History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots

Appears in *International Conference on Tools with Artificial Intelligence, pages 312-319, 2000*

Khaled Ben Lamine and Froduald Kabanza
Dépt. de math et info
Université de Sherbrooke
Sherbrooke, QC J1K 2R1 Canada
{benlamin,kabanza}@dmi.usherb.ca

Abstract

Behavior-based robot control systems have shown remarkable success for controlling robots evolving in real world environments. However they can fail in different manners due to their distributed control and their local decision making. In this case, monitoring can be used to detect failures and help to recover from them. In this work, we present an approach for specifying monitoring knowledge and a method for using this knowledge to detect failures. In particular we show how temporal fuzzy logic can be used to represent monitoring knowledge and then utilized to effectively detect runtime failures. New semantics are introduced to take into consideration uncertainty and noisy information. There are numbers of advantages to our approach including a declarative semantics for the monitoring knowledge and an independence of this knowledge from the implementation details of the control system. Moreover we show how our system can deal effectively with noisy information and sensor readings. Experiments with two real-world robots and the simulator are used to illustrate failure examples and the benefits of failure detection and noise elimination.

1. Introduction

Some key issues in the design of robot control architectures are whether the architecture should be centralized or distributed, whether the reasoning should be deliberative or reactive, whether there is a global model of the world and whether the control should be top-down or bottom-up.

In one end of the spectrum deliberative architectures operate in a sense, plan, act open-loop style. Sequences of actions emerge from the interplay of the planner, the given goals, and the world model constructed from the sensory data. Actions are dictated by a generated plan. They depend on the information available at the planning time. Thus discrepancies between the "real world" at execution time and the internal "world model" used at planning time might cause the robot plan to fail. Such discrepancies are caused by unexpected events, or contingencies and can be handled at plan generation, by anticipating them or through monitoring of the execution of the system [1, 7, 3].

In the other end, behavior-based approaches [2] operate in a situation-action style. In this case actions are triggered by the perceptual conditions. There is no explicit representation of goals, plans or internal "world model". Goals are implicit in the condition-action coupling and plans emerge in real-time as one action is executed in a way that it triggers another one. As actions are dictated by local information sensed, failures occur when the local decisions don't go toward the achievement of the overall goal. In deliberative approaches progress toward the goal is guaranteed as long as there isn't discrepancies between the real world and the internal model. This is not the case for behavior based approaches where sensory information are frequently updated. Then, the robot can get trapped in a local minimum or wander indefinitely in a region formed by obstacles. Failures can be handled, in these cases, by introducing a short memory [12] or changing the local configuration of the environment [17].

These failure detection techniques rely on heuristic monitoring of robot's behaviors to detect potential failures. By "heuristic", we mean that there is no well-defined semantic behind the verification method. These methods rather rely on rules of thumbs and handle failures in an ad-hoc fashion. While this effectively helps in detecting some failures, it is often difficult to analyze and understand the range of typical failures covered by heuristic monitoring strategies.

Actually we need a language to specify failure conditions and facilities to monitor them. The work of Jahanian *et al.* [8] is particularly interesting. Real-time conditions to be monitored and verified are specified using a temporal logic called Real Time Logic (RTL). This is a logic of events. Timing conditions are specified in terms of starting time and ending time of relevant events. The evaluation of these conditions is made over a runtime trace gathered during the system execution. Another related approach was proposed by Felder and Morzenti [5].

An another problem facing real world robot's monitoring systems is uncertainty coming from the complexity of the environment itself, from noisy sensors, or from imprecise actuators. According to [14] there is three ways to cope with uncertainty.

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 recover from errors; or

3. Reason about it, by using techniques for the representation and the manipulation of uncertain information.

The approach we advocate here is in the same line of inquiry of Jahanian *et al.* work, but is more tailored for behavior-based robots. First of all, we use a temporal fuzzy logic to account for noisy information, uncertainty, and fuzzy behaviors in the SAPHIRA mobile robot architecture. On a more technical level, we use a state-based logic. Basic propositions in our logic relates to states rather than to events. Accordingly, our approach for checking conditions specified in that logic is different. We use an incremental method that can do the verification *on the fly*. This method is inspired from [9], where a similar approach was used to generate plans by verifying temporal logic goals over simulated traces of predicted execution sequences. An earlier version of our approach was presented in [10]. In this paper we extend the experimental setting to show how noise is eliminated. Different models of noise are studied and we show the effectiveness of our approach for filtering noise. Also some insights on the algorithmic complexity are discussed.

The remainder of this paper is organized as follows. In the next section, we define our new temporal fuzzy logic. then we give some examples of failure specifications. This is followed with a description of the algorithm used to monitor and check the violation of behavioral properties expressed in that logic. We also discuss the complexity of the algorithm. Finally some empirical results are presented. In particular, we show how we can deal with noisy information.

2. Monitoring knowledge specification

To specify properties that must be monitored, we need a language of state sequences. Linear temporal formulas have been used successfully for specifying such properties for the purpose of verifying concurrent systems [6]. Formulas in such logics are interpreted over models that are infinite sequences of states and temporal modalities are used to assert properties of these sequences.

In our case, we also use linear temporal logic formulas, but with a fuzzy semantics. The basic building blocs of our temporal logic are fuzzy propositions. They specify facts about states. The truth of a proposition has a real-value over the interval [0,1] indicating the degree of truth. Temporal connectors are then involved to specify properties of sequences of states. It is important to notice that we are dealing here with degrees of truth and not with degrees of uncertainty [4]. For instance we may have a proposition $VisibleBall$ stating that some ball of interest is visible to the robot. In practice, robotics vision is noisy so that it is often hard to determine whether the ball is visible or not. The graded truth value of the proposition $VisibleBall$ allows us to include fuzzy statements such as "slightly visible" or "completely visible". In the case of uncertainty reasoning, the real number between 0 and 1 would reflect the possibility that the ball is visible or not visible. The greater the value is the greater the possibility that the ball is visible.

In reality we have both situations. While fuzzy propositions take a graded truth values they are also uncertain. This is in accordance with the facts that some state variables have values obtained from uncertain readings caused by intrinsic errors, communication disturbance or computation limitations (e.g., in image recognition). Uncertainty can be handled using possibility or probability theory.

In our approach, instead of evaluating a proposition from just one snapshot, we observe its truth value on a whole period and conclude its truthness based on snapshots taken during that period. To allow this, our propositions are evaluated over segments of state sequences rather than over a single state. The size of the segment is determined empirically. When the segment has only one state, then this is equivalent to a traditional state-based evaluation. In theory, it is possible to attain the same result by a temporal formula rather than a single proposition, but this would not be efficient. In this case, the truth value of the formula would be obtained after some logical entailment, while in our case, it is obtained from a single function evaluation.

2.1. Syntax

Our fuzzy temporal formulas are constructed from an enumerable collection of propositions; Boolean connectives $\wedge$ (and), $\neg$ (not), $\vee$ (or), $\rightarrow$ (implies), and the temporal con-

nectives $\bigcirc$ (next), $\Box$ (always), $\Diamond$ (eventually), U (until). The formulas formation rules are:

- every fuzzy proposition p is a formula and
- if f_1 and f_2 are formulas, then so are $\neg f_1$, $f_1 \wedge f_2$,$f_1 \vee f_2$, $f_1 \rightarrow f_2$, $\bigcirc f_1$, $\Box f_1$, $\Diamond f_1$, and $f_1 \, U \, f_2$.

In addition to these basic rules the language contains a set of atomic fuzzy propositions $\{\perp_{0.0}, \ldots, \perp_{1.0}\}$, where $\perp_{0.0}$ denotes statements "completely" *false* and $\perp_{1.0}$ denotes statements "completely" *true*.

2.2. Semantics

Formulas are interpreted over models of the form $\langle w, \pi, \Pi \rangle$, where:

- w is an infinite sequence of worlds state $w_0, w_1, \ldots$;
- Π a set of real-valued functions that evaluate propositions p_k in world states. $\pi_j(p_k, w_i)$ returns the truth value in $[0, 1]$ of proposition p_k in the world state w_i, $\pi_j \in \Pi$. Thus, the truth value of a proposition usually depend on a state. But atomic fuzzy propositions $\perp_k$ always have the same value regardless of the state. Thus, $\pi(\perp_{0.5}, w_i)$ always yields 0.5.
- π is a real-valued function that evaluate formulas in world states. $\pi(f, w_i, \Pi)$ returns the truth value in $[0, 1]$ of formula f in the world state w_i.

For a state w_i in a model $M = \langle w, \pi, \Pi \rangle$, proposition p, or formulas f, f_1, and f_2 :

- $\pi(\perp_k, w_i, \Pi) = k$; $k \in [0, 1]$
- $\pi(p, w_i, \Pi) = \pi_k(p, w_i)$; $\pi_k \in \Pi$
- $\pi(\neg p, w_i, \Pi)$ = $1 - \pi(p, w_i, \Pi)$
- $\pi(f_1 \wedge f_2, w_i, \Pi)$ = $\pi(f_1, w_i, \Pi) \otimes \pi(f_2, w_i, \Pi)$
- $\pi(f_1 \vee f_2, w_i, \Pi)$ = $\pi(f_1, w_i, \Pi) \oplus \pi(f_2, w_i, \Pi)$
- $\pi(f_1 \rightarrow f_2, w_i, \Pi)$ = $\pi(f_2, w_i, \Pi) \oslash \pi(f_1, w_i, \Pi)$
- $\pi(\bigcirc f, w_i, \Pi)$ = $\pi(f, w_{i+1}, \Pi)$
- $\pi(\Box f, w_i, \Pi) = \pi(f, w_i, \Pi) \otimes \pi(\Box f, w_{i+1}, \Pi))$
- $\pi(\Diamond f, w_i, \Pi) = \pi(f, w_i, \Pi) \oplus \pi(\Diamond f, w_{i+1}, \Pi)$
- $\pi(f_1 \, U \, f_2, w_i, \Pi) = \pi(f_2, w_i, \Pi) \oplus ((\pi(f_1, w_i, \Pi) \otimes \pi(f_1 \, U \, f_2, w_{i+1}, \Pi))$

where $x \otimes y$ is the minimum of x and y, that is, the fuzzy counter-part of and binary logic connective; $x \oplus y$ is the maximum of x and y, that is, the fuzzy counter-part of or binary logic connective; $x \oslash y$ is the maximum of $(1 - y)$ and x, that is the fuzzy counter-part of $\rightarrow$ binary logic connective. [1]

The function $\pi(p, w_i, \Pi)$ returns the truth value of a proposition p at a given state w_i in a runtime trace. This truth value not only depends on the state w_i, but on a subsequence ending at w_i. The length of the subsequence and the interpretation mechanism are implicit in the user-defined proposition evaluation functions. Thus, for propositions, π invokes user-defined proposition evaluation functions. For instance, assume p is the proposition $VisibleBall$. We define a function that will evaluate p to a value that depends on how the vision system sees the ball on each of the latest 4 states. We then take an "average" value over these states. Here, the number 4 is set empirically.

More formally, following Yager's approach [18], we use *ordered weighted average* (OWA) operators to evaluate the truth value of propositions over histories.

Definition 1 *An OWA operator of dimension n is a mapping F from $[0, 1]^n$ to $[0, 1]$ associated with a wieghting vector $W = [W_1, W_2, \ldots, W_n]$, such that*

1. $W_i \in [0, 1]$
2. $\sum_i W_i = 1$

and

$$F(a_1, a_2, \ldots, a_n) = W_1 b_1 + W_2 b_2 + \ldots W_n b_n$$

where b_i is the ith largest element in the collection $a_1, a_2, \ldots, a_n$

Different OWA operators can be defined depending on the weighting vector. For example $[1, 0, 0 \ldots]$ represent the max operator, $[0, 0, \ldots, 1]$ represent the min operator and $[1/n, 1/n, \ldots, 1/n]$ represent the average operator.

We associate with each proposition a specific OWA so that the evaluation of a proposition corresponds to an "or-anding" of the truth values over a recent state history. Thus, we have:

$\pi_k(p, w_i) = F(\pi_s(p, w_{i_1}), \pi_s(p, w_{i_2}), \ldots, \pi_s(p, w_{i_n}))$; $\pi_k \in \Pi$

Here π_s is a real-valued function that returns the value of a proposition based on a single world state. The weights of the OWA and the extent of the history needed to evaluate a

[1] In general, fuzzy logic may use different definitions of and, or and $\rightarrow$. It is generally required that $\otimes$ be any continuous triangular norm or *t-norm*, with quasi-inverse $\oslash$, and $\oplus$ is any continuous *t-conorm*. The definitions we have adopted satisfy those conditions and are among those most frequently used.

proposition are defined empirically depending on the application and the properties being expressed by propositions. Automated learning of such parameters is also an interesting research topic [13].

Since the evaluation of a formula yields a real-valued value, instead of true or false, we have degrees of truthness or conversely, degrees of falsity. Nevertheless, assuming some empirical threshold value (e.g., false for values below 0.5 and true otherwise), we can talk about a property being violated or being satisfied.

Formulas in our logic have a future semantics but are evaluated on trace that represents the history. This is coherent with the fact that, a formula is used to express the desirable future behaviors of the system, but then the system will check it against its runtime trace to determine whether or not the current execution is effectively consistent with the formula.

3. Examples of specifications

Our experiments are being conducted with an *Activmedia* Pioneer I mobile robot and Pioneer AT mobile robot, both equipped with seven sonars used to perceive obstacles, a Fast Track Vision system from Newton Labs used to perceive colored object and a gripper used for grasping objects.

The sensors and actuators suffer from noisy reading and uncertainty. For example, variation of the light intensity can affect precision in seeing colored object, and wheel slippage can affect precision in measured travel distances.

In addition, the robot is controlled over a radio modem link which can suffer from environment disturbance.

One of the tasks that we have experimented consists in searching for a red ball and bringing it to a home location marked by green. The green location have to be localized too. We programmed this tasks, by decomposing it into two subtasks: searching and homing. The searching subtask includes searching for the red ball, approaching it, and then grasping it. The homing subtask includes searching for the home location, approaching it, and then releasing the red ball.

In our early tests we noted some failures conditions. For example, when searching for the red ball, the ball may become visible only for a brief period of time in the visual field of the camera, for example because the robot's vision angle becomes obstructed by an obstacle. In such a situation, the robot should not consider that it has found the ball to begin approaching it. Another failure situation is when the ball is in a corner the robot cannot reach it. This may cause a stall if the robot commits to its goal and persists in trying to grasp the ball. To capture such failures situations, we use fuzzy temporal logic formulas to express contextual properties under which robotics behaviors must operate. Here are some examples:

1. Context failure
   - $\Box(ApproachingBall \rightarrow VisibleBall)$ when approaching the ball it must remain on the visual field of the camera
   - $\Box(ApproachingHome \rightarrow (VisibleHome \wedge Ballgrasped))$ when approaching home it must remain on the visual field of the camera and the ball must be held on.
2. Goal failure
   - $\Box \Diamond(SearchingBall \wedge Visibleball)$ when searching the ball it has to be visible for a period of time to be considered found.
   - $\Box \Diamond(GettingBall \wedge Graspedball)$ when grasping the ball it has to be be reached.
3. Stall failure
   - $\neg \Diamond \Box(ActionSumNull \wedge ActionStopNull)$ If this property is violated, this means that we have a stall. That is, a stall occurs when the summation of the behavior suggested actions is null and the stop behavior (used to stop the robot when there is no action suggested) is not active.
4. Sequencing failure
   - $\neg \Box(AproachingBall\, U\ SearchingBall)$ Approaching the ball should not, always, promote searching for the ball.

In the above examples we recognize safety temporal properties as well as different classes of progress properties as defined in [11]. For example the stall correspond to a persistence formula ($\Diamond \Box P$).

4. Progression algorithm

Our algorithm is an extension of the *formula progression algorithm* from [9] to handle our fuzzy semantics. As in [9], a formula is verified over a sequence of states (a runtime trace in our case) by *progressing* it on the fly over the trace. More specifically, this means that each state is a labelled with a formula that must be evaluated by each sequence starting from this state. Given any state and its label, the label of a successor in the state history is obtained by applying the algorithm described in Fig. 1.

The input of the formula progression algorithm is a formula f, a state w_i, and set Π of evaluations functions π_i that evaluates a proposition p_i in states. The evaluation functions are OWA operators defined for each proposition. The output is a formula that, when evaluated over a sequence

from w_{i+1} it has the same value as the evaluation of the input formula over the sequence w_i. This algorithm satisfies the following theorem.

Theorem 1 *Let $w_1, w_2, \ldots$ denote any infinite sequence of world states, Π a set of evaluation functions that evaluate propositions in states. Then for any state w_i and a formula f, $\pi(f, w_i, \Pi) = \pi(Progress\_formula(f, w_i, \Pi), w_{i+1}, \Pi)$.*

The proof is based on the observation that the algorithm is merely a rewriting of the formula interpretation rules given in section 2.2.

Progress_Formula(f, w_i, Π)

1. case f
2. p_i (p_i a proposition): return $\bot_{\pi_i(p_i, w_i)}$ where $\pi_i \in \Pi$
3. $\neg f_1$: $\neg$ Progress_Formula(f, w_i, Π)
4. $f_1 \wedge f_2$: Progress_Formula$(f_1, w_i, \Pi) \wedge$ Progress_Formula(f_2, w_i, Π)
5. $f_1 \vee f_2$: Progress_Formula $(f_1, w_i, \Pi) \vee$ Progress_Formula(f_2, w_i, Π)
6. $f_1 \rightarrow f_2$: Progress_Formula $(f_1, w_i, \Pi) \rightarrow$ Progress_Formula(f_2, w_i, Π)
7. $\bigcirc f_1 : f_1$
8. $\Diamond f_1$: Progress_Formula$(f_1, w_i, \Pi) \vee \Diamond f_1$
9. $\Box f_1$: Progress_Formula$(f_1, w_i, \Pi) \wedge \Box f_1$
10. $f_1 \, U \, f_2$: Progress_Formula$(f_2, w_i, \Pi) \vee$ (Progress_Formula$(f_1, w_i, \Pi) \wedge f_1 \, U \, f_2$)

Figure 1. Formula progression algorithm

This theorem is in turn the basis of our temporal checker. The basic process consists in progressing the formula over the runtime trace. That way, each new state added to the current trace obtains a formula label that is computed by the above formula progression algorithm. The theorem implicitly states that a state where the formula is "made false" (more precisely, its value is below an empirically set threshold) violates the temporal property expressed by the original formula. However, progressing formulas over infinite sequences is not suitable for robotic applications where some timing constraints can be involved. For this reason, when implementing the progress algorithm formulas are evaluated only on specific context. For example, the formula $\Box(ApproachingBall \rightarrow VisibleBall)$ is effective only when the robot is approaching the ball so that it does not have to be evaluated in any other context. Also, we can associate a time out to formulas. The progress algorithm will, in this case, return false when the formula is evaluated to false or when it is timed out.

The progression algorithm also uses Boolean simplification on the intermediate results. The following simplification rules are applied at various steps of the algorithm where $0 \leq M, N \leq 1$ represents user defined thresholds, for True (N) and False (M).

- $(\bot_k \wedge f | f \wedge \bot_k) \rightarrow \bot_k$ if $k \leq M$
- $(\bot_k \wedge f | f \wedge \bot_k) \rightarrow f$ if $k \geq N$
- $\neg \bot_k \rightarrow \bot_{1-k}$

These simplifications are used for efficiency purpose. Some recursive calls of the progress algorithm are avoided because of these transformations.. For example, when one of the conjunct of an $\wedge$ connective evaluate to $\bot_k$ where $k \leq M$ i.e. "$False$" all the formula is made$\bot_k$ i.e. "$False$".

4.1. Examples of formulas progression

Assume we want to check one of the example formulas above, namely: $\Box(ApproachingBall \rightarrow VisibleBall)$. For this, let's use the weight vector $(0.0, 0.5, 0.5, 0.0)$ for the OWA operator associated to the propositions $ApproachingBall$ and $VisibleBall$. For example if the truth values of $VisibleBall$ over the 4 last trace states $S_1 \ldots S_4$ is $(1.0, 1.0, 1.0, 0.0)$ then when evaluating the proposition $VisibleBall$ in the state S_4 we have

$$\begin{aligned}\pi(VisibleBall, S_4) &= 0.0 \times 1.0 + 0.5 \times 1.0 + 0.5 \times 1.0 \\ &\quad + 0.0 \times 0.0 \\ &= 1.0.\end{aligned}$$

That is, the new value of $VisibleBall$ in state $S4$ is 1.0 instead of 0.0 when we evaluate it in a single state.

With this trace and given the formula $\Box(ApproachingBall \rightarrow VisibleBall)$, i.e. when approaching a ball it must be visible, the progress algorithm produces the formula $(\bot_{1.0} \wedge \Box(ApproachingBall \rightarrow VisibleBall)) \equiv \Box(ApproachingBall \rightarrow VisibleBall)$ in all states.

Figure 2 shows the progression of formula f_0 in a state S_i. The dotted arrows represent logical simplifications and the solid ones correspond to steps from the algorithm given in Figure 1.

4.2. Complexity of the progression algorithm

The complexity of the progression algorithm lies with the repeated progression of a formula through a sequence

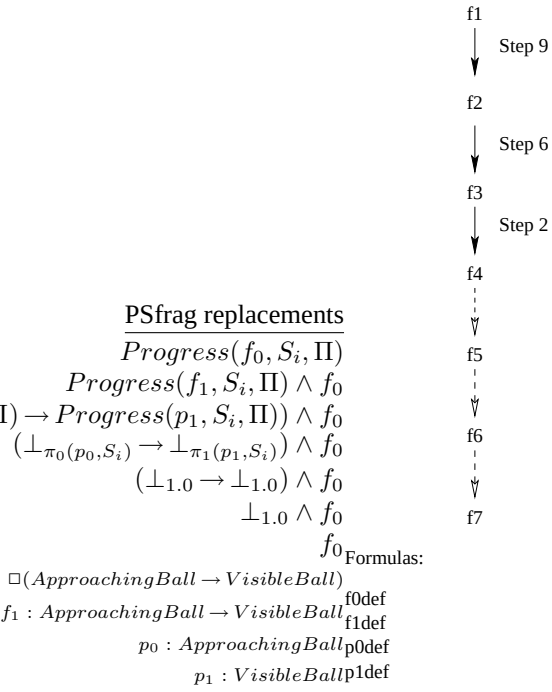

Figure 2. Example of formula progression

of states. During monitoring we have to progress the original temporal formula through every state w_i. This formula might grow in length with each progression, which can lead to growing space and time requirement. For example consider the formula $\Box \Diamond P$ progressed through a world state where P does not hold. The progression algorithm yields the new formula, $\Diamond P \wedge \Box \Diamond P$ which is twice the size of the original formula.

However, the progressed formula has some sub-formulas in common with the original one ("$\Diamond P$"). Hence, we can gain space by sharing the data structure of the sub-formulas. In fact, in this example we only need to add a new "$\wedge$" at the top level. Furthermore, not all formulas continue expanding. Some formulas will generate conditions that have to be evaluated only in the next state. So their length will diminish. For example if we progress the formula $\Box(P \rightarrow \bigcirc Q)$ through a world w in which P holds, the result will be the formula $\Box(P \rightarrow \bigcirc Q) \wedge Q$. That is, the always test is propagated to the next state, and in addition the next state will be required to satisfy Q since P is "true" in w. On the other hand, if P is "false" then the progressed formula would simply be $\Box(P \rightarrow \bigcirc Q)$. That is, there is no requirements on the next state, we simply propagate the formula.

The progress algorithm evaluates formulas in states. So it involves evaluating OWA operators. The evaluation in this case implies sorting the values of the propositions in different states and then computing the OWA average according to the given weight vector. Since the dimension of the OWA weight vector is not high the evaluation is generally efficient. Finally, the different state values that the monitoring system receives from the robot control system might involve some calculated functions. But the monitoring system have no control over these functions.

5. Empirical results

We have implemented a monitoring system (see figure 3) to collect traces, to check for potential failures specified by temporal fuzzy logic formulas, and to notify the SAPHIRA control program of those detected failures. The general structure of the monitoring system consists of: (1) a monitoring knowledge base containing models of the expected/unexpected ideal behavior of the robot in terms of temporal constraints; (2) a trace collector for tracking the actual behavior of the system; (3) a failure detection module for checking a temporal fuzzy logic formula over the current trace to determine whether or not it violates the formula; and (4) a failure diagnosis module that evaluates a trace of the temporal logic verification process to determine the type of failure in a format that is meaningful to the robot control system. In our actual implementation the trace collector is reduced to simply passing inputs to the failure detection component. The failure diagnosis component is also reduced to simply passing notifications to the robot control system. Ideally, each component of the monitoring system can be viewed as an intelligent agent. For example, the trace collector agent can decide which trace to collect and the length of the trace. The diagnosis agent can analyze the failure and suggests the appropriate recovery strategy.

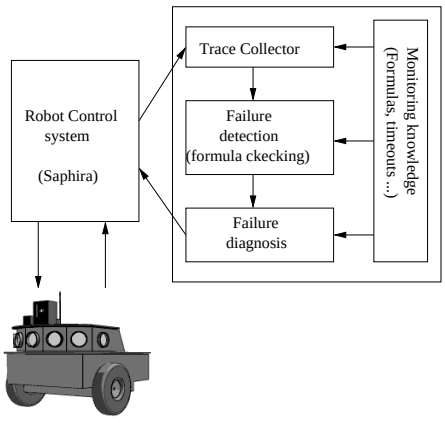

Figure 3. The general structure of the monitoring system

5.1. Noise filtering

In this part we aim at evaluating our detection algorithm with regards to noise tolerance. Noise can be introduced by a variety of sources. The sensors and actuators are the primary source of noise. They can add random values to the actual ones. Noise can also appear from communication disturbance or malfunctioning. In typical traces the noise can be modelled with either a salt-and-pepper or a gaussian distribution. Salt-and-pepper noise is most often used to model communication disturbance or a malfunctioning sensor. The gaussian noise is most often used to model natural noise processes such as those occurring from electronic noise in the sensors or actuators systems. The shape of these noise types as a function of the fuzzy truth value level can be modelled as a histogram. For example, there are only two possible values of noise, a and b, for the salt-and-pepper noise type and the probability of each is A and B. The typical value for pepper-noise is 1 and for salt-noise is 0. Noise is well studied in image processing systems [16]. It can be removed using spatial filters (filters based on a small neighborhood of pixels). Two types of filters are used, order filters and mean filters. Order filters use the ordering of neighborhood pixel gray values to select the "correct" value. Mean filters use an "average" value to compute the correct one. Order filters are suitable to remove salt-pepper noise while mean filters are used for gaussian noise. We can also define hybrid filters for the combination of the two types of noises.

In our case OWA operators act as a filter to remove noise (uncertainty) from a trace. Different type of filters can be defined using the OWA operator with different weight vectors. For example to eliminate the salt or pepper noise we can use the $[1, 0, \ldots, 0]$ (max) , and the $[0, 0, \ldots, 1]$ (min) vector respectively. To eliminate the gaussian noise we use the $[1/n, 1/n, \ldots, 1/n]$ corresponding to the arithmetic mean filter, and to eliminate both we use the weight vector $[0, 1/(n-2), \ldots, 1/(n-2), 0]$ which is equivalent to the alpha-trimmed mean filter type in image processing.

The task used in the experiment is to track and approach a moving goal in a simulated environment. The goal position moves between two point and the robot can "see" the goal position only if it lies within a certain angle ahead of it. We add different type of noises to the trace and we note the percentage of failures (the robot can't track the goal position). The tests are conducted using three increasing speed for the moving goal position (S1, S2, and S3) and with the OWA operator equivalent to the alpha-trimmed mean filter. Tables 1, 2, and 3 show the results. The results are straightforward, the use of OWA operator as a noise filter can eliminate noise of the different types. But when the noise become dominant the filter can't eliminate all the noise.

	Salt-pepper			
	0.0	0.05	0.1	0.2
S1	1	1	0.95	0.5
S2	1	1	0.95	0.48
S2	0.75	0.75	0.64	0.4

Table 1. Salt-pepper noise

	Gaussian			
	μ 0, σ 0.0	μ 0, σ 0.2	μ 0, σ 0.4	μ 0, σ 0.6
S1	1	1	0.9	0.8
S2	1	1	0.9	0.8
S2	0.75	0.75	0.7	0.6

Table 2. Gaussian noise

5.2. Failure coverage

In this part we aim at evaluating our approach in terms of failure coverage. We tested our system in real world environment. We have written one control program that contains no monitoring processes (*Prog1*) and two control programs that use our monitoring system (*Prog2, Prog3*).

Prog1 contains no monitoring processes. It serves as a reference program with regards to the robustness of the control system. It also gives an idea about the environment conditions. In fact, we use basic behaviors for writing the program. These behaviors can fail when the conditions they are designed for are no longer valid.

Both *Prog2* and *Prog3* use our monitoring system sending information and receiving failures notifications from it. Monitoring processes are replaced by fuzzy temporal formulas that are checked online by our monitoring system. We also added a communication process to send trace histories and receive failure notifications. *Prog3* uses noise filtering while in *Prog2* no noise filtering is used. There is no recovery from failures in either program.

We conducted 30 runs of each program and noted the number of failure notifications and the number of run failures. The number of run failures is high in part due to the fact that there is no monitoring facilities and because some failures are intentionally introduced. The number of notifications is the same as the number of failure because no recovery strategies are used. The experiments (Table 4)

	Gauss. + Salt-pep.		
	0.05, μ 0, σ 0.2	0.1,μ 0, σ 0.4	0.2,μ 0, σ 0.6
S1	1	0.85	0.75
S2	1	0.85	0.75
S2	0.75	0.70	0.58

Table 3. Salt-pepper + gaussian noise

demontrate that when using noise filtering the system performs better while detecting less failures. This is because noise is considered as a failure condition in *Prog2* while ignored in *Prog3*.

	Prog 1	Prog 2	Prog 3
notifications	0	22	15
failures	20	22	15
runs	30	30	30

Table 4. Empirical results

6. Conclusion

Monitoring is the process of recording event occurrences during program execution in order to gain runtime information about the system states as well as information about the environment in which the system evolves [15, 8]. Therefore, when the environment is uncertain and the collected information is noisy any monitoring solution have to deal with these issues. In this paper we presented a formal tool for monitoring behavior based robot control systems which explicitly deals with such conditions. In particular we demonstrated how OWA operators are used to filter noise from the collected traces. Empirical results show the effectiveness of the approach with different types of noises. However this effectiveness holds to the fine tuning of the OWA operators weight vector and its dimension. In this case, adaptive noise filter might be used to alter its basic behavior depending on the local statistics of the collected traces. Adaptive filters are also used in image processing but they are more tailored to image data and incur more computation. A detailed study of the statistic properties of typical traces have to be done. We also assumed that the noise is mainly salt-pepper and gaussian. While this assumption seems to hold in our real environment tests, further investigation is necessary to determine the noise model in different environments and with different robot systems.

References

[1] J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 735–740, St. Paul, Minnesota, 21–26 Aug. 1988. Morgan Kaufmann.

[2] R. C. Arkin. *Behavior-Based Robotics*. MIT press, 1998.

[3] M. Beetz and D. McDermott. Improving robot plan during their execution. In *Proc. Second International Conference on AI Planning Systems*, pages 3–12, 1994.

[4] D. Dubois and H. Prade. Partial truth is not uncertainty: Fuzzy logic versus possibilistic logic. *IEEE Expert*, 9(4):15–19, August 1994.

[5] M. Felder and A. Morzenti. Validating real-time systems by history checking trio specifications. *ACM Transactions on Software Engineering and Methodology*, 3(4), October 1994.

[6] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15th Work. Protocol Specification, Testing, and Verification*, Warsaw, June 1995. North-Holland.

[7] G. Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 453–464, 1998.

[8] F. Jahanian, R. Rajkumar, and S. C. V. Raju. Run-time monitoring of timing constraints in distributed real-time systems. *Real-Time Systems Journal*, 7(3):247–273, 1994.

[9] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.

[10] K. B. Lamine and F. Kabanza. Using temporal fuzzy logic for monitoring behavior based mobile robots. In *Proc. of the IASTED conference in Robotics and Applications*, pages 116–121, Hawaii, August 2000.

[11] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag: Heidelberg, Germany, 1992.

[12] F. Pin and S. Bender. Adding memory processing behaviors to the fuzzy Behaviorist-based navigation of mobile robots. In *ISRAM'96 Sixth International Symposium on Robotics and Manufacturing*, Montpelier, France, May 27-30 1996.

[13] R. R.Yager and D. Filev. On the issue of obtaining owa operator weights. *Fuzzy Sets and Systems*, 94:157–169, 1998.

[14] A. Saffiotti. Handling uncertainty in control of autonomous robots. In *Applications of Uncertainty Formalisms in Information*, pages 198–224. Lecture Notes in Computer Science, Vol. 1455, 1998.

[15] J. P. J. Tsai and S. J. H. Yang. *Monitoring and Debugging of Distributed Real-Time Systems*. IEEE Computer Society Press, 1995.

[16] S. E. Umbaugh. *Computer vision and image processing : a practical approach using CVIPtools*. Prentice Hall, 1998.

[17] W. Xu. A virtual target approach for resolving the limit cycle problem in navigation of a fuzzy behaviour-based moblile robot. *Robotics and Autonomous Systems*, 30:315–324, 2000.

[18] R. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man, And Cybernetics*, 18(1):183–190, 1988.