Using fuzzy temporal logic for monitoring behavior-based mobile robots

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

Abstract

This paper presents a model and an implementation of a runtime environment for specifying and monitoring properties of behavior-based robot control systems. The proposed approach supports collecting events that are recorded and examined at run-time. Temporal fuzzy logic is used as a formal language for specifying behaviors properties and new semantics are introduced to take into consideration environment unpredictability and uncertainty. These ideas are developed in the SAPHIRA mobile robot's control environment, but they can also be applied to other behavior-based architectures. Experiments with two real-world robots are used to illustrate failure examples and the benefits of failure detection.

Keywords: mobile robots, behavior-based robotics, temporal reasoning, uncertainty in AI.

1 Introduction

Behavior-based approaches – see for example [1] and SAPHIRA [7]– have shown remarkable success in controlling robots evolving in real world environment. Briefly, in these approaches we remove the non essential assumptions that could prevent from an adaptation to unanticipated events. Also the decision process about the action to take in a given situation is distributed across several simple processes. Typically, such processes, also called behaviors, are implemented as direct mapping from local sensors data to control actions.

Designing concurrent reactive behaviors based on local considerations make them easy to program and debug. However, their combination may cause unpredicted and undesirable results. For instance, there may be places where the desire to reach a goal destination exactly balances the urge to turn away from obstacles, yielding a stall (null move and turn action). In other situations, the combination of behaviors may suggest a turn in a given situation and then a turn in the opposite direction in the next situation, so that the robot oscillates between these two moves. Behaviors can also fail if the context they are designed for is no longer valid. For example, approaching an object may require the object to remain visible for a certain period of time in order to locate it.

Detecting such failures is a non trivial problem, yet a very important one in behavioral approaches. Without the ability to detect anomalies, a robot won't be able to autonomously adjust its behaviors and overcome unpredicted failures. Actually we need a feedback system that incorporates a "progress" criterion into behaviors, and facilities to monitor this criterion.

Current failure detection techniques for mobile robots rely on heuristic monitoring of robot's behaviors to detect potential failures [8, 12]. 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.

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 [10] 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.

In this paper we present a framework for monitoring behavior-based robot control systems. Along with the third way above, we define a fuzzy temporal logic that is used to specify desirable system behaviors. We also provide a method for checking online the violation of these behaviors. There is numerous 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.

In order to fix a context, these ideas are developed in the SAPHIRA [7] mobile robot's control environment, but they can also be applied to other behavior-based architectures such as [3].

The remainder of this paper is organized as follows. In the next section, we discuss related work. We then define our new fuzzy temporal logic. This is followed with a description of the approach used to monitor and check the violation of behavioral properties expressed in that logic. Finally, we present some empirical results before concluding.

2 Related Work

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 [11, 5]. The work of Jahanian *et al.* 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 [2].

The approach we advocate here is in the same line of inquiry, but is more tailored for behavior-based robots. First of all, we use a fuzzy temporal logic to account for fuzzy behaviors in the SAPHIRA mobile robot architecture and noisy, uncertain information. 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 [6], where a similar approach was used to generate plans by verifying temporal logic goals over simulated traces of predicted execution sequences.

Apart from validating the control system, the ultimate goal of monitoring, in the case of behavior-based control, is to make the system more adaptive. In this setting, the monitoring system gives feedback to the robot's decision making processes which can then adapt their control strategies. However, the integration of monitoring and decision making is beyond the scope of the present paper and hence will not be discussed.

3 Temporal properties specification

Linear temporal logic formulas have been used successfully for specifying properties for the purpose of verifying concurrent systems [4]. 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 truth of a proposition is a real value between 0 and 1. For example, the truth value of the proposition $VisibleBall$ will be a real number between 0 and 1 reflecting our incapacity to draw clear boundaries between thruthness and falseness of a proposition. This allows us to include fuzzy statements such as "slightly visible" or "completely visible". On the other hand, it is not wise to conclude that the ball is visible from just one snapshot because of noise inputs. Rather, we should observe the ball on a whole period and conclude that it is visible 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.

3.1 Syntax

Our fuzzy temporal formulas are constructed from an enumerable collection of propositions; Boolean connectives $\wedge$ (and), $\neg$ (not), and the temporal connectives $\bigcirc$ (next), $\Box$ (always), $\Diamond$ (eventually), and U (until). The formulas formation rules are:

- every fuzzy proposition p is a formula; in particular, we have built-in static propositions corresponding to real values in $[0, 1]$; for the sake of clarity, the fuzzy propositions corresponding to a real-value x is simply noted x; hence 0.5 and 0.65 are fuzzy propositions;
- if f_1 and f_2 are formulas, then so are $\neg f_1$, $f_1 \wedge f_2$, $\bigcirc f_1$, $\Box f_1$, $\Diamond f_1$, and $f_1 \, U \, f_2$.

3.2 Semantics

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

- w is an infinite sequence of worlds state $w_0, w_1, \ldots$;
- π is a real-valued function that evaluates propositions in states. For a given proposition p and a state w_i, $\pi(p, w_i)$ returns the truth value in $[0, 1]$ of proposition p in the world state w_i. Thus, the truth value of a proposition usually depends on a state. But built-in static fuzzy propositions always have the same value regardless of the state. Thus, $\pi(0.5, w_i)$ always yields 0.5.

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

- $\pi(\neg p, w_i) = 1 - \pi(p, w_i)$
- $\pi(f_1 \wedge f_2, w_i) = \pi(f_1, w_i) \otimes \pi(f_2, w_i)$
- $\pi(f_1 \vee f_2, w_i) = \pi(f_1, w_i) \oplus \pi(f_2, w_i)$
- $\pi(\bigcirc f, w_i) = \pi(f, w_{i+1})$
- $\pi(\Box f, w_i) = \pi(f, w_i) \otimes \pi(\Box f, w_{i+1}))$
- $\pi(f_1 \, U \, f_2, w_i) =$
  $\pi(f_2, w_i) \oplus ((\pi(f_1, w_i) \otimes \pi(f_1 \, U \, f_2, w_{i+1}))$

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. [1]

The function $\pi(p, w_i)$ 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. Here, the number 4 is set empirically.

More formally, following Yager's approach [13], we use *ordered weighted average* (OWA) operators to evaluate the truth 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 $[1, 0, 0 \ldots]$ represents the max operator, $[0, 0, \ldots, 1]$ represents the min

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

operator and $[1/n, 1/n, \ldots, 1/n]$ represents 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:

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

Where 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 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 [9].

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.

4 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
   - $\Box(SearchingBall \rightarrow \Diamond\ Visibleball)$ when searching the ball it has to be visible for a period of time to be considered found.
   - $\Box(GettingBall \rightarrow \Diamond\ Graspedball)$ when grasping the ball it has to be be reached.
3. Stall failure
   - $\neg\,\Box(ActionSumNull \wedge ActionStopNull)$ 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
   - $\neg\,\Box(AproachingBall\, U \quad SearchingBall)$ Approaching the ball should not, always, promote searching for the ball.

```
Progress_Formula($f, w_i, \pi$)
1. case $f$
2.   $p$ ($p$ a proposition): return $\pi(p, w_i)$
3.   $\neg f_1$: $\neg$ Progress_Formula($f, w_i, \pi$)
4.   $f_1 \wedge f_2$: Progress_Formula($f_1, w_i, \pi$) $\wedge$
     Progress_Formula($f_2, w_i, \pi$)
5.   $f_1 \vee f_2$: Progress_Formula ($f_1, w_i, \pi$) $\vee$
     Progress_Formula($f_2, w_i, \pi$)
6.   $\bigcirc f_1$ : $f_1$
7.   $\Box f_1$ : Progress_Formula($f_1, w_i, \pi$) $\wedge$ $\Box f_1$
8.   $f_1\, U\, f_2$ : 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

5 Temporal Checker

Our temporal checker is an extension of the *formula progression algorithm* from [6] to handle our fuzzy semantics. As in [6], 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 labeled 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 a function π that evaluates propositions in states. The function π invokes the OWA operator 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 an evaluation function that evaluate propositions in states. Then for any state w_i and a formula f, $\pi(f, w_i) = \pi(Progress\_formula(f, w_i, \pi), w_{i+1})$.*

The proof (omitted here due to space limitations) is based on the observation that the algorithm is merely a rewriting of the formula interpretation rules given in section 3.2.

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 formula are evaluated only on specific context. For examples, the formula $\Box(ApproachingBall \to VisibleBall)$ 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.

5.1 Examples of Formula Progression

Assume we want to check one of the example formulas above, namely: $\Box(ApproachingBall \to VisibleBall)$. 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 $VisibleBall$.

Assume the following trace (Figure 2), indicating the truth values for the two different propositions in 90 different states of a runtime trace. With this trace and given the formula $\Box(ApproachingBall \to VisibleBall)$, i.e. when approaching a ball it must be visible, the progress algorithm produces the formula $(1.0 \wedge \Box(ApproachingBall \to VisibleBall))$ in all states. Therefore, the formula is not violated by $trace1$. In the case of the second trace ($trace2$) $VisibleBall$ is false in states $S30$ through $S35$. The progress algorithm returns $(0.0 \wedge \Box(ApproachingBall \to VisibleBall))$ i.e. false, in states subsequent to $S31$. This means that the formula is violated and a recovery strategy must be used.

In the first trace (Trace 1) $VisibleBall$ is false in states $S30$ and $S40$, so the formula should be violated. But using OWA operators for evaluating $VisibleBall$

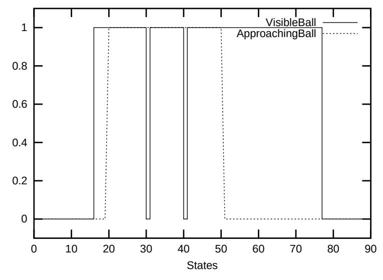


Figure 2: Trace 1

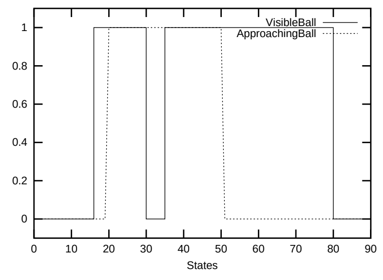


Figure 3: Trace 2

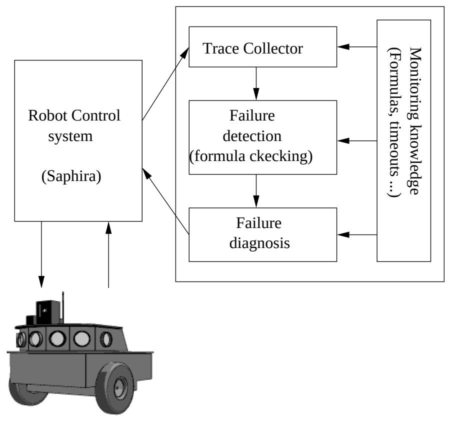
make the value of $VisibleBall$ in states $S30$ and $S40$ true. Actually, we consider the value of $Visibleball$ in states $S30$ and $S40$ as noise and ignore them. In the second example, unlike the first trace, we consider that the robot has lost the visual contact with the ball.

6 Empirical results

We have implemented a monitoring system (see figure 4) 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 this section we aim at evaluating our approach in terms of failure coverage and design effort. We have written two control programs *Prog1* and *Prog2* using the SAPHIRA programming language and a control program *Prog3* communicating with our monitoring system. *Prog1* contains no monitoring processes and serves as a reference program with regards to the robustness of the control system itself. It also gives an idea about the environment conditions. *Prog2* contains ad hoc solutions using the programming language available in SAPHIRA and gives an idea about the complexity of writing monitoring processes. *Prog3* uses our monitoring system, including the temporal checker, sending information and receiving failures notifications from it. We have also written simple recovery strategies to use when failures occur. We focused on three frequent failures: (1) the tracked ball is lost from the visual field of the robot; (2) the ball slips from the gripper; and (3) the ball remains for a short period between the grippers paddles (the robot is equipped with an infrared beam to detect the presence of an object between the paddles of the gripper). Recovery strategies include randomly searching for the ball when we loose visual contact with it, and opening the gripper and going back when the ball slips from the paddles. In addition, when conducting our tests some of the above failures were intentionally provoked.

Table 1 shows the results of our tests. We conducted 30 runs of each program and noted the number of failure notifications (recovery) and the number of run failures. The number of run failures in *Prog1* is high in part due to the fact that there is no monitoring facilities and because some failures are intentionally introduced, like taking out the ball from the gripper or hiding it. *Prog2* and *Prog3* performed better than *Prog1* and *Prog3* was the best.



Trace Collector
Failure detection (formula ckecking)
Failure diagnosis
Monitoring knowledge (Formulas, timeouts ...)
Robot Control system
(Saphira)

Figure 4: The general structure of the monitoring system

Prog3 reported less run failures and less failure notifications than *Prog2*. At first glance it seems surprising

	Prog 1	Prog 2	Prog 3
notifications	0	25	15
failures	20	11	6
runs	30	30	30
design, testing	easy	difficult	medium

Table 1: Empirical results

that detecting less failures leads to a more robust program. However, *Prog2* monitoring processes are in fact too sensitive to failures and treat noisy sensor readings as failure conditions. As a result a frequent switching between recovery and control processes is observed. Using our failure detection approach *Prog3* seems more committed to its current goal a and less prone to noise. The fact is that our failure detection algorithm is based on a state history while *Prog2* failure detection algorithm is based on a snapshot of the current situation to decide that we have a failure. Using OWA operators to evaluate propositions noisy readings are ignored. For example an object is considered visually lost only if it is not detected over a certain period of time. But it can be visible in some cycles of the period considered.

7 Conclusion

Monitoring behavior-based mobile robots evolving in uncertain and noisy environment is a very challenging problem. Since monitoring rely on collecting runtime information of the system and the environment, any monitoring solution have to deal with noisy sensor readings and uncertainty. In this paper we presented a formal tool for monitoring behavior based robot control systems while taking into account uncertainty and noisy information. Our approach have several advantages. First, it provides a declarative semantics for expressing monitoring knowledge. Second it hides implementation details of the monitoring knowledge, and third it provide a high degree of modularity (new monitoring knowledge can be added without affecting the control system). Results showed the effectiveness of our approach for dealing with noise and uncertainty. However, this effectiveness holds to the fine tuning of the OWA operators weight vector. Also, we have assumed that the monitoring knowledge will come from the user just like the other forms of knowledge for controlling the robot. So an important area for future investigation will be to employ learning and reasoning techniques to determine suitable OWA operators given the nature of the environment or to derive monitoring knowledge from the basic behaviors used to control the robot. These techniques can be integrated in the trace collector and the diagnosis component of the system we described in figure 4.

References

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

[2] 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.

[3] E. Gat. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*, volume 2, pages 1622–1627, 1994.

[4] 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.

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

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

[7] 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.

[8] F.G. Pin and S.R. 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.

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

[10] 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.

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

[12] W.L. 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.

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