Handling uncertainty in control of autonomous robots*

Alessandro Saffiotti

Applied Autonomous Sensor Systems
Department of Technology and Science
Örebro University, S-70182 Örebro, Sweden
URL: http://www.aass.oru.se
E-mail: alessandro.saffiotti@aass.oru.se

Abstract. Autonomous robots need the ability to move purposefully and without human intervention in real-world environments that have not been specifically engineered for them. These environments are characterized by the pervasive presence of uncertainty: the need to cope with this uncertainty constitutes a major challenge for autonomous robots. In this note, we discuss this challenge, and present some specific solutions based on our experience on the use of fuzzy logic in mobile robots. We focus on three issues: how to realize robust motion control; how to flexibly execute navigation plans; and how to approximately estimate the robot's location.

1 I had a dream

It is Monday morning, and it is raining. I enter my office. Edi, my purple personal robot, promptly realizes my presence, and happily rolls out to get me some coffee. Down the corridor, it slips over the water left by somebody's shoes, and has to correct its trajectory by sensing the walls. The cafeteria's main door is closed, so Edi crosses the library and enters the cafeteria by its side door. Having obtained a cup of coffee from Gianni, the barman, it comes back by the same way, moving smoothly in order not to spill the coffee. Three people are smoking in the corridor, and Edi has to maneuver around them, and around a cart hiding behind them. I have just finished reading my mail when Edi comes in with my coffee on its head. It is still hot.

What makes this story a dream? After all, the task performed by Edi may not seem very complex to an outsider; and we know that industrial robots can perform seemingly elaborate tasks with high reliability. Yet, this story is probably close to the best that today's most advanced research robots can do. Having a commercially produced, inexpensive robot to reliably perform tasks of this type in our offices and houses is still a dream. Why?

* In: M. Wooldridge and M. Veloso, eds, *Artificial Intelligence Today* (Springer-Verlag, DE, 1999) pp. 381–408.

A crucial observation to understand the difficulties involved is that control programs are typically based on a *model* of the controlled system. Control engineers use a mathematical description of the plant to design their regulators; and AI planning programs incorporate a symbolic description of the target system, of its dynamics, and of the effects of our actions on it. Now, in the case of robot operation the controlled system is composed of the robot *and* its workspace, and we need to account for both these elements in our models — see Figure 1.

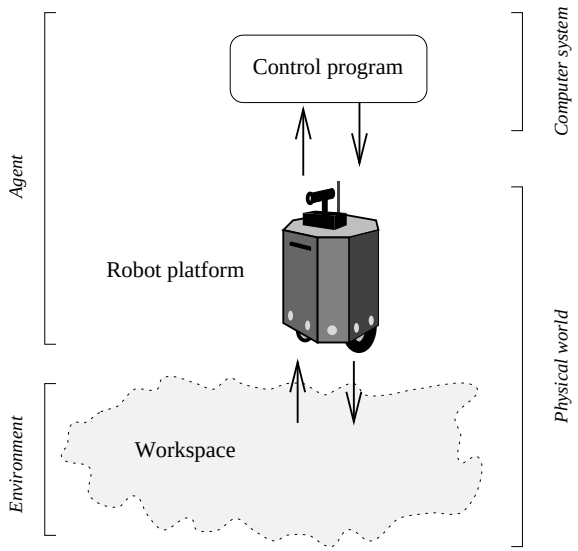

Fig. 1. The two facets of the robotic problem. The external observer sees a physical agent (the robot's body and its software) operating in an environment. The designer sees a control program controlling a physical system (the robot's body and its workspace).

A reasonably accurate model of the robot on its own can usually be obtained. For industrial robots, a complete model of the workspace is also usually available. The workspace is highly engineered and completely determined at design time: we know where the workpiece will be, and we know that there will not be a child standing in front of it (if these assumptions are wrong, the robot fails to perform its task). Unfortunately, the same amount of knowledge cannot be obtained in general for the type of real-world environments where we would like our autonomous robots to operate.

There are two sources of problems in getting a reliable model of the "robot + environment" system. The first one is the environment. Our prior knowledge of unstructured environments is necessarily incomplete, uncertain, and approximate: maps typically omit some details and temporary features, spatial relations

between objects may have changed since the map was built, and the metric information may be imprecise and inaccurate. Moreover, the environment dynamics is typically complex and unpredictable: objects can move, other agents can modify the environment, and relatively stable features may change slowly with time (e.g., seasonal variations).

The second problem is that the interaction between the robot and the environment may be extraordinarily difficult to model. The results of the robot's actions are influenced by a number of environmental conditions which are hard to be accounted for: wheels may slip, and a gripper may lose its grasp on an object. And the relation between the perceptual information and the reality is elusive: noise in sensor measurements introduces uncertainty; the limited sensor range, combined with the effect of environmental features (e.g., occlusion) and of adverse observation conditions (e.g., lighting), leads to imprecise data; and errors in the measurement interpretation process may lead to incorrect beliefs.

Several of these sources of difficulties appear in the above dream. Edi must use a map to plan its route to the cafeteria, but this map turns out to be inaccurate (the door is closed) and incomplete (the people who are smoking are not in it). The effect of Edi turning its wheels is modified as a result of the wet floor. And the cart only comes into the sensor's view at the last moment. In general, builders of autonomous robots must face a great challenge: to design robust control programs that reliably perform complex tasks in spite of the large amount of uncertainty[1] inherent to real-world environments. As Lumelsky and Brooks already remarked in 1989 ([24], p. 714.)

> The next big leap of industrial and household automation depends on our ability to overcome the very expensive and often unrealistic requirement of structure and order characteristic of today's "hard" (industrial) automation.

In this note, we present some ways in which we can make it possible to take this leap. In the next section, we analyze the sources of uncertainty in the autonomous navigation task, and discuss how the robotics community has tried to deal with them. In the following sections, we present some possible solutions to the problem of uncertainty in robot navigation, based on the analysis of a test case: the use of fuzzy logic in the SRI International mobile robot Flakey (see [39] for more comprehensive reports on this work). We concentrate on three specific important problems: how to define robust behavior-producing modules; how to flexibly execute complex tasks; and how to reliably establish the robot's position with respect to a map of the environment. We conclude this note by a few remarks on the choice of a "best" way to deal with uncertainty in robotics.

[1] Throughout this note, we use the term "uncertainty" in its most general flavor; we shall use more specific terms like "imprecision," "vagueness" and "unreliability" when we want to focus on a specific facet of uncertainty.

2 The challenge of uncertainty

Consider again Figure 1, and recall our claim that many of the difficulties in modeling the "robot + environment" system come from the inherent uncertainty of real-world environments. We now inspect this claim in deeper detail, and sketch the most common solutions adopted to circumvent this problem.

Suppose for concreteness that we are modeling our system by using the tools from the theory of dynamical systems (similar considerations could be made assuming a logic-based representation in the mainstream of AI). For example, we may model the system by the following pair of equations:

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) \\ y_t &= g(x_t)\,, \end{aligned} \tag{1}$$

where x_t denotes the state of the system "robot + environment" at time t; u_t the control signals sent to the robot at t; and y_t the signals returned by the robot's sensors at t. The f function, usually called the state transition function, accounts for the dynamics of the controlled system; the g function, usually called the output function, accounts for the observability of the system.

Equations (1) do not take any uncertainty into account: we know for sure that whenever we are in state x_t, we will observe the output $g(x_t)$; and that if we apply control u_t in this state, we will end up in state $f(x_t, u_t)$. A possible way to bring uncertainty into the picture is by introducing two random variables v and w that represent the effects of the unknown factors, or *disturbances*. For instance, we might write

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) + v_t \\ y_t &= g(x_t) + w_t\,. \end{aligned}$$

To use this technique, we must be able to somehow specify the values taken by the v_t and w_t variables, for instance by defining two probability distributions for these variables. In other words, we must provide a *model of the uncertainty* that affects our model. This model can be provided for many physical systems; e.g., errors in the action of many effectors may be effectively modeled by assuming a Gaussian distribution over v_t. Unfortunately, in the case of most of the uncertainty that affects real-world unstructured environments, we are not able to precisely characterize and quantify the disturbances.

As an example of this, consider the uncertainty induced in the environment by the presence of people. People walk around, and they may change the position of objects and furniture. The actions of most persons cannot be described by a deterministic or a stochastic process — we just cannot write a meaningful probability distribution of when I will go to the printer room. As a consequence, we cannot write a meaningful model of the disturbances induced in the state transition function f by the presence of people. A similar observation can be made for the disturbances introduced in the output function g. The reliability of visual recognition, for instance, is influenced by the lighting conditions, which may depend on the cloudiness. And the reliability of the distances measured by

a sonar sensor is influenced by the geometry and the reflectance property of the objects in the environment. In each case, a probabilistic (or otherwise) model of uncertainty is either meaningless, or overly complex to obtain.

One possible way to approach this problem is to attack the uncertainty at its very source, by carefully engineering the robot and its workspace so that the uncertainty is minimized and the residual uncertainty can be fully characterized in some way. As we have noticed, this is typically done in industrial robots, where the dynamics of the work-cell is completely determined, and sensing is limited to internal precise sensors that monitor the position of the robot's parts. Some amount of environment engineering has often been applied to autonomous robotics too: from the early days of Shakey [28], where the observation conditions were carefully controlled, and the set of possible "unforeseen" events was known a priori; to the current service robots that patrol hospital floors by following a white or magnetic strip.

Careful engineering can result in good performance, but it has obvious drawbacks. First, adding sophisticated sensors to a robot may enormously increase its cost and fragility; domestic or service robots are expected to use cheap and "easy" sensors (typically, sonar sensors). Second, having to modify the environment is usually undesirable, as it increases the costs and reduces the autonomy of the robot: we do not want an autonomous wheelchair to be restricted to move in selected streets and buildings. Third, sometimes engineering the environment is just impossible: think of a rescue robot going to a disaster area. Fourth, relying on such engineering may reduce the robustness of the robot; for instance, robots that follow a white strip tend to get hopelessly lost if a long portion of the strip is obscured. Finally, and perhaps most importantly, not all sources of uncertainty can be eliminated in this way, and some uncertainty is inherent to the nature of the environment: the actions of humans and the variability of the lighting conditions are typical examples of this.

If we want to build easily available robots that inhabit our homes, offices, or factory floors, we should accept the idea that the platform cannot be overly sophisticated, and that the environment should not be modified. Hence, we should strive to build robust control programs that reliably perform complex tasks in spite of the environmental uncertainties.

A strategy that has been widely employed in the robotics literature has been to abandon the idea to completely model the environment at the design phase, and endow the robot with the capability of building or updating the model on-line. This strategy led to the so-called *hierarchical* architectures, sketched in Figure 2. The robot uses exteroceptive sensors to acquire a model of the environment as it is at the moment when the task must be performed.[2] From this model, a planning program builds a plan that will perform the given task in

[2] Exteroceptive sensors, like a camera or a sonar sensor, observe the state of the environment; proprioceptive sensors, like a compass or shaft encoders on the wheels, observe the state of the robot's body. Although exteroceptive sensors are usually mounted on the robot, we prefer to draw them as a separate entity to emphasize the difference between exteroceptive and proprioceptive information.

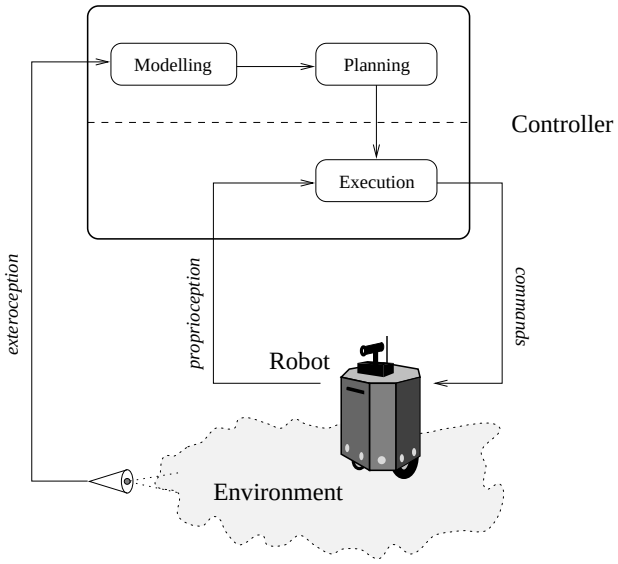

Fig. 2. Hierarchical architecture. The high-level layer builds a model of the environment and generates a plan for action. The low-level blindly executes this plan.

the given environment. This plan is then passed to a lower-level control program for execution. Typically, execution proceeds "blindly" — the controller may use a model of the robot and monitor the state of the robot's effectors (proprioception), but it does not try to sense or model the environment anymore. In a sense, the hierarchical approach factors the environment out of the controlled system, thus making the control problem tractable. This approach has been extensively used in the robotics literature; in most cases, the plan consists of a path leading to a goal position, and execution consists in tracking this path.

It is not difficult to see the limitations of the hierarchical approach when dealing with real-world environments. The model acquired by the robot is necessarily incomplete and inexact, due to the uncertainty in perception. Moreover, this model is likely to rapidly become out of date in a dynamic environment, and the plan built from this model will then turn out to be inadequate to the situation actually encountered during execution. The fact that the modeling and planning processes are usually complex and time consuming exacerbates this problem. Intuitively, the feedback loop with the environment must pass through all these processes — for this reason, this approach is also known as the "Sense-Model-Plan-Act", or SMPA approach. The complexity of the processes in the SMPA loop makes the response time of the robotic system far too long for dynamic environments (of the order of seconds).

By the mid-eighties technological improvements had caused the cost of mobile platforms and sensors to drop, and mobile robots began to appear in several AI

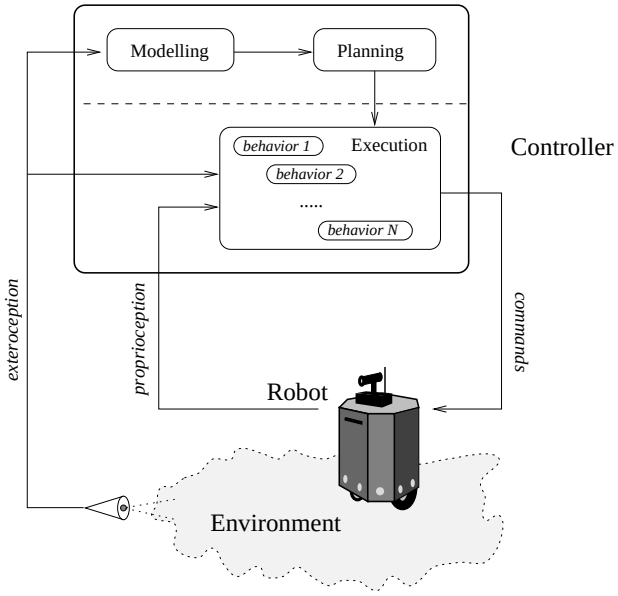

Fig. 3. Hybrid architecture. The lower layer uses perception to dynamically adapt plan execution to the environmental contingencies. Complexity in this layer is managed by a *divide et impera* strategy.

research labs. Research on autonomous navigation was strongly pushed, and a number of new architectures were developed that tried to integrate perception and action more tightly. The general feeling was that planning should make as few assumptions as possible about the environment actually encountered during execution; and that execution should be sensitive to the environment, and adapt to the contingencies encountered. To achieve this, perceptual data has to be included into the executive layer, as shown in Figure 3 (we'll see the meaning of the "behavior" modules in a moment). Architectures of this type are often called *hybrid* because they combine ideas from hierarchical architectures and from behavior-based architectures [5].

Although a seemingly simple extension, the inclusion of perceptual data in the execution layer has two important consequences. First, it makes robot's interaction with the environment much tighter, as the environment is now included in a closed-loop with the (usually fast) execution layer. Second, the complexity of the execution layer has to be greatly increased, as it now needs to consider multiple objectives: pursuing the tactical goals coming from the planner; and reacting to the environmental events detected by perception.

Following the seminal works by Brooks [5], Payton [31] and Arkin [1], most researchers have chosen to cope with this complexity by a *divide et impera* strategy: the execution layer is decomposed into small independent decision-making

processes, or *behaviors*, as shown in Figure 3. Other decompositions of the execution layer are possible: for example, in Khatib's proposal [32], an "elastic" path given by the planner is first modified to avoid collisions with sensed obstacles, and then it is given to a path tracking process. In all cases, the execution layer uses local sensor information to decide immediate reactions to the environmental contingencies, while trying to promote the overall goals.

Hybrid architectures do not solve the autonomous navigation problem, but they provide a convenient framework in which the different sub-problems can be dealt with and integrated. We still need to decide how the uncertainty in each sub-problem should be addressed. The rest of this paper is devoted to illustrating some ways to do so using the specific tools of fuzzy logic. As we shall show, fuzzy logic has features that are particularly attractive in light of the problems posed by autonomous robot navigation. Fuzzy logic allows us to model different types of uncertainty and imprecision; to build robust controllers starting from heuristic and qualitative models; and integrate symbolic reasoning and numeric computation in a natural framework. In the next pages, we shall illustrated these points by using our work on the robot Flakey as a test case. (See [40] for an overview of the uses of fuzzy logic in autonomous robotics.)

3 Robust behavior

The first issue that we consider is the design of the individual behavior-producing modules that appear in Figure 3. Each one of these modules fully implements a control policy for one specific sub-task, or behavior, like following a path, avoiding sensed obstacles, or crossing a door.

Fuzzy control is credited with being an adequate methodology for designing robust controllers that are able to deliver a satisfactory performance in face of large amounts of noise in the input and of variability in the parameters. The key to this robustness is to be found in the interpolation mechanism implemented by fuzzy controllers, which embodies the idea that similar inputs should produce similar actions. In addition to this, the rule format and the use of linguistic variables make fuzzy control an adequate design tool for non-linear systems for which a precise mathematical model cannot be easily obtained, but for which heuristic control knowledge is available. Finally, fuzzy controllers lend themselves to efficient implementations, including hardware solutions. (See, for instance, [18, 22] for a reminder of the basic principles of fuzzy control.)

These characteristics fit well the needs of autonomous robotics, where: (i) a mathematical model of the environment is usually not available; (ii) sensor data is uncertain and imprecise; and (iii) real-time operation is of essence. It is no surprise, then, if fuzzy control has since long attracted the attention of robot developers, and it represents today the most common application of fuzzy logic in the robotics domain. Notable examples include the early fuzzy controller developed in 1985 by Sugeno and Nishida to drive a model car along a track [43]; and the more recent behavior-based fuzzy controllers included in the award-winning robots Flakey [6], Marge [30], and Moria [44]. In the rest of this section,

we detail our use of fuzzy control to implement basic behaviors in Flakey. (See [37, 39] for a more complete treatment.)

In our approach, we express desirable behavioral traits as quantitative *preferences*, defined over the set of possible control actions, from the perspective of the goal associated with that behavior. Following the formal semantic characterization of Ruspini [33, 34], we describe each behavior B in terms of a desirability function

$$Des_B : State \times Control \to [0, 1]\,,$$

where *State* is the internal state of the robot, including variables that represent relevant quantities in the environment or internal reference points, and *Control* is the set of possible robot's actions. For each state x and control c, the value of $Des_B(x, c)$ measures the desirability of applying the control c when the state is x *from the point of view of attaining the goal associated with* B. For example, if the value of the state suggest that there is an obstacle on the left of the robot, then right turning control actions will have higher desirability than left turning ones from the point of view of an obstacle avoidance behavior.

A desirability function is given in the form of a set R of fuzzy rules

$$\text{IF } A_i \text{ THEN } C_i, \quad i = 1, \ldots, n\,,$$

where A_i is a propositional formula in fuzzy logic whose truth value depends on the state, and C_i is a fuzzy set of control values. For each possible control value c, $C_i(c)$ quantifies the extent by which c is a good instance of C_i. From these fuzzy rules, a desirability function Des_R is computed by

$$Des_R(x, c) = [A_1(x) \wedge C_1(c)] \ \vee \cdots \vee \ [A_n(x) \wedge C_n(c)]\,, \tag{2}$$

where $\wedge$ and $\vee$ denote fuzzy conjunction and disjunction, respectively (these are min and max in our current implementation, but can be any t-norm / t-conorm pair in the general case). Intuitively, this equation characterizes a control c as being desirable in the state x, if there is some rule in R that supports c and whose antecedent is true in x. This interpretation of a fuzzy rule set is that of a classical (Mamdani type) fuzzy controller [22], generalized so as to allow each antecedent A_i to be an arbitrary fuzzy-logic formula.

IF (lane-too-right $\wedge$ $\neg$lane-angled-left) THEN turn-medium-right
IF (lane-too-left $\wedge$ $\neg$lane-angled-right) THEN turn-medium-left
IF (lane-angled-right $\wedge$ $\neg$centerline-on-left) THEN turn-smooth-right
IF (lane-angled-left $\wedge$ $\neg$centerline-on-right) THEN turn-smooth-left

Fig. 4. Fuzzy control rules for the FOLLOW behavior.

Figure 4 shows a set of rules that implement the FOLLOW behavior. This behavior is intended to make the robot proceed along the mid-line of a given

lane; it can be used to go down a corridor, of drive on a road. The fuzzy predicates used in the rule antecedents depend on the the current position of the lane with respect to the robot. The consequents of the rules are triangular fuzzy subsets of the space of possible steering (and, in general, velocity) commands.

A desirability function specifies, for each input variable value, a ranking over possible controls rather than a unique control value to apply in that situation. The robot eventually employs this ranking to *choose* one specific control $\hat{c}$ that is sent to the controlled system. A possible mechanism to accomplish that selection is centroid defuzzification:

$$\hat{c} = \frac{\int c\, Des_R(x, c)\, dc}{\int Des_R(x, c)\, dc}, \tag{3}$$

which computes the mean of possible control values, weighted by their degree of desirability. Centroid defuzzification has been found satisfactory in our experiments whenever the rules in a rule set do not suggest dramatically opposite actions. In these cases, centroid defuzzification obviously does not work as averaging of such multi-modal desirability measures might result in selection of a very undesirable choice (e.g., the best trade-off between avoiding an oncoming train by jumping to the left or to the right is hardly to stay on the track!). Our empirical strategy has been to design the rule sets so as to avoid production of multi-modal desirability functions — roughly, we insist that rules that propose opposite controls have mutually exclusive antecedents, so that only unimodal fuzzy sets are produced for every input. Other authors [47] have relied, however, on alternative defuzzification functions.

Up to this point, we have been a little vague on the content of the robot's internal state. In general, the state will contain variables holding the reference values for the controller, related to the behavior's goal. For example, in the FOLLOW behavior above, the state contains the current position of the lane to follow. For a path-tracking behavior, it may contain a representation of the path, or a set of way-points to achieve.

As we have discussed in the previous section, behaviors that rely on precomputed paths can be ineffective in real-world dynamic environments, as the environment actually encountered during execution may differ significantly from the model used in the planning phase. For this reason, most behaviors in current autonomous robots are sensor-based: the controller takes as input data from the (exteroceptive) sensors, rather than an internal reference, thus moving the robot with respect to the perceived features in the environment. Typical examples include moving along a wall or a contour, reaching a light source or a beacon, and avoiding obstacles. Sensor-based behaviors can be more tolerant to uncertainty, in that they consider the environment as it is during actual execution.

One way to implement sensor-based behaviors is to include data from the sensors in the internal state, and to use these data as input to the controller. For example, to go down a corridor, we could use the sonar sensors to keep sensor contact with the corridor's walls. In our approach, we take a slightly more involved route: we use sensor data to update variables in the state that

are related to the goal of the behavior, and then use these variables in the antecedents of the fuzzy rules.

More precisely, each goal-directed behavior in Flakey maintains in the state a *descriptor* of the object relevant to that behavior. For example, the FOLLOW behavior maintains a descriptor of the lane to follow, and uses the rules in Figure 4 with respect to it. An explicit procedure, called *anchoring* [35], is employed to maintain the correspondence between descriptors and environmental features detected by the sensors. Which features should be used for anchoring depends on the specific instance of the behavior. For example, to go down a given corridor, we use the FOLLOW behavior and anchor the lane to the walls of that corridor.

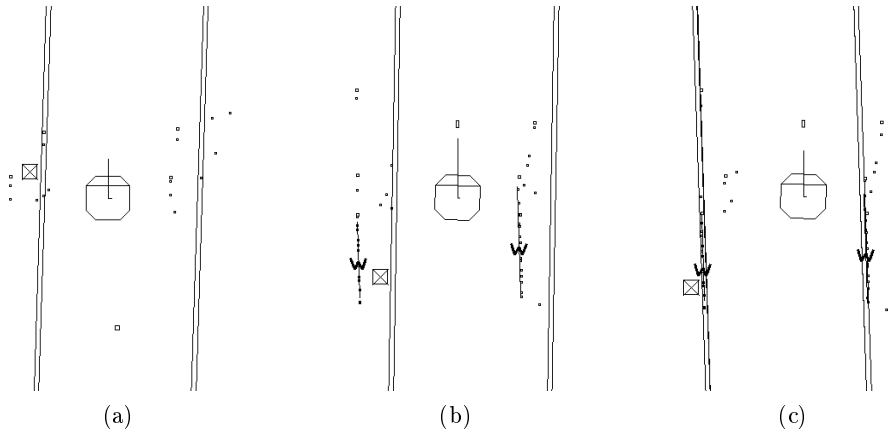

Fig. 5. Anchoring a corridor descriptor to sensor readings during corridor following; (a) the descriptor is used as a starting assumption; (b) the walls are detected; (c) the descriptor is updated.

Figure 5 shows an example of this. Initially (a), the robot is given a descriptor, produced from prior map information, of a corridor to follow (double lines), and FOLLOW operates with respect to this descriptor. After a while (b), enough sonar readings, marked by the small dots in the figure, are gathered to allow the perceptual routines to recognize the existence of two parallel walls, marked by "W." In this example, the descriptor does not match precisely the position of the real corridor: this may be due to errors in the robot's estimate of its own location, or to inaccuracies in the map. Finally (c), the perceived position of the walls is used to update the descriptor — i.e., the descriptor is *anchored* to the sensor data. As the rules of FOLLOW are based on the properties of the descriptor, anchoring implies that the robot will now follow the actual corridor. The corridor descriptor is normally re-anchored at each control cycle whenever the walls are visible. Note that the ability to recover from vague or imprecise assumptions originating from inexact prior knowledge is particularly important in

practice, since our maps typically contain only approximate metric information (see Section 5 below).

Descriptors serve several functions. First, they allow us to give a behavior an explicit *goal* by initializing its descriptor using the relevant data (e.g., the position of a specific corridor to follow). Second, descriptors act as sources of credible *assumptions* when perceptual data is not available, as is the case when first engaging a new corridor, or when the walls are momentarily occluded by obstacles. Third, they allow us to *decouple* the problem of control from the problem of interpreting noisy data; the latter problem is confined to the anchoring process. Finally, the use of descriptors results in more *abstract* behaviors than those obtained by a purely reactive approach. For example, the FOLLOW behavior above follows the general direction of the corridor, and not the precise contour of the walls; following the contour may produce wrong decisions when the walls are interrupted by obstacles and open doors.

Not all behaviors are goal-directed, and not all behaviors need a descriptor to represent their goal object. For example, the obstacle avoidance behavior KEEP-OFF is implemented in Flakey by purely reactive rules, that is, rules whose antecedents only depend on the current readings from the sonars.[3] Figure 6 shows a run of this behavior in a completely unknown environment. The robot wanders around at random while trying to stay away from static and moving obstacles as they were perceived. The smoothness of motion, both in turning and acceleration, can be seen in the wake of small dots that indicate the robot's trajectory (one dot per second).

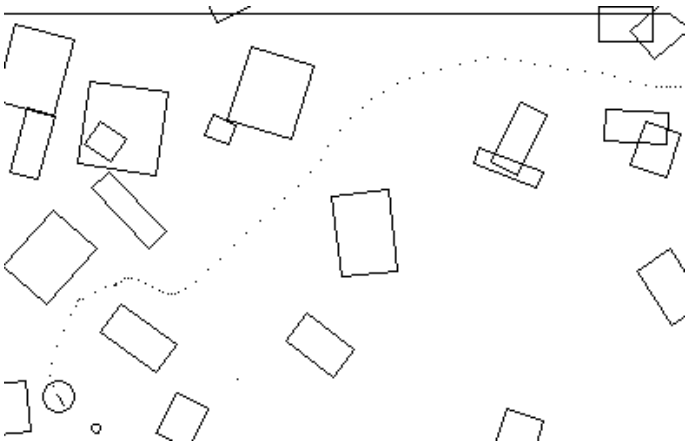

Fig. 6. A run of the obstacle avoidance behavior in an unknown environment.

[3] This is not entirely true, as KEEP-OFF also uses the last 50 readings in order to improve sonar coverage; this however does not change the essentially reactive nature of the behavior.

In our experience, fuzzy behaviors are generally easy to write and to debug, and they perform well in face of uncertainty. We have written a dozen behaviors for Flakey, including behaviors for avoiding obstacles, for facing a given object, for crossing a door, and reaching a near location, and so on. Each behavior typically consists of four to eight rules involving up to a dozen fuzzy predicates. The main difficulty that we have encountered is that the debugging and tuning of the behaviors must be done by trials and errors — this is the price to pay for not using a mathematical model of the controlled system. Although writing most behaviors was very easy, some comparatively complex behavior, like KEEP-OFF, required extensive fine-tuning. In future, we may consider the use of learning techniques (e.g., [4, 14, 25]).

4 Flexible task execution

A behavior is a small unit of control aimed at achieving one simple goal in a restricted set of situations. Autonomous robots need to perform complex tasks, usually requiring the activation and cooperation of a number of behaviors. In this section, we focus on the problem of how to organize and coordinate the execution of basic behaviors in order to perform a given complex task. With respect to Figure 3 above, this means that we focus on the link between the behavior-based execution at the lower level, and the goal-oriented planning activity at the higher level. The simplest example is the coordination of an obstacle avoidance behavior and a target reaching behavior to achieve the goal of safely reaching a given position in the presence of unexpected or moving obstacles.

Since the first appearance of behavior-based approaches in the mid-eighties, authors have noticed the importance of the problem of behavior coordination. Many proposals are based on a simple on-off *switching* scheme: in each situation, one behavior is selected for execution and is given complete control of (some of the) effectors [5, 8, 10, 29, 31]. Unfortunately, this simple scheme may be inadequate in situations where several criteria should be simultaneously taken into account. To see why, consider a robot that encounters an unexpected obstacle while following a path, and suppose that it has the option to go around the obstacle from the left or from the right. This choice may be indifferent to the obstacle avoidance behavior. However, from the point of view of the path-following behavior, one choice might be dramatically better than the other. In most implementations, the obstacle avoidance behavior could not know about this, and would take an arbitrary decision.

To overcome this limitation, several researchers have proposed coordination schemes that allow the parallel execution of different behaviors, and perform a weighted combination of the commands they issue. These schemes bring about the important issue of how to resolve conflicts between the outputs of different concurrent behaviors. Consider the case where several behaviors are simultaneously active. Ideally, we would like the robot to select the controls that best satisfy all the active behaviors. This may not be possible, though, if some behaviors suggest different actions. In the rest of this section, we show how behavior

coordination can be obtained by *context-dependent blending* (CDB), the behavior coordination technique that we have implemented in Flakey.

The key observation to CDB is that behaviors are not equally applicable to all situations: for instance, corridor following is most applicable when we are in a corridor and the path is clear, while obstacle avoidance is more applicable when there is an obstacle on the way. Correspondingly, we associate to each behavior a *context* of applicability, expressed by a formula in fuzzy logic. Given a set $\mathcal{B} = \{B_1, \ldots, B_k\}$ of behaviors, we denote by Cxt_i the formula representing the context of B_i. We then define the *context-dependent blending* of the behaviors in $\mathcal{B}$ to be the composite behavior described by the following desirability function:

$$Des_{\mathcal{B}}(s,c) = (Cxt_1(s,c) \wedge Des_1(s)) \ \vee \cdots \vee \ (Cxt_k(s,c) \wedge Des_k(s)). \qquad (4)$$

Intuitively, the composite desirability function is obtained by merging the individual recommendations from all the behaviors, each one discounted by the truth value of the corresponding context in the current state.

Equation (4) extends (2) to the meta-level, by merging the outputs of a set of behaviors rather than those of a set of control rules. Correspondingly, context-dependent blending may be expressed by a set of fuzzy meta-rules (or "context-rules") of the form

$$\text{IF } Cxt_j \text{ THEN } B_j\,, \qquad j = 1, \ldots, m, \qquad (5)$$

where Cxt_j is a formula in fuzzy-logic describing a context, and B_j is a behavior.[4] A set of context-rules of this type can be evaluated to produce a combined desirability function using (4). This desirability function can then be defuzzified by (3) to produce a crisp control. This way to realize context-dependent blending corresponds to using a hierarchical fuzzy controller, as schematized in Figure 7. This is how CDB has been implemented in Flakey.

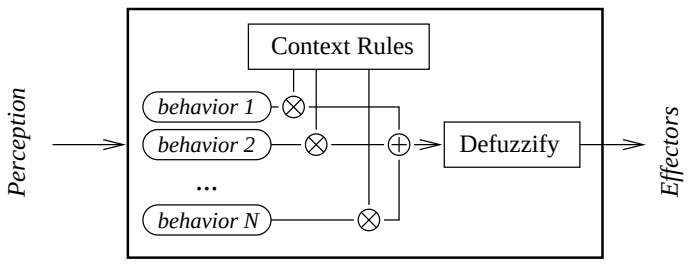

Fig. 7. A hierarchical fuzzy controller that implements Context-Dependent Blending.

[4] The hierarchical structure is reminiscent of the one previously proposed by Berenji et al. [2]. Context-dependent blending generalizes and extends that proposal by allowing dynamic modification of the degrees of importance of each goal.

A few observations should be made on Figure 7. First, it is essential that the defuzzification step be performed *after* the combination: the decision taken from the collective preference can be different from the result of combining the decisions taken from the individual preferences [36]. Second, although in Figure 7 all the context-rules are grouped in one module, the same effect can be obtained by including each context-rule inside the corresponding behavior; this solution would be more amenable to a distributed implementation. Third, CDB can be iterated: we can use the structure in Figure 7 to implement each individual behavior, and combine several such (complex) behaviors using a second layer of context-rules; and so on. (Defuzzification should still be the last step.)

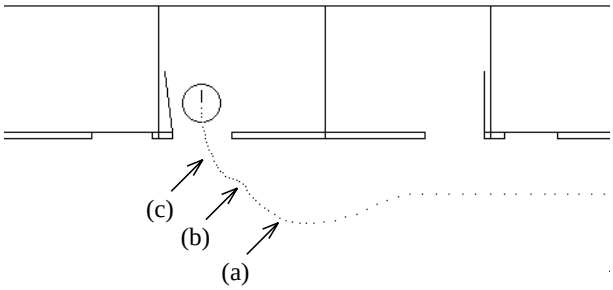

Fig. 8. Context-Dependent Blending of behaviors can compensate for inexact prior knowledge: (a) the CROSS behavior uses a wrong estimate of the door position; (b) KEEP-OFF intervenes to avoid a collision; (c) by blending both behaviors, the robot safely passes through the opening.

The CDB mechanism can be used to combine reactive and goal-directed behaviors. For example, the following context rules determine the blending between the obstacle avoidance behavior KEEP-OFF, and a behavior CROSS for passing through a doorway.

IF obstacle-close THEN KEEP-OFF
IF near-door $\wedge$ $\neg$(obstacle-close) THEN CROSS

An interesting outcome of this type of combination is an increased tolerance to imprecision in the prior knowledge. Figure 8 illustrates this point. In (a), the CROSS behavior is relying on the estimated position of the door to cross based on map information. This estimate turns out to be off by some 40 centimeters, and the robot is grossly misheaded. In (b), the edge of the door has been detected as a close obstacle, and the preferences of KEEP-OFF begin to dominate, thus causing the robot to slow down and re-orient toward the free space corresponding to the door opening. Later (c), both behaviors cooperate to lead the robot though the office door — i.e., through the sensed opening that is more or less at the assumed position. During these maneuvers, the preferences of both behaviors are considered, through (4), and contribute to the control choices.

Context	*Behavior*
obstacle	KEEP-OFF(OG)
¬obstacle ∧ at(Corr-2) ∧ ¬at(Corr-1)	FOLLOW(Corr-2)
¬obstacle ∧ at(Corr-1) ∧ ¬near(Door-5)	FOLLOW(Corr-1)
¬obstacle ∧ near(Door-5) ∧ anchored(Door-5)	CROSS(Door-5)
¬anchored(Corr-2)	SENSE(Corr-2)
¬anchored(Corr-1)	SENSE(Corr-1)
¬anchored(Door-5)	FIND(Door-5)

Fig. 9. A set of context rules forming a plan for reaching Room-5 in the environment of Figure 10 (see the text for explanations).

Context-dependent blending can also be used to execute full plans. In [36] we have proposed to represent plans as sets of *situation→action* rules of the form (5). These rules can be directly executed by the hierarchical controller above; interestingly, and differently from many languages for representing reactive plans, these rules can also be easily generated by classical AI planning techniques. For example, the set of rules shown (in a simplified form) in Figure 9 constitutes a plan to navigate to Room-5 in the environment shown in Figure 10 (top). The arguments passed to the behaviors are the object descriptors to be used to control motion ("OG" denotes the occupancy grid built from the sensor readings and used for obstacle avoidance). This plan has been generated by a simple goal regression planner from a topological map of the environment, and from a description (provided by the behavior designer) of the preconditions and effects of the basic behaviors available to the robot.[5] In this experiment, the map does not contain the position of Door-5, but states that it is the only doorway in Corr-1. Hence, the plan includes the behavior FIND, aimed at detecting the door; it also includes two more perception-oriented behaviors, called SENSE, whose aim is to help perception of the corridor walls when these are not anchored.

The lower part of Figure 10 shows the time evolution of the context values for the behaviors in this plan during the reported run. At (a), Flakey is in Corr-2, and FOLLOW(Corr-2) is active. As the robot approaches Corr-1 (c), the truth value of the context of FOLLOW(Corr-1) increases, and the robot smoothly turns into Corr-1 and follows it. At (g), the side sonars detect the door and Door-5 is anchored, and thus Flakey engages in the door crossing maneuvers. Notice that the perceptual behaviors become inactive once the corresponding feature has been detected and anchored (b, d, f, g), but they may come back to activity if anchoring is lost, like in (e), where a number of boxes occluded the left wall for some length. Also notice that at several occasions in the run the KEEP-OFF behavior blends in to go around unforeseen obstacles.

The above example shows that sets of fuzzy context rules can represent plans that are tolerant to uncertainty. Our plan could accommodate the extremely

[5] The planner uses the preconditions to build the contexts of activation. This is similar to building the triangular tables used in the robot Shakey [28].

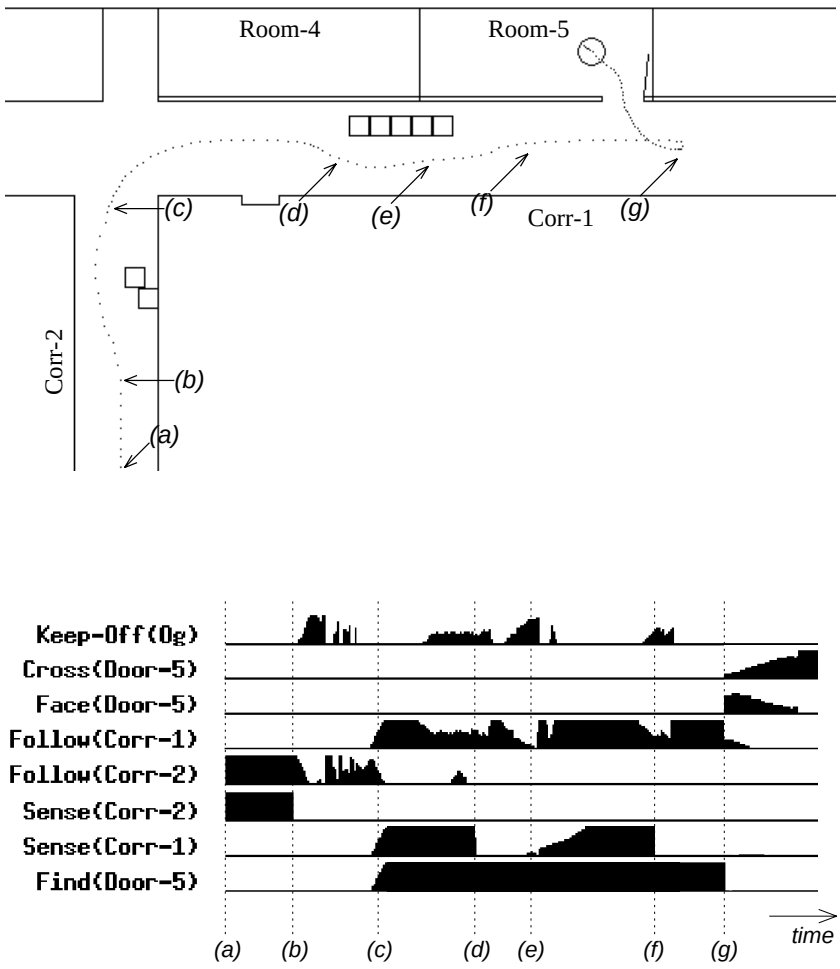

Fig. 10. A run of the plan in Figure 9 (top), and the corresponding evolution over time of the activation of the behaviors (bottom).

weak information available at planning time about the position of Door-5. The highly conditional nature of this plan also allows the robot to cope with the uncertainty in the effect of actions. For example, the effect of SENSE is to have the corridor anchored; however, this effect may be undone by later actions — like following the corridor in the area where a wall is occluded. This problem could not be detected at planning time, as the planner does not know about the obstacles in the corridor. However, the context rules just re-activate the SENSE behavior when the need to do so arises. (In this sense, our plans are similar to Schopper's universal plans [41].)

Following its implementation on Flakey [37], CDB has been used by several researchers in autonomous robotics [13, 26, 44–46]. CDB provides a flexible means to implement complex behavior coordination strategies in a modular way using a logical rule format. This modularity simplifies writing and debugging of complex behaviors: the individual behaviors and the coordination rules can be debugged and tuned separately; a few unforeseen interferences may remain, but identifying and correcting these have proved in our experience to be a much easier task than writing and debugging a complex monolitic behavior.

CDB is strictly more general than other coordination schemes commonly used in robotics. It can simulate both on-off behavior switching, and the vector summation scheme used in the popular potential field techniques [16, 20], but it differs from them in the general case. The fact that the same format is used for the control rules and the context rules has several advantages: it allows us to write increasingly complex behaviors in a hierarchical fashion; it facilitates the use of standard AI planning techniques to generate coordination strategies that achieve a given goal; and it allows us to formally analyze the resulting combined behavior. A more detailed analysis of CDB can be found in [36].

5 Approximate self-localization

The approaches that we have discussed up to here cope with uncertainty by *tolerating* it, that is, by building robust control programs that try to achieve their goals despite the presence of errors and inaccuracies in sensing and in prior knowledge. Another way to approach the uncertainty problem is by explicitly *representing* it, and by reasoning about its effects. To see the difference, consider the problem of building a robot navigation plan. A tolerant approach would generate a highly conditional plan, where a good amount of decision is postponed until the execution phase, and which includes provisions for real-time sensing in order to reactively adapt to the execution contingencies. The approach presented in the last section is an example of this. By contrast, an explicit representation approach would try to model the different sources of uncertainty, by estimating the (say) probability that an action will fail, that the position of an object will be different from what is expected, and so on; and would generate a plan that maximizes some measure of certainty of success.

One issue in robotics where an explicit representation of uncertainty is often preferred is reasoning with spatial knowledge. Spatial information about the environment can be obtained from several sources, including different types of sensors, motion history, and prior knowledge. If we have a measure of uncertainty for all items of information, then we can combine them to reduce uncertainty, for instance by synchronously fusing the data from different sensors, and/or by diachronically accumulating these data over time. The result of this process is usually to build a *map* of the environment.

Using a map brings about the important problem of *self-localization*: how to estimate of the robot's position with respect to this map. Knowing one's position is necessary to relate the perceived world to the map, and to decide which ac-

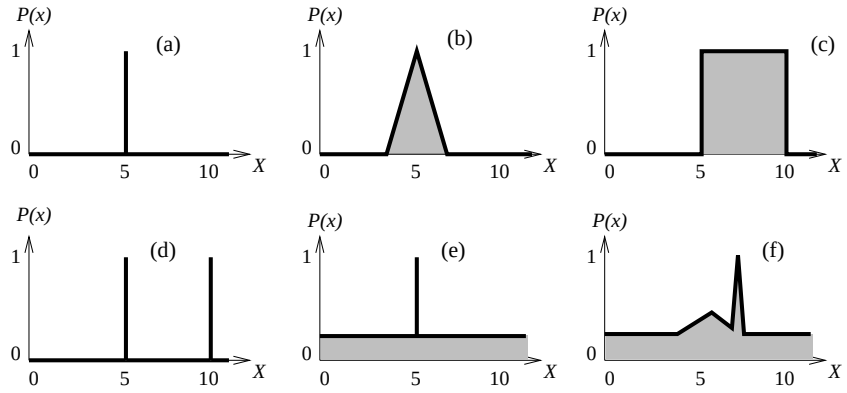

Fig. 11. Representing different types of uncertainty by fuzzy sets: (a) crisp; (b) vague; (c) imprecise; (d) ambiguous; (e) unreliable; (f) combined.

tions to perform. Self-localization can use information from the odometric sensors (e.g., wheel encoders) to update the robot's position. Unfortunately, odometry has cumulative errors, and the robot's odometric estimate of its position can diverge from reality without bounds. Landmark-based techniques are commonly used to correct this problem [19]. A landmark can be any perceptually recognizable feature of the environment that is represented in the robot's map, like doors or walls. The robot uses the difference between the perceived and the expected locations of perceived landmark to correct the estimate of its own position.

Most existing approaches to landmark-based self-localization are based on a probabilistic representation of spatial uncertainty, and use some form of Kalman filter [15, 23, 27, 42] to update the robot's position estimate. These approaches can be very effective, provided that: (1) the underlying uncertainty can be given a probabilistic interpretation; (2) the initial estimate is good enough; and (3) the required data is available. In particular, the latter requirement means that (3a) we have an accurate dynamic model of the robot; (3b) we have an accurate stochastic model of the sensors; (3c) these systems do not change in unpredictable ways with time.

These conditions are pretty demanding, and they may easily be violated in the case of autonomous robots. In these cases, fuzzy logic may offer valuable alternatives which require less demanding assumptions: for example, fuzzy-based localization methods typically need only qualitative models of the system and of the sensors. In what follows, we illustrate this point by discussing the way we have used fuzzy logic for approximate map representation and for self-localization in Flakey (see [38] for more on this).

We represent the approximate location of an object by a fuzzy subset of a given space, read under a possibilistic interpretation [48, 49]: if P_o is a fuzzy set representing the approximate location of object o, then we read the value of $P_o(\boldsymbol{x}) \in [0, 1]$ as the *degree of possibility* that o be actually located at $\boldsymbol{x}$. This

representation allows us to model different aspects of locational uncertainty. Figure 11 shows six approximate locations in one dimension: (a) is a crisp (certain) location; in (b), we know that the object is located at approximately 5 (this is commonly referred to as "vagueness"); in (c), it can possibly be located anywhere between 5 and 10 ("imprecision"); in (d), it can be either at 5 or at 10 ("ambiguity"); (e) shows a case of "unreliability": we are told that the object is at 5, but the source may be wrong, and there is a small "bias" of possibility that it be located just anywhere. As an extreme case, we represent total ignorance by the "vacuous" location $P(x) = 1$ for all x: any location is perfectly possible. Finally, (f) combines vagueness, ambiguity and unreliability. Clearly, the information provided by a measurement device can present any of the above aspects, alone or in combination. It is important to emphasize that a degree of possibility is *not* a probability value: e.g., there is no necessary relation between the observed frequency of a location and its possibility; and degrees of possibility of disjoint locations need not add up to one.

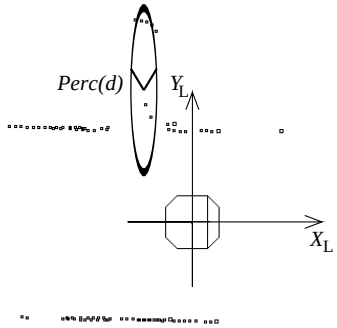

Fig. 12. The approximate location of a perceived door. The radiuses of the ellipsoid are proportional to the width of fuzzy location along the corresponding component. The "V" in the middle indicates the width of the fuzzy set of possible orientations.

We use fuzzy sets to represent the approximate position of objects in the map of the environment. More precisely, our approximate maps contain the approximate positions of the major environmental features (doors, walls, corridors, ...), represented by a fuzzy point in R^3 — that is, a fuzzy subset of the set of (x, y, θ) coordinates in a global Cartesian frame, where θ is the orientation of the object measured with respect to the X axis. We also use fuzzy sets to represent the approximate position of features locally perceived by the sensors. For example, Figure 12 shows the fuzzy position *Perc(d)* of a door d detected by the sonar sensors.[6] The shape of this fuzzy set is due to the fact that the sonar signature

[6] In our current implementation, a fuzzy location is encoded by three triangular fuzzy sets representing its projections on the three axes X, Y and Θ. Future implementations will use less restrictive representations.

of a door on a side can only give a reliable indication about the position of the door along the longitudinal axis, while the distance and orientation of the door remains extremely vague. This set might also include a "bias" to reflect the unreliability of perceptual recognition, as in Figure 11 (e-f).

The above representation can be used by an algorithm for approximate self-localization. In [38] we have proposed a recursive algorithm of this type. Its outline is very simple. At each time-step t, the robot has an approximate hypothesis of its own location in the map, represented by a fuzzy subset H_t of the map frame. During navigation, the robot's perceptual apparatus recognizes relevant features, and searches the map for matching objects using a fuzzy measure of similarity. Each matching pair is used to build a *fuzzy localizer*: a fuzzy set representing the approximate location in the map where the robot *should be* in order to see the object where the feature has been observed. So, each localizer provides one imprecise source of information about the actual position of the robot. All these localizers, plus odometric information, are combined by fuzzy intersection to produce the new hypothesis H_{t+1}, and the cycle repeats.

Figure 13 illustrates the operation of our algorithm. Each screen-dump shows: on the left, the robot's internal view of its local workspace (details of this view are not important for the discussion here); on the right, the internal map of the environment used by the robot; this includes the robot self-location estimate H_t, represented by the ellipsoid around the robot's shape (the narrow cone in front of the robot indicates the uncertainty in orientation). In (a) the robot has some uncertainty as to its position along the X axis; however, there is little uncertainty along the Y axis and in the orientation, due to a previous matching of the corridor's walls. After a while, the sonar sensors detect a door D on the robot's right. By matching this percept to the approximate map, the robot builds the localizer loc_D shown in (b): this indicates that the robot should be located somewhere in the fuzzy set loc_D. The robot also builds the dead reckoning localizer loc_{dr} from the previous H_t using odometric information from the wheel encoders. The loc_D and the loc_{dr} localizers are then intersected to produce the new, narrower estimate H_{t+1} of the robot's position, as shown in (c).

The fuzzy self-localization algorithm above was able to keep Flakey well registered during several navigation experiments in an unstructured office environment. The algorithm was also able to produce a correct location hypothesis starting from a situation of total ignorance — a difficult task for probability-based methods. Finally, fuzzy locations can be smoothly integrated inside fuzzy-logic based controllers as the ones reported above. This is an important issue, as locational information eventually has to be used to take decisions.

6 Concluding remarks

Several factors contribute to make truly autonomous robots still a dream — although a not entirely unrealistic one. For one thing, we do not fully understand the principles that underlie intelligent agency, and the intricate relation between agents and their environments. This is a fundamental problem that has been

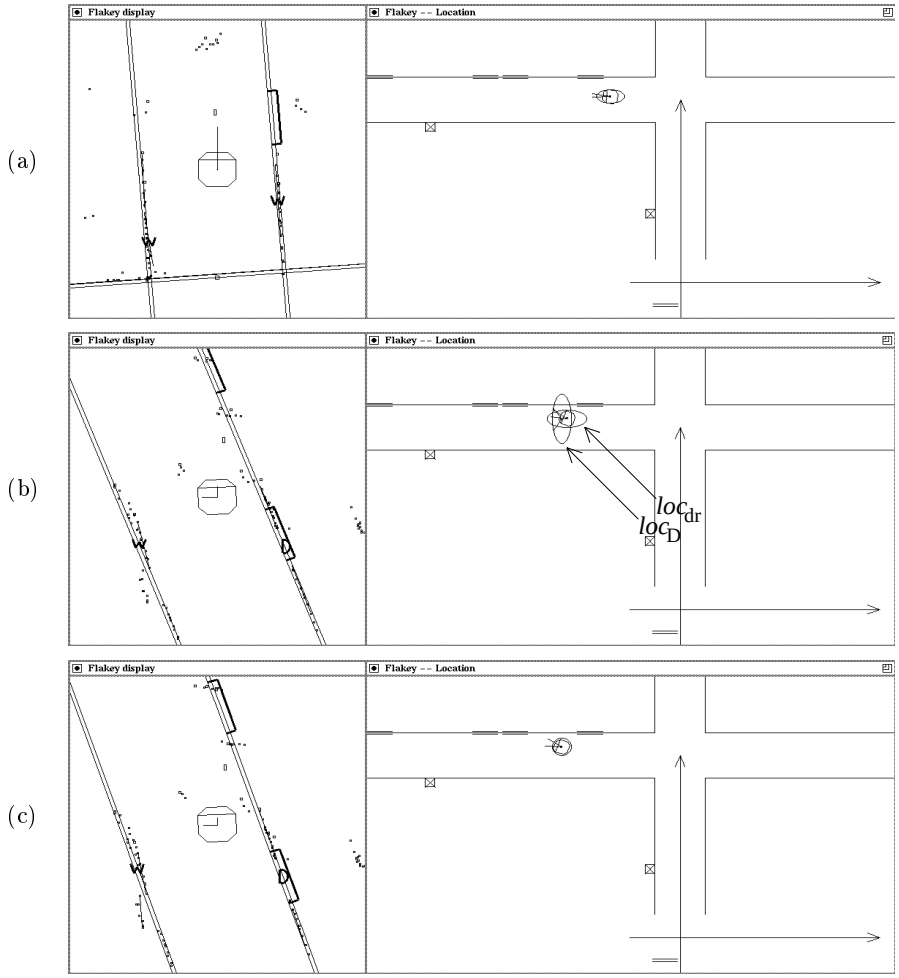

Fig. 13. Fuzzy self-localization. Left: the robot's internal view of its surroundings. Right: the robot's map of the environment, with (a) initial location hypothesis H_t; (b) dead reckoning and door localizers; (c) updated hypothesis H_{t+1}.

attracting attention from philosophers and scientists for centuries. On a more practical key, autonomous robotics needs the contributions of a number of different technologies developed in different fields, including artificial intelligence, control theory, vision, signal processing, mechatronics, and so on. Integrating these technologies in one system is a hard engineering problem. Finally, autonomous robots must operate in face of the large uncertainty which is inherent to the nature of real-world, unstructured environments, and to the robot-environment interaction. It is this last aspect that we have discussed in this note.

As with most problems involving uncertainty, we can take three different attitudes toward the presence of uncertainty in the robotics domain:

1. Get rid of it, by carefully engineering the robot and/or the environment;
2. Tolerate it, by writing robust programs able to operate under a wide range of situations, and to recover from errors; or
3. Reason about it, by using techniques for the representation and the manipulation of uncertain information.

Industrial robots show us that we can achieve remarkable performance taking the first attitude. If we want to build easily available robots that inhabit our homes, offices, or factory floors, though, we must live with the idea that the platform cannot be overly sophisticated, and that the environment should be only minimally modified. Then, we must consider the second or third attitudes to some extent.

By taking the tolerant attitude, we try to build control programs that provide a reasonable performance in face of large variability in the parameters of the model (due to poor prior knowledge), and large uncertainty in the state of the system (due to poor sensor data and errors in actions). We may express this by saying that these programs should rely on a weak model of the "robot + environment" system. By contrast, taking the third attitude we make the model richer, by explicitly including information about the uncertainty and variability of the parameters. The aim here is to build control programs that reason about this uncertainty in order to choose the actions that are more likely to produce the intended results.

Numerous examples of all these attitudes exist in the robotic domain, showing that often the same problem can be attacked with any one of them. It is difficult to say which uncertainty should be better eliminated by engineering the robot or the environment, and which one should be tolerated or explicitly represented. In general, there is a fundamental tradeoff between using better engineering or better programs (see [6] for an illustration of this tradeoff in the context of a robotic competition).

In this note, we have presented embodiments of the second and the third attitudes to address a few important issues in autonomous robot navigation. For the tolerant attitude, we have shown how we can implement robust behaviors that can tolerate uncertainty in sensing and action; and how we can use these behaviors to flexibly execute plans built from uncertain prior knowledge while adapting to the actual contingencies encountered. As for the representation attitude, we have shown how we can represent approximate spatial information, and reason about it in order to infer a good estimate of the robot's location. The solutions that we have presented were originally developed for the mobile robot Flakey, leading to good experimental results. These solutions have since been ported to a second robot, Edi; they have also been included in the general robot architecture Saphira [17], used on several robotic platforms. Two of these solutions are particularly novel: context-dependent blending, and fuzzy self-localization. The former is being increasingly applied in the autonomous robotics field; for the latter, some related work has been independently developed [12].

Needless to say, there are many other issues in autonomous robotics for which we need to take uncertainty into account, and which we have not touched here. Three omitted issues are particularly worth mentioning. First, perceptual interpretation and modeling. This is a huge field in itself, including a number of subproblems like signal processing, sensor fusion, image interpretation, 3D modeling, and active perception, just to mention a few. Each subproblem is the object of a vast literature, where dealing with uncertainty often plays an important role (see, e.g., [3] for a starting point). Second, planning. Several planning techniques that explicitly take uncertainty into account have been proposed both in the robotics and the AI literature (see, e.g., [7, 21]). Third, learning. Learning techniques can cope with the uncertainty in the model of the system by giving the agent the ability to discover and adaptively modify the model by itself (whether this model is explicitly or implicitly represented is not an issue here). Machine learning techniques have been widely used in autonomous robotics: an interesting sample can be found in two recent special issues [9, 11].

Whatever the issue, the problem of uncertainty needs to be addressed by choosing one of the three attitudes above and, if we opt for an explicit approach, by choosing a specific uncertainty formalism. Although most of the literature on dealing with uncertainty in robotics is based on probabilistic techniques, solutions based on fuzzy logic are being increasingly reported (see [40] for an overview). We have hinted at a few advantages of this choice in the pages above. Still, we emphasize that the choice of the formalism to use depends on the robot-environment-task configuration: there is no "best" way to deal with uncertainty in robotics, but there are as many best ways as there are different robots, environments and tasks.

Acknowledgments

This research was partly supported by the BELON project, founded by the *Communauté Française de Belgique*. The work on Flakey was performed while the author was with the AI Center of SRI International, in strict collaboration with Enrique Ruspini, Kurt Konolige and Leonard Wesley.

References

1. R. C. Arkin. Motor schema based navigation for a mobile robot. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 264–271, 1987.
2. H. Berenji, Y-Y. Chen, C-C. Lee, J-S. Jang, and S. Murugesan. A hierarchical approach to designing approximate reasoning-based controllers for dynamic physical systems. In *Proc. of the Conf. on Uncertainty in Artif. Intell.*, pages 362–369, Cambridge, MA, 1990.
3. I. Bloch. Information combination operators for data fusion: A comparative review with classification. *IEEE Trans. on Systems, Man, and Cybernetics*, A-26(1):52–67, 1996.
4. A. Bonarini and F. Basso. Learning to compose fuzzy behaviors for autonomous agents. *Int. J. of Approximate Reasoning*, 17(4):409–432, 1997.

5. R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.
6. C. Congdon, M. Huber, D. Kortenkamp, K. Konolige, K. Myers, E. H. Ruspini, and A. Saffiotti. CARMEL vs. Flakey: A comparison of two winners. *AI Magazine*, 14(1):49–57, Spring 1993.
7. C. Da Costa Pereira, F. Garcia, J. Lang, and R. Martin-Clouaire. Planning with graded nondeterministic actions: a possibilistic approach. *Int. J. of Intelligent Systems*, 12:935–962, 1997.
8. M. Dorigo and M. Colombetti. *Robot shaping: an experiment in behavior engineering*. MIT Press / Bradford Books, 1997.
9. M. Dorigo (Editor). Special issue on: Learning autonomous robots. *IEEE Trans. on Systems, Man, and Cybernetics*, B-26(3), 1996.
10. J. R. Firby. An investigation into reactive planning in complex domains. In *Proc. of the AAAI Conf.*, pages 202–206. AAAI Press, Menlo Park, CA, 1987.
11. J. A. Franklin, T. M. Mitchell, and S. Thrun (Editors). Special issue on: Robot learning. *Machine Learning*, 23(2-3), 1996.
12. J. Gasós and A. Martín. Mobile robot localization using fuzzy maps. In T. Martin and A. Ralescu, editors, *Fuzzy Logic in AI — Selected papers from the IJCAI'95 Workshop*, number 1188 in Lecture Notes in AI, pages 207–224. Springer-Verlag, 1997.
13. S. G. Goodridge, M. G. Kay, and R. C. Luo. Multi-layered fuzzy behavior fusion for reactive control of an autonomous mobile robot. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 579–584, Barcelona, SP, 1997.
14. F. Hoffmann and G. Pfister. Evolutionary learning of a fuzzy control rule base for an autonomous vehicle. In *Proc. of the Conf. on Information Processing and Management of Uncertainty (IPMU)*, pages 1235–1238, Granada, SP, 1996.
15. A. H. Jazwinski. *Stochastic processes and filtering theory*. Academic Press, 1970.
16. O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.
17. K. Konolige, K.L. Myers, E.H. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1):215–235, 1997.
18. R. Kruse, J. Gebhardt, and F. Klawonn. *Foundations of Fuzzy Systems*. Wiley and Sons, 1994.
19. B. J. Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2:129–153, 1978.
20. J. C. Latombe. *Robot Motion Planning*. Kluver Academic Publishers, Boston, MA, 1991.
21. A. Lazanas and J. C. Latombe. Motion planning with uncertainty: a landmark approach. *Artificial Intelligence*, 76(1-2):285–317, 1995.
22. C. C. Lee. Fuzzy logic in control systems: fuzzy logic controller (Parts I and II). *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):404–435, 1990.
23. J. J. Leonard, H. F. Durrant-Whyte, and I. J. Cox. Dynamic map building for an autonomous mobile robot. *Int. J. of Robotics Research*, 11(4):286–298, 1992.
24. V. J. Lumelsky and R. A. Brooks. Special issue on sensor-based planning and control in robotics: Editorial. *IEEE Trans. on Robotics and Automation*, 5(6):713–715, 1989.
25. M. Maeda, M. Shimakawa, and S. Murakami. Predictive fuzzy control of an autonomous mobile robot with forecast learning function. *Fuzzy Sets and Systems*, 72:51–60, 1995.
26. F. Michaud. Selecting behaviors using fuzzy logic. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 585–592, Barcelona, SP, 1997.

27. P. Moutarlier and R. Chatila. Stochastic multisensory data fusion for mobile robot location and environment modeling. In *5th Int. Symp. on Robotics Research*, pages 207–216, Tokyo, JP, 1989.
28. N. J. Nilsson. SHAKEY the robot. Technical Note 323, SRI Artificial Intelligence Center, Menlo Park, CA, 1984.
29. N. J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.
30. I. Nourbakhsh, S. Morse, C. Becker, M. Balabanovic, E. Gat, R. Simmons, S. Goodridge, H. Potlapalli, D. Hinkle, K. Jung, and D. Van Vactor. The winning robots from the 1993 robot competition. *AI Magazine*, 14(4):51–62, Winter 1993.
31. D. W. Payton. An architecture for reflexive autonomous vehicle control. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1838–1845, San Francisco, CA, 1986.
32. S. Quinlan and O. Khatib. Elastic bands: connecting path planning and robot control. In *Procs. of the Int. Conf. on Robotics and Automation*, volume 2, pages 802–807, Atlanta, Georgia, 1993. IEEE Press.
33. E. H. Ruspini. On the semantics of fuzzy logic. *Int. J. of Approximate Reasoning*, 5:45–88, 1991.
34. E. H. Ruspini. Truth as utility: A conceptual synthesis. In *Proc. of the Conf. on Uncertainty in Artif. Intell.*, pages 316–322, Los Angeles, CA, 1991.
35. A. Saffiotti. Pick-up what? In C. Bäckström and E. Sandewall, editors, *Current Trends in AI Planning — Procs. of EWSP '93*, pages 166–177. IOS Press, Amsterdam, Nederlands, 1994.
36. A. Saffiotti, K. Konolige, and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.
37. A. Saffiotti, E. H. Ruspini, and K. Konolige. Blending reactivity and goal-directedness in a fuzzy controller. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 134–139, San Francisco, California, 1993. IEEE Press.
38. A. Saffiotti and L. P. Wesley. Perception-based self-localization using fuzzy locations. In L. Dorst, M. van Lambalgen, and F. Voorbraak, editors, *Reasoning with Uncertainty in Robotics*, number 1093 in LNAI, pages 368–385. Springer-Verlag, Berlin, DE, 1996.
39. A. Saffiotti (maintainer). Fuzzy logic in the autonomous mobile robot Flakey: on-line bibliography. http://aass.oru.se/People/Saffiotti/flakeybib.html. Also ftp://aass.oru.se/pub/saffiotti/robot/.
40. A. Saffiotti (maintainer). Using fuzzy logic in autonomous robotics: web resource collection. http://aass.oru.se/Events/FLAR/index.html.
41. M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Procs. of the Int. Joint Conf. on Artificial Intelligence*, pages 1039–1046, 1987.
42. R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *Int. J. of Robotics Research*, 5(4):56–68, 1986.
43. M. Sugeno and M. Nishida. Fuzzy control of model car. *Fuzzy Sets and Systems*, 16:103–113, 1985.
44. H. Surmann, J. Huser, and L. Peters. A fuzzy system for indoor mobile robot navigation. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 83–86, Yokohama, JP, 1995. IEEE Press.
45. E. Tunstel, H. Danny, T. Lippincott, and M. Jamshidi. Autonomous navigation using an adaptive hierarchy of multiple fuzzy behaviors. In *Proc. of the IEEE Int. Sym. on Computational Intelligence in Robotics and Automation*, Monterey, CA, 1997.

46. C. Voudouris, P. Chernett, C. J. Wang, and V. L. Callaghan. Hierarchical behavioural control for autonomous vehicles. In A. Halme and K. Koskinen, editors, *Proc. of the 2nd IFAC Conf. on Intelligent Autonomous Vehicles*, pages 267–272, Helsinki, FI, 1995.
47. J. Yen and N. Pfluger. A fuzzy logic based robot navigation system. In *Procs. of the AAAI Fall Symposium on Mobile Robot Navigation*, pages 195–199, Boston, MA, 1992.
48. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
49. L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.