy rule base in a different and more complex floor plant (Figure 6). As can be seen, the robot achieves its task as well, and the path is fairly smooth and straight, even when passing the door.

6. – CONCLUSIONS.
In this paper a hybrid fuzzy-genetic agent based system for autonomous robots in uncertain environments is proposed. These agents are composed of two components: first a collection of fuzzy controllers that implement the different behaviour based schemes of the robot, and second, a fuzzy meta-controller that combines the different behaviours and where the weights
associated to each behaviour are learned using a genetic algorithm. In this way this meta-
controller coordinates conflicts and competition among multiple behaviours efficiently. We have 
shown how this learning is carried on with the agent architecture we have defined. We have 
developed some tools to assist on the overall controller deployment process. To test the 
performance of the whole system we have carried out several tests with the simulator. The 
results are quite satisfactory. Future work includes the test of more complex tasks and 
environment conditions, and the study of the correlation between the simulator and the real 
robot performance.

REFERENCES.