Model-Free Execution Monitoring in Behavior-Based Mobile Robotics

O. Pettersson **L. Karlsson** **A. Saffiotti**

Center for Applied Autonomous Sensor Systems, Department of Technology, Örebro University, S-70182 Örebro, Sweden
{opn, lkn, asaffio}@tech.oru.se

Abstract

In this paper we present a model-free execution monitor for behavior-based mobile robots. By model-free we mean that the monitoring is based only on the actual execution, without involving any predictive models of the controlled system. Model-free monitors are especially suitable for systems where it is hard to obtain adequate models. In our approach we analyze the activation levels of the different behaviors using a pattern recognition technique. Our model-free execution monitor, which is realized by radial basis function networks, is shown to give a high performance in several realistic simulations.

1 Introduction

In mobile robotics hybrid architectures have become the *de facto* standard [8]. A hybrid architecture might be conceptually divided into two parts: a lower-level sensor driven motor controller, and a higher-level knowledge-based deliberation process, both working in parallel. In a dynamic and uncertain environment, the lower-level part must act fast and robustly. A common solution to this problem is to use a behavior-based approach [2]. But also the higher-level part must be prepared for unexpected events that can cause problems. By introducing an execution monitoring system these problems can be detected. This monitoring system should have at least two functionalities [9]:

- *Fault detection*, that indicates that something is going wrong in the monitored system.
- *Fault isolation*, that makes a classification of what is going wrong.

These systems are often referred to as Fault Detection and Isolation (FDI) systems.

Execution monitors can be classified into two major groups: those that use a predictive model in order to verify that the system evolves as expected, called *model-based monitoring*, and those that solely observe the actual execution of the system, called *model-free monitoring* [9].

The most common approach is model-based execution monitoring. Typically, these systems model actions as state transitions: there is a well-defined state before and after the action. Some examples of model-based approaches to execution monitoring are the early work on PLANEX [7], the Integrated Planning, Execution and Monitoring (IPEM) system [1], the Livingstone system [17], rationale-based monitoring [16], the Task Control Architecture (TCA) [6], and work by Bjäreland [4]. A few researchers have been interested in model-free execution monitoring. For example, Lamine and Kabanza [10] use temporal logic formulas to specify the different failures. These formulas are supplied by the designer. Their approach can be seen as model-free, in the sense that no predictive models are involved.

We are interested in model-free execution monitoring, since in some cases it is difficult to obtain adequate models. This concerns, for example, behavior-based systems. A model-free approach based on learning may also let the system benefit from past experiences, and to adapt to changing conditions. Since model-based execution monitoring has been shown to be very useful in many cases, there is no reason to leave out knowledge that we already have about the system. We claim that a model-free approach could be used as a complement to model-based approaches.

In figure 1 our pattern recognition based model-free execution monitor is shown. Here the two functionalities fault detection and fault isolation are implemented as two radial basis function networks. Our monitor is built up hierarchically; when a fault is detected the fault isolation process is invoked. The monitor has been applied to a robot with a behavior-based control system, and the input data consists of the activation levels of the behaviors.

In the rest of this paper, we provide a brief introduction to our robotic architecture and the use of activation levels, and we describe our methodology: activation level merging, feature extraction, and finally

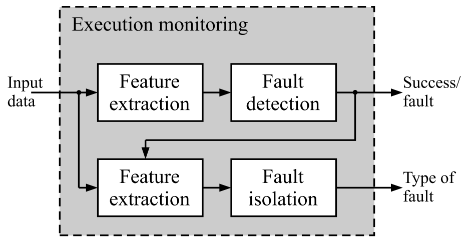

Figure 1: *Model-free execution monitoring.*

classification using radial basis function networks. We also present a series of experiments and an evaluation of the performance of our system, which shows that our approach is feasible.

2 Thinking Cap and Behavior-Plans

We consider execution monitoring on a mobile robot that is controlled by a hybrid architecture evolved from [15] called the Thinking Cap. The Thinking Cap (TC) consists of a fuzzy behavior-based controller, and navigation planning. In figure 2 the TC including execution monitoring is shown. In order to achieve a goal the planner selects a number of behaviors to be executed in different situations. Depending on the current situation, the different behaviors are activated to different degrees.

A system such as the TC, which is based on blending behaviors and where there is no clear notion of where an action starts or ends, is difficult to fit into models where actions are considered as discrete state transitions. On the other hand, the activation levels of the different behaviors are readily accessible, and the patterns they form over time contain valuable information.

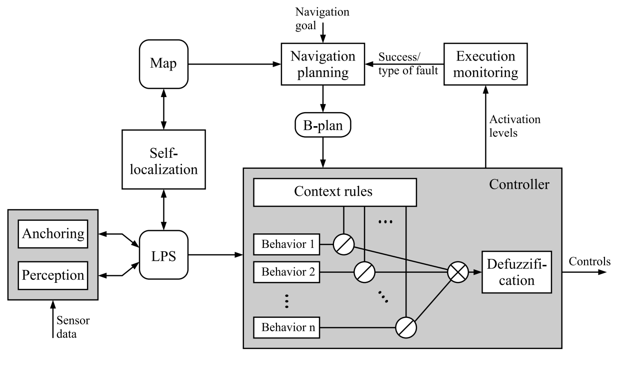

Figure 2: *The Thinking Cap including execution monitoring.*

In the Thinking Cap, a plan, called a "behavioral-plan" or *B-plan*, is represented by a set of context rules associating individual behaviors with the contexts in which they are activated [14]:

$$context \quad \rightarrow \quad behavior$$

For example, the following simplified B-plan could be used to navigate to room *R4* starting from room *R3* and crossing door *D4*:

$$\begin{aligned} at(R3) \land \neg near(D4) &\quad \rightarrow \quad Reach(D4) \\ near(D4) \land \neg facing(D4) &\quad \rightarrow \quad Face(D4) \\ facing(D4) \land \neg at(R4) &\quad \rightarrow \quad Cross(D4) \end{aligned}$$

In each context rule in the B-plan, the *behavior* component is a behavior composed of a set of fuzzy control rules which produces a fuzzy set over control actions, assigning a value in the interval $[0, 1]$ to each control action, indicating how appropriate or desirable that action is. A control action can be for instance a certain acceleration of the robot.

The *context* part is a formula in fuzzy logic [18] which in any given state has a real-valued truth value ranging from 0 (fully false) to 1 (fully true); this value constitutes the activation level of the associated behavior. The formula is evaluated in the internal state of the robot, which contains information from sensors, maps etc. In the B-plan above, for instance, the *Cross(D4)* behavior would have a high activation level in those states when the robot is facing door *D4* and not yet in room *R4*. The context and the behavior are combined with a fuzzy implication operator (quasi-inverse $\oslash$), and results in a new fuzzy set over control actions.

Many behaviors in the B-plan can be active simultaneously, contributing to the control of the robot to different degrees as determined by their contexts. Thus, the output from the different context-behavior-pairs are combined using a fuzzy conjunction (t-norm $\otimes$) such as the *min* operator. Finally, from this combined fuzzy set over control actions, a single control action is then chosen – a process called defuzzification.

In the remainder of the paper, we will focus on the activation levels of behaviors, that is the values obtained from the contexts in the B-plan.

3 Activation Level Analysis

Let us assume that the activation levels traced over time form specific patterns depending of the status of the execution. If this is true, pattern recognition techniques could be used as a basis for the monitoring process. In our case a supervised learning approach to pattern recognition was considered most suitable, since the system designers need to decide what is a fault and what are the type of faults.

The methodology we employ is to train two Radial Basis Function (RBF) networks by running the robot on a number of scenarios where different plans are executed, both with successful results and with various faults. The input data to the RBF networks are features which are extracted from the activation levels. Together with the input data the correct classification, i.e., success or type of fault is presented to the networks. One network is used for fault detection, and one for fault isolation, as shown in figure 1. Different feature extraction methods are used for the two networks. Once the networks have been properly trained, they can be used for online fault detection and isolation.

3.1 Activation Level Merging

Since the B-plans contain different sets of behaviors given different navigation goals, our pattern recognition approach would be severely restricted if it was applied directly on the activation levels of individual behavior instances. For example, different plans may contain different instances of the behavior *Cross* involving different doors. Hence we need to find a way to transform the plan dependent activation levels into a plan independent space. A straightforward solution to this problem is to consider all behaviors of a certain type together, and to merge the activation levels connected to behaviors of the same type.

Let a behavior type be a set of all behaviors of the same type, e.g., $T_{Cross} = \{Cross(x) | x \text{ is a door}\}$. Then for a given B-plan P, the *merged activation level* in state s for behavior type T is computed by:

$$\bar{C_T}(s) = max_{B \in T} C_B(s) \tag{1}$$

where $C_B(s)$ denotes the activation level connected to the behavior B in each context-rule in P. The different $\bar{C_T}(s)$ change over time, as the state s changes. Figure 3 shows an example of the activation levels during execution of a navigation task.

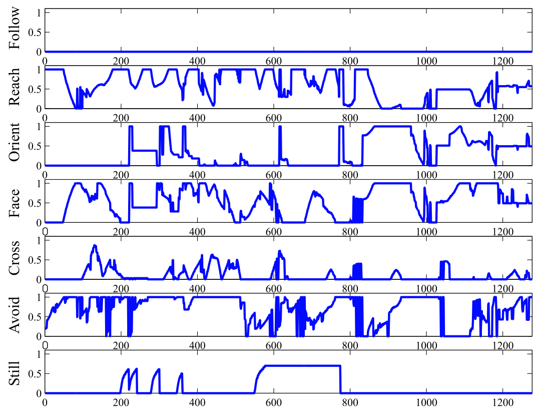

Figure 3: *The merged activation levels changing over time in a navigation scenario including several B-plans. The time is presented in 0.1 s.*

3.2 Feature Extraction

In signal processing, feature extraction is often performed by different frequency analysis methods, e.g., FFT-analysis and power density spectrum analysis [13]. These methods are applicable only when analysing periodic signals. Since the activation levels are not periodic, other feature extraction methods must be found.

We have investigated three different feature extraction methods, namely: the variance, the median and the energy. The feature extraction is performed in time windows on the merged activation levels of each behavior type. In the experiments the median was shown to be most suitable for fault detection, and the energy performed best in fault isolation. The median is calculated by sorting the samples within the window and selecting the middle value. The energy is calculated according to equation 2. For a given window ranging from n_0 to $n_0 + w$, let x'_i denote the energy over the activation level of the ith behavior type:

$$x'_i = \sum_{n=n_0}^{n_0+w} (x_i(n))^2, \tag{2}$$

where $x_i(n)$ denotes the merged activation level at time n. From the features that are extracted from the activation levels of the separate behavior types we form a feature vector $\mathbf{x}$. This vector is passed to the radial basis function networks.

3.3 Classification

In earlier work [12] a simple multiple linear regression method was used for the classification part. Here it was found that the type of faults were not easily separated by linear discrimination. Since artificial neural networks perform very well in pattern recognition problems [3], and have nonlinear discrimination power, we have chosen to use RBF networks for classification. The motivation of using a RBF networks instead of the most popular multi-layer feed forward network with a backpropagation training algorithm is that the training is much faster for the former [5].

As shown in figure 4 a RBF network can be described as a two-layer network. We have chosen an approach, presented in [5], where layer 1 (L_1) consists of a number of radial basis functions ϕ_j, where $j = 0, 1, \ldots, M$, with fixed centers and parameters in the nonlinearities. In contrary, the output layer (L_2) performs a linear combination on this new space and the only adjustable parameters are the weights of this linear combination. Mathematically the network can be described as a mapping function:

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj} \phi_j(\mathbf{x}), \tag{3}$$

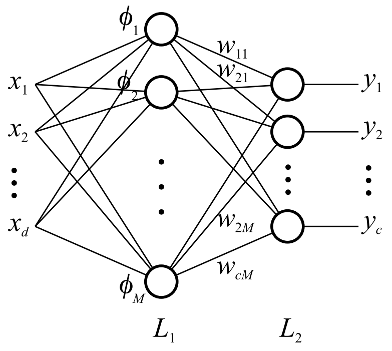

Figure 4: *The structure of a radial basis function network.*

where w_{kj} denote the weight between output neuron y_k and the radial basis function ϕ_j, and ϕ_0 is a bias with activation value fixed at $\phi_0 = 1$. The nonlinear basis function is in our case a Gaussian:

$$\phi_j(\mathbf{x}) \quad = \quad e^{-\frac{\|\mathbf{x}-\boldsymbol{\mu}_j\|^2}{2\sigma^2}}, \tag{4}$$

where $\boldsymbol{\mu}_j$ is the vector determining the center for basis function ϕ_j and σ is a constant that controls the smoothness properties of the interpolating function. Since the basis functions are considered fixed, the training of the output layer weights w_{kj} is equivalent to the training of a single-layer network, termed generalized linear discriminant. Hence we can optimize the weights, i.e., train our network, by minimizing a suitable error function such as the sum-of-squares error function:

$$E \quad = \quad \frac{1}{2}\sum_n \sum_k \left(y_k(\mathbf{x}^n) - t_k^n\right)^2, \tag{5}$$

where t_k^n is the target value, i.e., the correct classification, for output neuron y_k given input vector $\mathbf{x}^n$.

In this approach the network has no neurons initially. Instead they are added one at a time until the error E is decreased to a given level, or the maximum number of neurons is reached. The selection procedure is summarized as follows:

1. The error E is calculated for all input vectors $\mathbf{x}$.

2. The input vector $\mathbf{x}^n$ that gives the greatest error is found.

3. A new radial basis function and the corresponding weight are added. The new cluster center μ_j is set to be equal to the found input vector that gives the greatest error.

4. All the weights are updated to minimize the over all error E.

The static value σ was chosen manually in order to give the smallest error. When the network is fully trained it can be used for online fault detection or isolation.

4 Experiments

A large number of simulations were performed to evaluate our execution monitoring method. The simulation takes place in the Nomadic simulator [11], which includes simulation of sensor and positioning noise. Three different scenarios are tested in the experiments, namely:

1. success,

2. fault caused by a closed door, and

3. fault caused by a corrupted world-model.

The success scenario of course represents a successful execution of the task. In this scenario all doors are open and the robot achieves its given navigation task. In the closed door scenario one of the doors are found closed. Initially, the robot's world-model contains faulty information about the door, but this is corrected when the door is found closed. In the last scenario, the world-model is corrupted. Here, the robot has the faulty information that the door is opened, when it is actually closed. In this case the information about the door in the world-model is never updated.

Let us follow an example of a simulated scenario. Given the task: "Fetch books from room $R4$ and $R1$ and deliver them in room $R2$," TC creates three B-plans that are executed in sequence. Figure 5 shows an example of the simulation of the given task. In this corrupted world-model scenario the door leading to room $R2$ is closed and that is never updated in the world-model. Therefore the robot keeps on trying to cross this door.

All three scenarios were simulated 15 times, giving a total of 45 full simulated runs. Various amounts of obstacles were used, from no obstacles at all to highly cluttered rooms. In the two fault scenarios different doors were closed. Figure 6 shows the execution monitoring output of the simulated scenario in figure 5, and the merged activation levels are shown in figure 3.

5 Performance

It is of great importance that quantifiable criteria are used when evaluating the performance of a monitoring system. Inspired by [9] we have chosen to measure the following criteria:

- *Reaction speed*, that is the ability of the technique to detect faults with reasonably small delays after their arrivals.

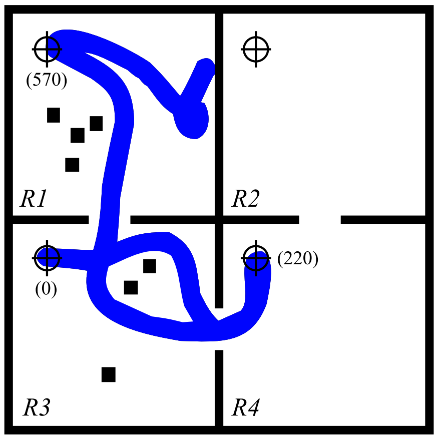

Figure 5: *An example of a simulation run showing the closed door with a corrupted world-model scenario. The numbers within parenthesis represent the time in 0.1 s.*

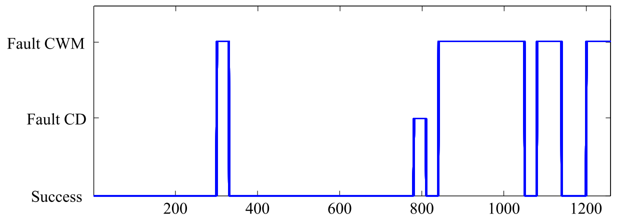

Figure 6: *The execution monitoring output giving an indication of success or the two faults: corrupted world-model or closed door.*

- *Robustness*, that is the ability of the technique to operate in the presence of noise, disturbances and modeling errors, with few false alarms.
- *Isolation performance*, that is the ability of the diagnostic system to distinguish faults on the physical properties of the plant, on the size of faults, noise, disturbances and model errors, and on the design of the algorithm.

The reaction speed depends on the type of feature extraction that is used. In our experiments the median within a window of 30 samples gave the best performance in fault detection. Since the sampling rate is 10 Hz the reaction speed for fault detection is 3 s. The best fault isolation performance was obtained by using the energy within a window of 40 samples, giving a reaction speed of 4 s.

Robustness is tested by introducing obstacles in the environment and adding odometry and sonar noise. Up to 5 % odometry bias and noise, and 10 % sonar noise was added. However, the greatest challenge to robustness was the presence of obstacles. We tried both very clean environments and highly cluttered ones.

The performance in robustness and isolation is calculated by evaluating the mean error $\bar{E}$ and standard error deviation s. To get an unbiased measure of these, 80 % of the data was randomly picked out for training and the rest was used for testing. This procedure was then repeated 10 times:

$$\bar{E} = \sum_{i=1}^{10} \frac{E_i}{10}, \qquad s = \sqrt{\sum_{i=1}^{10} \frac{(E_i - \bar{E})^2}{10-1}}, \tag{6}$$

where E_i denotes the error rate, i.e., the amount of misclassified samples, of radial basis function network i.

The performance of our execution monitor, based on the experiments presented in section 4, is summarized in table 1.

Table 1: *Monitoring performance measure.*

Reaction speed	Time	
Fault detection	3 s	
Fault isolation	4 s	
Robustness	$\bar{E}$	s
False alarms	0.0613	0.0500
Missed alarms	0.0969	0.0496
Isolation performance	$\bar{E}$	s
Closed door	0.0263	0.0812
Closed door with CWM	0.0297	0.0333

6 Conclusions

We have presented a model-free approach to execution monitoring of a behavior-based control system for mobile robots. Fault detection and fault isolation is performed by a pattern recognition approach using two radial basis function networks that are trained with a supervised learning algorithm by forcing the robot into success or known fault states. We have demonstrated the feasibility of this approach in a large number of simulated experiments.

A model-free approach is particularly suitable in systems where it is difficult to obtain adequate models. We also believe that the usability for a service robot will benefit from this approach. Most probably it is easier for the user to train the robot by telling what is success and fault, instead of compiling detailed predictive models in a formal language. An obvious disadvantage of using model-free monitoring is that it is not always feasible to force the robot into all known fault states. For example, it would be very expensive to learn from experience that falling down the stairs is a fault.

In future work we will implement and evaluate this monitor on a real robot. We would also like to try this approach on other behavior-based robot platforms, for example, a robot arm. The feature extraction and classification methods can be further developed to fit our needs.

Acknowledgments

The authors would like to thank their financial contributors, namely the Swedish Research Council and the Knowledge Foundation, that have partly financed this work.

References

[1] J.A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the AAAI Conference*, pages 83–88, Palo Alto, CA, USA, 1988.

[2] R.C. Arkin. *Behavior-Based Robotics.* MIT Press, Cambridge, MA, USA, 1998.

[3] C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, UK, 1995.

[4] M. Bjäreland. *Model-Based Execution Monitoring.* Ph.D. Thesis, Linköping University, Department of Computer and Information Science, 2001.

[5] S. Chen, C.F.N. Cowan, and P.M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[6] J.L. Fernández and R.G. Simmons. Robust execution monitoring for navigation plans. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 551–557, New York, NY, USA, 1998.

[7] R.E. Fikes. Monitored execution of robot plans produced by STRIPS. In *Proceedings of the IFIP Congress 71*, pages 189–194, Ljublijana, Yugoslavia, 1971.

[8] E. Gat. The three-layer architectures. In R.P. Bonasso, D. Kortenkamp, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. MIT Press, 1998.

[9] J.J. Gertler. *Fault Detection and Diagnosis in Engineering Systems.* Marcel Dekker, New York, NY, USA, 1998.

[10] K.B. Lamine and F. Kabanza. History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, pages 312–319, Los Alamitos, CA, USA, 2000.

[11] Nomadic Technologies, Inc., Mountain View, CA, USA. *User's Manual*, March 1997.

[12] O. Pettersson, L. Karlsson, and A. Saffiotti. Steps towards model-free execution monitoring on mobile robots. In *Swedish Workshop on Autonomous Robots (SWAR 02)*, pages 45–52, Stockholm, Sweden, 2002.

[13] J.G. Proakis and D.G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications.* Prentice-Hall, Upper Saddle River, NJ, USA, 1996.

[14] A. Saffiotti. *Autonomous Robot Navigation: A Fuzzy Logic Approach.* Ph.D. Thesis, Université Libre de Bruxelles, Faculté de Sciences Appliquées, 1998.

[15] 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.

[16] M.M. Veloso, M.E. Pollack, and M.T. Cox. Rationale-based monitoring for planning in dynamic environments. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 171–179, Menlo Park, CA, USA, 1998.

[17] B.C. Williams and P.P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the AAAI Conference*, pages 274–282, Menlo Park, CA, USA, 1996.

[18] L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.